**imageNeusoft Confidential**

File No. : ProjectNo.- D00-D01-T01-流水号

Project Name

软件需求规格说明书

（Software requirement specification）

**(Division)**

Neusoft Reach Automotive Technology Co., Ltd

**东软睿驰秘密(Neusoft Reach Secret)**

文件编号(File No.)：项目编号- D00-COP02-SW-T31-流水号

项目名称(项目编号)

软件需求规格说明书

(Software requirement specification)

版本(Version)：s.xy

日期(Date)：YYYY-MM-DD

|  |  |  |
| --- | --- | --- |
| 编制  Prepared by | 审核  Reviewed by | 批准  Approved by |
|  |  |  |

东软睿驰汽车技术(沈阳)有限公司

**(版权所有，翻版必究)**

Neusoft Reach Automotive Technology(Shenyang) Co., Ltd

**(Copyright by Reach Corporation, All Rights Reserved)**

变更履历(Change Log)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **修改编号**  **No.** | **版本**  **Version** | **修改内容**  **Contents Revised** | **状态**  **Status** | **修改人/日期**  **Reviser/Date** | **审批人/日期**  **Approve/Date** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

*填写说明：*

|  |  |
| --- | --- |
| ***No.*** | *顺序号* |
| ***Version*** | *每次修改需要添加一条变更履历,版本号需要升级。* |
| ***Contents Revised*** | *需要体现变更请求号(基线建立之后必须填写)、变更原因、变更位置(章节号)和变更内容。* |
| ***Status*** | *Draft->In review->Approved->Released*  *Approved和Released修改后状态变为Modified*  *Modified经过In review后才可以Approved*  *需要客户参与评审的文档才有Released状态* |
| ***Reviser/Date*** | *-* |
| ***Approve/Date*** | *Draft、In Review、Modified状态的文档变更履历不需要填写审批信息，Approved和Released状态的文档必须填写审批信息。如果审核和审批为不同人，都需要填写到此处。* |

**Contents**

[1 引言（Introduction） 1](#_Toc73546266)

[1.1 目的（Goal） 1](#_Toc73546267)

[1.2 范围（Scope） 1](#_Toc73546268)

[1.3 参考文档（Reference） 1](#_Toc73546269)

[1.4 术语及缩略语（Terms And Abbreviations） 1](#_Toc73546270)

[2 软件系统概述（Software System Overview） 1](#_Toc73546271)

[2.1 软件系统背景（Software System Background） 1](#_Toc73546272)

[2.2 软件系统目标（Software System Goal） 2](#_Toc73546273)

[2.3 外部关联 （External Association） 2](#_Toc73546274)

[3 功能需求（Functional Requirement） 2](#_Toc73546275)

[3.1 Persistency存储库和清单的关联关系 2](#_Toc73546276)

[3.1.1 [SWRD\_Per\_00001] Persistency清单里的关联关系 2](#_Toc73546277)

[3.1.2 [SWRD\_Per\_00003] Key Value Storages清单里的关联关系 4](#_Toc73546278)

[3.1.3 [SWRD\_Per\_00004] File Storages清单里的关联关系 5](#_Toc73546279)

[3.2 功能集群生命周期 6](#_Toc73546280)

[3.2.1 [SWRD\_Per\_00005] 初始化Persistency 6](#_Toc73546281)

[3.2.2 [SWRD\_Per\_00006] 关闭Persistency 6](#_Toc73546282)

[3.2.3 [SWRD\_Per\_00007] 初始化和关闭Persistency 概述 7](#_Toc73546283)

[3.3 并行访问Persistency数据 7](#_Toc73546284)

[3.3.1 [SWRD\_Per\_00008] 并行访问Persistency Data 7](#_Toc73546285)

[3.3.2 [SWRD\_Per\_00009] 文件句柄失效 8](#_Toc73546286)

[3.4 安全概念 9](#_Toc73546287)

[3.4.1 [SWRD\_Per\_00010] KVS存储库的加密解密和签名验证 9](#_Toc73546288)

[3.4.2 [SWRD\_Per\_00011] KVS中单个键值对的加密解密和签名验证 11](#_Toc73546289)

[3.4.3 [SWRD\_Per\_00012] FS存储数据的加密解密和签名验证 13](#_Toc73546290)

[3.4.4 [SWRD\_Per\_00013] FS中单个文件的加密解密和签名验证 15](#_Toc73546291)

[3.4.5 [SWRD\_Per\_00014] 数据的hash验证 17](#_Toc73546292)

[3.5 冗余概念 18](#_Toc73546293)

[3.5.1 [SWRD\_Per\_00015] 冗余的类型 CRC 18](#_Toc73546294)

[3.5.2 [SWRD\_Per\_00016] 冗余的类型 MOutOfN 19](#_Toc73546295)

[3.5.3 [SWRD\_Per\_00017] 冗余的类型 hash 20](#_Toc73546296)

[3.5.4 [SWRD\_Per\_00018] 冗余可以检测数据库是否损坏 20](#_Toc73546297)

[1. CRC检测： 21](#_Toc73546298)

[2.MOutOfN检测： 21](#_Toc73546299)

[3.5.5 [SWRD\_Per\_00019] 冗余检测出错误可以修复数据库 21](#_Toc73546300)

[3.6 安装和升级Persistency Data 22](#_Toc73546301)

[3.6.1 [SWRD\_Per\_00021] 存储数据的安装 23](#_Toc73546302)

[3.6.1.1 [SWRD\_Per\_00022] 安装Key Value Storage 23](#_Toc73546303)

[3.6.1.2 [SWRD\_Per\_00024] 安装File Storage 25](#_Toc73546304)

[3.6.2 存储数据的升级 27](#_Toc73546305)

[3.6.2.1 [SWRD\_Per\_00026] 版本号 27](#_Toc73546306)

[3.6.2.2 [SWRD\_Per\_00028] 升级策略 28](#_Toc73546307)

[3.6.2.3 [SWRD\_Per\_00031] 升级Key-value Storage 30](#_Toc73546308)

[3.6.2.4 [SWRD\_Per\_00032] 升级File Storage 31](#_Toc73546309)

[3.6.2.5 [SWRD\_Per\_00033] 用户自定义升级 33](#_Toc73546310)

[3.6.3 [SWRD\_Per\_00034] 存储数据升级成功后 33](#_Toc73546311)

[3.6.4 [SWRD\_Per\_00035] 升级失败后回退存储数据 34](#_Toc73546312)

[3.6.5 [SWRD\_Per\_00036] 卸载存储数据 35](#_Toc73546313)

[3.7 资源管理 35](#_Toc73546314)

[3.7.1 [SWRD\_Per\_00037] KVS库设置最小值 35](#_Toc73546315)

[3.7.2 [SWRD\_Per\_00038] KVS库设置最大值 36](#_Toc73546316)

[3.7.3 [SWRD\_Per\_00039] FS库设置最小值 36](#_Toc73546317)

[3.7.4 [SWRD\_Per\_00040] FS库设置最大值 37](#_Toc73546318)

[3.8 Key Value Storages支持的数据类型 37](#_Toc73546319)

[3.9 访问文件的附加信息 38](#_Toc73546320)

[4 非功能需求(Non-Functional Requirements) 39](#_Toc73546321)

[4.1 制约（Constraint） 39](#_Toc73546322)

[4.1.1 [SWRD\_ID]非功能需求1(Non-Function Requirement No.1) 39](#_Toc73546323)

[4.1.2 [SWRD\_ID]非功能需求2(Non-Function Requirement No.2) 40](#_Toc73546324)

[4.2 性能（Performance） 40](#_Toc73546325)

[4.2.1 [SWRD\_ID]非功能需求3(Non-Function Requirement No.3) 40](#_Toc73546326)

[4.2.2 [SWRD\_ID]非功能需求4(Non-Function Requirement No.4) 40](#_Toc73546327)

[4.3 质量（Quality） 41](#_Toc73546328)

[4.3.1 [SWRD\_ID]非功能需求5(Non-Function Requirement No.5) 41](#_Toc73546329)

[4.3.2 [SWRD\_ID]非功能需求6(Non-Function Requirement No.6) 41](#_Toc73546330)

[5 接口说明（API） 41](#_Toc73546331)

[5.1 接口头文件（API Header files） 41](#_Toc73546332)

[5.2 接口共同数据类型（API Common Data Types） 41](#_Toc73546333)

[5.3 Key Value Storage 接口定义 42](#_Toc73546334)

[5.3.1 [SWRD\_API\_Per\_00001] OpenKeyValueStorage 42](#_Toc73546335)

[5.3.2 [SWRD\_API\_Per\_00002] RecoverKeyValueStorage 43](#_Toc73546336)

[5.3.3 [SWRD\_API\_Per\_00003] ResetKeyValueStorage 44](#_Toc73546337)

[5.3.4 [SWRD\_API\_Per\_00004] GetCurrentKeyValueStorageSize 45](#_Toc73546338)

[5.3.5 keyvalueStorage 类 45](#_Toc73546339)

[5.3.5.1 [SWRD\_API\_Per\_00006] KeyValueStorage 46](#_Toc73546340)

[5.3.5.2 [SWRD\_API\_Per\_00008] operator= 47](#_Toc73546341)

[5.3.5.3 [SWRD\_API\_Per\_00010] ~KeyValueStorage() 48](#_Toc73546342)

[5.3.5.4 [SWRD\_API\_Per\_00011] GetAllKeys 48](#_Toc73546343)

[5.3.5.5 [SWRD\_API\_Per\_00012] KeyExists 49](#_Toc73546344)

[5.3.5.6 [SWRD\_API\_Per\_00013] GetValue 50](#_Toc73546345)

[5.3.5.7 [SWRD\_API\_Per\_00014] SetValue 51](#_Toc73546346)

[5.3.5.8 [SWRD\_API\_Per\_00015] RemoveKey 52](#_Toc73546347)

[5.3.5.9 [SWRD\_API\_Per\_00016] RecoverKey 53](#_Toc73546348)

[5.3.5.10 [SWRD\_API\_Per\_00017] ResetKey 54](#_Toc73546349)

[5.3.5.11 [SWRD\_API\_Per\_00018] RemoveAllKeys 55](#_Toc73546350)

[5.3.5.12 [SWRD\_API\_Per\_00019] SyncToStorage 55](#_Toc73546351)

[5.3.5.13 [SWRD\_API\_Per\_00020] DiscardPendingChanges 56](#_Toc73546352)

[5.4 File Storage 接口定义 57](#_Toc73546353)

[5.4.1 [SWRD\_API\_Per\_00021] OpenFileStorage 57](#_Toc73546354)

[5.4.2 [SWRD\_API\_Per\_00022] RecoverAllFiles 59](#_Toc73546355)

[5.4.3 [SWRD\_API\_Per\_00023] ResetAllFiles 60](#_Toc73546356)

[5.4.4 [SWRD\_API\_Per\_00024] GetCurrentFileStorageSize 61](#_Toc73546357)

[5.4.5 [SWRD\_API\_Per\_00025] OpenMode 61](#_Toc73546358)

[5.4.6 [SWRD\_API\_Per\_00026] operator | 62](#_Toc73546359)

[5.4.7 [SWRD\_API\_Per\_00027] operator|= 62](#_Toc73546360)

[5.4.8 [SWRD\_API\_Per\_00028] FileCreationState 63](#_Toc73546361)

[5.4.9 [SWRD\_API\_Per\_00029] FileModificationState 63](#_Toc73546362)

[5.4.10 [SWRD\_API\_Per\_00030] FileInfo 64](#_Toc73546363)

[5.4.10.1 [SWRD\_API\_Per\_00031] creationTime 64](#_Toc73546364)

[5.4.10.2 [SWRD\_API\_Per\_00032] modificationTime 64](#_Toc73546365)

[5.4.10.3 [SWRD\_API\_Per\_00033] accessTime 65](#_Toc73546366)

[5.4.10.4 [SWRD\_API\_Per\_00034] fileCreationState 65](#_Toc73546367)

[5.4.10.5 [SWRD\_API\_Per\_00035] fileModificationState 66](#_Toc73546368)

[5.4.11 [SWRD\_API\_Per\_00036] FileStorage 类 66](#_Toc73546369)

[5.4.11.1 [SWRD\_API\_Per\_00037] FileStorage 66](#_Toc73546370)

[5.4.11.2 [SWRD\_API\_Per\_00039] operator= 67](#_Toc73546371)

[5.4.11.3 [SWRD\_API\_Per\_00041] ~FileStorage 68](#_Toc73546372)

[5.4.11.4 [SWRD\_API\_Per\_00042] GetAllFileNames 68](#_Toc73546373)

[5.4.11.5 [SWRD\_API\_Per\_00043] DeleteFile 69](#_Toc73546374)

[5.4.11.6 [SWRD\_API\_Per\_00044] FileExists 70](#_Toc73546375)

[5.4.11.7 [SWRD\_API\_Per\_00045] RecoverFile 71](#_Toc73546376)

[5.4.11.8 [SWRD\_API\_Per\_00046] ResetFile 72](#_Toc73546377)

[5.4.11.9 [SWRD\_API\_Per\_00047] GetCurrentFileSize 73](#_Toc73546378)

[5.4.11.10 [SWRD\_API\_Per\_00048] GetFileInfo 74](#_Toc73546379)

[5.4.11.11 [SWRD\_API\_Per\_00049] OpenFileReadWrite 75](#_Toc73546380)

[5.4.11.12 [SWRD\_API\_Per\_00052] OpenFileReadOnly 78](#_Toc73546381)

[5.4.11.13 [SWRD\_API\_Per\_00055] OpenFileWriteOnly 82](#_Toc73546382)

[5.4.12 [SWRD\_API\_Per\_00058] Origin枚举 85](#_Toc73546383)

[5.4.13 [SWRD\_API\_Per\_00059] ReadAccessor 86](#_Toc73546384)

[5.4.13.1 [SWRD\_API\_Per\_00060] ReadAccessor 87](#_Toc73546385)

[5.4.13.2 [SWRD\_API\_Per\_00062] operator= 88](#_Toc73546386)

[5.4.13.3 [SWRD\_API\_Per\_00064] ~ReadAccessor 88](#_Toc73546387)

[5.4.13.4 [SWRD\_API\_Per\_00065] PeekChar 89](#_Toc73546388)

[5.4.13.5 [SWRD\_API\_Per\_00066] PeekByte 90](#_Toc73546389)

[5.4.13.6 [SWRD\_API\_Per\_00067] GetChar 91](#_Toc73546390)

[5.4.13.7 [SWRD\_API\_Per\_00068] GetByte 92](#_Toc73546391)

[5.4.13.8 [SWRD\_API\_Per\_00069] ReadText 93](#_Toc73546392)

[5.4.13.9 [SWRD\_API\_Per\_00071] ReadBinary 95](#_Toc73546393)

[5.4.13.10 [SWRD\_API\_Per\_00073] ReadLine 97](#_Toc73546394)

[5.4.13.11 [SWRD\_API\_Per\_00074] GetSize 98](#_Toc73546395)

[5.4.13.12 [SWRD\_API\_Per\_00075] GetPosition 98](#_Toc73546396)

[5.4.13.13 [SWRD\_API\_Per\_00076] SetPosition 99](#_Toc73546397)

[5.4.13.14 [SWRD\_API\_Per\_00077] MovePosition 100](#_Toc73546398)

[5.4.13.15 [SWRD\_API\_Per\_00078] IsEof 101](#_Toc73546399)

[5.4.14 [SWRD\_API\_Per\_00079] ReadWriteAccessor 类 102](#_Toc73546400)

[5.4.14.1 [SWRD\_API\_Per\_00080] SyncToFile 102](#_Toc73546401)

[5.4.14.2 [SWRD\_API\_Per\_00081] SetFileSize 103](#_Toc73546402)

[5.4.14.3 [SWRD\_API\_Per\_00082] WriteText 104](#_Toc73546403)

[5.4.14.4 [SWRD\_API\_Per\_00083] WriteBinary 105](#_Toc73546404)

[5.4.14.5 [SWRD\_API\_Per\_00084] operator<< 107](#_Toc73546405)

[5.5 Update and Remove of Persistency Data接口定义 107](#_Toc73546406)

[5.5.1 [SWRD\_API\_Per\_00085] RegisterApplicationDataUpdateCallback 107](#_Toc73546407)

[5.5.2 [SWRD\_API\_Per\_00086] UpdatePersistency 108](#_Toc73546408)

[5.5.3 [SWRD\_API\_Per\_00087] ResetPersistency 109](#_Toc73546409)

[5.6 Redundancy Handling接口定义 110](#_Toc73546410)

[5.6.1 [SWRD\_API\_Per\_00088] RecoveryReportKind 枚举 110](#_Toc73546411)

[5.6.2 [SWRD\_API\_Per\_00089] RegisterRecoveryReportCallback 111](#_Toc73546412)

[5.7 Handle Classes接口定义 112](#_Toc73546413)

[5.7.1 [SWRD\_API\_Per\_00090] SharedHandle class 112](#_Toc73546414)

[5.7.1.1 [SWRD\_API\_Per\_00091] SharedHandle 113](#_Toc73546415)

[5.7.1.2 [SWRD\_API\_Per\_00093] operator= 113](#_Toc73546416)

[5.7.1.3 [SWRD\_API\_Per\_00095] operator bool 114](#_Toc73546417)

[5.7.1.4 [SWRD\_API\_Per\_00096] operator-> 115](#_Toc73546418)

[5.7.1.5 [SWRD\_API\_Per\_00098] operator\* 116](#_Toc73546419)

[5.7.2 [SWRD\_API\_Per\_00100] UniqueHandle 类 117](#_Toc73546420)

[5.7.2.1 [SWRD\_API\_Per\_00101] UniqueHandle 117](#_Toc73546421)

[5.7.2.2 [SWRD\_API\_Per\_00103] operator= 118](#_Toc73546422)

[5.7.2.3 [SWRD\_API\_Per\_00105] operator bool 119](#_Toc73546423)

[5.7.2.4 [SWRD\_API\_Per\_00106] operator-> 119](#_Toc73546424)

[5.7.2.5 [SWRD\_API\_Per\_00108] operator\* 120](#_Toc73546425)

[5.8 Errors接口定义 121](#_Toc73546426)

[5.8.1 [SWRD\_API\_Per\_00110] PerErrc 枚举 121](#_Toc73546427)

[5.8.2 [SWRD\_API\_Per\_00111] GetPerDomain 122](#_Toc73546428)

[5.8.3 [SWRD\_API\_Per\_00112] MakeErrorCode 123](#_Toc73546429)

[5.8.4 [SWRD\_API\_Per\_00113] PerException 类 123](#_Toc73546430)

[5.8.4.1 [SWRD\_API\_Per\_00114] PerException 124](#_Toc73546431)

[5.8.5 [SWRD\_API\_Per\_00115] PerErrorDomain 类 124](#_Toc73546432)

[5.8.5.1 [SWRD\_API\_Per\_00116] Errc 124](#_Toc73546433)

[5.8.5.2 [SWRD\_API\_Per\_00117] Exception 125](#_Toc73546434)

[5.8.5.3 [SWRD\_API\_Per\_00118] PerErrorDomain 125](#_Toc73546435)

[5.8.5.4 [SWRD\_API\_Per\_00119] Name 126](#_Toc73546436)

[5.8.5.5 [SWRD\_API\_Per\_00120] Message 126](#_Toc73546437)

[5.8.5.6 [SWRD\_API\_Per\_00121] ThrowAsException 126](#_Toc73546438)

[附录A- 信息定义 127](#_Toc73546439)

[附录B- 配置信息 129](#_Toc73546440)

# 引言（Introduction）

## 目的（Goal）

编写本文的目的，是为了Persistency模块提供详细的软件需求的定义，给开发人员和测试人员提供设计和测试执行的标准。

## 范围（Scope）

本文使用者： 开发人员、测试人员、PSM、TeamLeader、TestLeader和产品负责人。

本文使用方法：

* 对于开发人员、根据本文中定义的功能/非功能要求进行后续的设计。
* 对于测试人员、通过理解本文中的要求，进行测试用例的制作和后续测试执行。

对于PSM、TeamLeader、TestLeader和产品负责人、来判断需求理解的正确性。

## 参考文档（Reference）

|  |  |  |
| --- | --- | --- |
| **序号（No.）** | **文档名（Document Name）** | **版本名（Revision）** |
| 1 | 《AUTOSAR\_SWS\_ Persistency》 | R-2111 |
| 2 | 《AUTOSAR\_TPS\_ManifestSpecification》 | R-2111 |
|  |  |  |

## 术语及缩略语（Terms And Abbreviations）

|  |  |  |
| --- | --- | --- |
| **编号** | **缩写** | **说明** |
| 1 | Process | Process是Executable的加载实例，是Application的一部分。 |
| 2 | Per | Persistency(持久化) |
| 3 | UCM | 升级模块 |
| 4 | KVS | Key Value Storage键值库 |
| 5 | FS | File Storage文件库 |

# 软件系统概述（Software System Overview）

## 软件系统背景（Software System Background）

为了满足自适应应用程序对数据进行持久化存储，Persistency模块能够提供方便的存储接口。

持久性为自适应应用程序提供了将信息存储在机器的非易失性存储器中的机制。该数据在启动和点火周期内可用。持久化提供两种存储方式，键值存储和文件存储，通过上位机可配置存储的初始化信息。也可以通过提供的接口，创建键值和文件，并对存储的内容进行修改等操作。可以通过升级功能来升级持久化存储的数据。

## 软件系统目标（Software System Goal）

Persistency模块开发的目标是根据AUTOSAR规范，为自适应应用程序提供了将信息存储在机器的非易失性存储器中的机制，并包含了多种保证数据安全的特性。

## 外部关联 （External Association）

根据规范定义，Persistency依赖加密模块，Log模块和UCM模块。

# 功能需求（Functional Requirement）

## Persistency存储库和清单的关联关系

Persistency功能集群有两种访问存储的机制：

Key Value Storages键值库

File Storages文件库

一个自适应应用程序可以使用多个键值库和文件库的组合。

### [SWRD\_Per\_00001] Persistency清单里的关联关系

Executable其中的一部分AdaptiveApplicationSwComponentTypes，

Model有两部分：其中一部分是Application的部署信息：PersistencyKeyValueStorage和PersistencyFileStorage。

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00001 |
| **Type** | valid |
| **Description** | 1. API规定类ara::per::KeyValueStorage和ara::per::FileStorage访问Key Value Storage和File Storage。 2. 基于PortPrototype的访问类型，Key Value Storage或者File Storage可以被以下三种方式访问：   Read Only：RPortPrototype  Read/Write：PRPortPrototype  Write Only：PPortPrototype   1. 用PersistencyPortPrototypeToDeploymentMapping来关联Process和PersistencyDeployment。   PersistencyDeployment基类派生PersistencyKeyValueStorage和PersistencyFileStorage。  PersistencyDeploymentElement基类派生PersistencyKeyValuePair和PersistencyFile。  PersistencyPortPrototypeToDeploymentMapping基类派生PersistencyPortPrototypeToKeyValueStorageMapping和PersistencyPortPrototypeToFileStorageMapping。 |
| **Upstream ID** |  |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 上位机配置：  上面需求第2条，配置方法如下，其中portKVS和portFS可以选择是只读或可写的口。    上面需求第3条，配置方法如下： |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **CR** |  |
| **Risk** | 无 |
| **Change Type** | 不变 |

另一部分是Application的设计信息PersistencyKeyValueStorageInterface和PersistencyFileStorageInterface。

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00002 |
| **Type** | valid |
| **Description** | 1. API规定持有类ara::per::KeyValueStorage和ara::per::FileStorage访问Key Value Storage和File Storage。   PersistencyInterface的PortPrototype的shortName作为访问上面全局方法的入参。   1. 基于PortPrototype的访问类型，Key Value Storage或者File Storage可以被以下三种方式访问：   Read Only：RPortPrototype  Read/Write：PRPortPrototype  Write Only：PPortPrototype   1. 用PersistencyPortPrototypeToDeploymentMapping来关联Process和PersistencyInterface 2. PersistencyInterface基类派生PersistencyKeyValueStorageInterface和PersistencyFileStorageInterface。   PersistencyInterfaceElement基类派生PersistencyDataElement和PersistencyFileElement， |
| **Upstream ID** |  |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | PersistencyKeyValueStorageInterface和PersistencyFileStorageInterface侧和PersistencyKeyValueStorage和PersistencyFileStorage属重复操作，功能效果一致，用户配置冗余，目前Interface侧隐藏。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
| **CR** |  |
| **Risk** | 无 |
| **Change Type** | 不变 |

### [SWRD\_Per\_00003] Key Value Storages清单里的关联关系

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00003 | |
| **Type** | Valid | |
| **Description** | 1. 在Application设计时，每个Key Value Storage被PersistencyKeyValueStorageInterface的PortPrototype代表，PersistencyKeyValueStorage包含部署信息。 2. 每个Key Value Storage可以包括多个Key Value Pairs，在Adaptive Application运行的时候Key Value Pairs可以通过API增加或者删除。   在安装或者升级Adaptive Application的时候，Key Value Storage 预定义的Key Value Pairs可以部署进去。这个由UCM模块触发。  Adaptive Application的软件包会提供安装或更新需要的部署信息和数据。   1. Application的Key Value Storage的安装和部署信息，PersistencyKeyValueStorageInterface提供的PortPrototype类型、PersistencyKeyValueStorage和Process由PersistencyPortPrototypeToKeyValueStorageMapping来关联。 | |
| **Upstream ID** |  | |
| **Dependencies** | 无 | |
| **Verification method** | 测试 |
| **Verification Criteria** | PersistencyKeyValueStorageInterface没有实现，由AdaptiveApplicationSwComponentType直接创建PortPrototype。  在上位机配置初始数据：参照NeuSAR\_aCore\_PER\_使用手册。  用上位机配置初始数据，用UCM安装后，调用API对键值对进行访问，例如增加或删除。  上面需求第3条，关联方式配置方法如下： | |
| **ASIL** | 无 | |
| **Status** | Draft | |
| **Priority** | H | |
| **CR** |  | |
| **Risk** | 无 | |
| **Change Type** | 不变 | |

### [SWRD\_Per\_00004] File Storages清单里的关联关系

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00004 |
| **Type** | Valid |
| **Description** | 1. 在Application设计阶段，每个File Storage被PersistencyFileStorageInterface的PortPrototype关联，PersistencyFileStorage包含部署信息。 2. 每个File Storage可以包括多个files。在Adaptive Application运行的时候files可以通过API增加或者删除。   在安装或者升级Adaptive Application的时候，File Storage预定义的files可以部署进去。这个由UCM模块触发。  Adaptive Application的软件包会提供安装或更新需要的部署信息和数据。   1. 在Application的File Storage的安装和部署信息是AdaptiveApplicationSwComponentType的PortPrototype提供的，PersistencyFileStorage和Process由PersistencyPortPrototypeToFileStorageMapping来关联。 |
| **Upstream ID** |  |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | PersistencyFileStorageInterface没有实现，由AdaptiveApplicationSwComponentType直接创建PortPrototype。  在上位机配置初始数据：参照NeuSAR\_aCore\_PER\_使用手册。  用上位机配置初始数据，用UCM安装后，调用API对文件进行访问，例如增加或删除。  上面需求第3条，关联方式配置方法如下： |
| **Risk** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **CR** |  |
| **Change Type** | 不变 |

## 功能集群生命周期

Application可以用ARA接口开始和关闭所有的功能集群，使用ara::core::Initialize和ara::core::Deinitialize。

Application不可以在ara::core::Initialize前和ara::core::Deinitialize后调用任何API，Per应该处理防止这些情况。

### [SWRD\_Per\_00005] 初始化Persistency

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00005 |
| **Type** | Invalid |
| **Description** | 当调用了ara::core::Initialize，Per才可以开始访问清单中定义的Key-Value Storages 和File Storages。 |
| **Upstream ID** | SWS\_PER\_00408、RS\_PER\_00018 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 无 |
| **ASIL** |  |
| **Status** | Draft |
| **Priority** | L |
| **Risk** |  |
| **CR** |  |
| **Change Type** | 新增 |

### [SWRD\_Per\_00006] 关闭Persistency

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00006 |
| **Type** | Invalid |
| **Description** | 当调用了ara::core::Deinitialize，持久性将隐式确保所有文件存储的所有打开文件都被持久保存，并且数据已经被ara::per::ReadWriteAccessor::SyncToFile，访问文件的句柄被释放。并且所有键值存储中未持久存储的值都将被删除，就像调用了ara::per::KeyValueStorage::DiscardPendingChanges一样。此后，所有访问结构应被释放。 |
| **Upstream ID** | SWS\_PER\_00409、RS\_PER\_00018 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 无 |
| **ASIL** |  |
| **Status** | Draft |
| **Priority** | L |
| **Risk** |  |
| **CR** |  |
| **Change Type** | 不变 |

### [SWRD\_Per\_00007] 初始化和关闭Persistency 概述

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00007 |
| **Type** | Invalid |
| **Description** | 当在ara::core::Initialize前和ara::core::Deinitialize后调用时返回error kNotInitialized。 |
| **Upstream ID** | SWS\_PER\_00410、RS\_PER\_00018 |
| **Dependencies** |  |
| **ASIL** |  |
| **Status** | Draft |
| **Priority** | L |
| **Verification method** | 测试 |
| **Verification Criteria** | 无 |
| **Risk** |  |
| **CR** |  |
| **Change Type** | 不变 |

## 并行访问Persistency数据

对于一个Process，Persistency数据只作为本地化的存储，Persistency数据不能给多个Processes共享，同一个Executable也不行。

背景是Persistency不能提供除CM以外的通信机制。

如果需要让多个Processes访问到Persistency数据，需要Application designer提供Service Interfaces去通信。

### [SWRD\_Per\_00008] 并行访问Persistency Data

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00008 |
| **Type** | Valid |
| **Description** | 1. 可以从同一个Application的一个进程的多个线程访问到Per数据，通过ara::per::OpenKeyValueStorage 和ara::per::OpenFileStorage可以返回共享句柄ara::per::SharedHandle，通过共享句柄可以在多个线程访问同一个KVS或者FS。 2. ara::per::RecoverKeyValueStorage，ara::per::ResetKeyValueStorage，ara::per::RecoverAllFiles ，ara::per::ResetAllFiles接口需要在Key Value Storage或File Storage关闭时才能生效。 3. 在多个线程里可以同时对KVS中的单个key操作，因为setValue()和getValue()操作的是原数据，像ara::per::KeyValueStorage::RemoveKey，ara::per::KeyValueStorage::RemoveAllKeys，ara::per::KeyValueStorage::SyncToStorage，和ara::per::KeyValueStorage::DiscardPendingChanges都是。   在多个线程里，对于FS中的文件不能共享访问，多个线程同时操作一个文件，对于文件操作位置会出错。像ara::per::FileStorage::DeleteFile, ara::per::FileStorage::RecoverFile, 和ara::per::FileStorage::ResetFile 不能在文件打开时操作。 |
| **Upstream ID** | SWS\_PER\_00309、RS\_PER\_00001 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 1. 可以被同一个Process的多个线程访问验证方法：   调用API OpenKeyValueStorage返回的句柄，在多个线程用这个句柄访问键值库的数据。调用OpenFileStorage返回的句柄，在多个线程中用这个句柄访问文件库的数据。  auto result = ::ara::per::OpenKeyValueStorage("/log/perTestWriter.json");  if (!result) { logger.LogError()<<\_\_FILE\_\_<<":"<<\_\_LINE\_\_<<":"<<“open error”;  return ;  }  std::cout <<"-- KVS testPerformance open kvs "<< std::endl;  auto db = std::move(result).Value();  db作为共享句柄用于调用API访问数据库。   1. 多个线程可以同时通过API Setvalue/Getvalue对单个键值对进行设置和读取。 2. 在一个数据库被损坏没有打开成功或者数据库关闭的情况下，ara::per::RecoverKeyValueStorage，ara::per::ResetKeyValueStorage，ara::per::RecoverAllFiles ，ara::per::ResetAllFiles接口生效。   数据库打开的句柄出了作用域句柄失效，数据库关闭。   1. 在多个线程里，对FS中的同一个文件共享访问，读取会报错。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 不变 |

### [SWRD\_Per\_00009] 文件句柄失效

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00009 |
| **Type** | Valid |
| **Description** | 当File Storage关闭时，关联的打开的文件库的ara::per::SharedHandle也就失效了，文件库也就关闭了。访问文件的ara::per::UniqueHandle也失效了。 |
| **Upstream ID** | SWS\_PER\_00425[2111A]、RS\_PER\_00001 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 打开失败时，文件句柄不可用。  关闭库时，文件句柄需要失效 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 新增，P3 |

## 安全概念

向Key Value Storage或File Storage存入数据时可以加密和签名，读出数据时解密和验证签名。

是否应用加密或身份验证，是在部署时决定的。

键值库、键值库的键、文件库或文件库的文件在创建存储时进行加密，打开存储库时进行解密。

用于存储的身份验证的签名在打开存储时也需要被验证，并在安装期间、保存键值存储和文件存储时进行计算。

Per应该使用加密模块API来加密和解密和创建和校验签名hash。使用的算法和关键字从CryptoKeySlot参考PersistencyDeploymentToCryptoKeySlotMapping或者PersistencyDeploymentElementToCryptoKeySlotMapping，或者访问密码学API。

### [SWRD\_Per\_00010] KVS存储库的加密解密和签名验证

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_000010 |
| **Type** | Valid |
| **Description** | 如果清单中配置了PersistencyDeploymentToCryptoKeySlotMapping.keySlotUsage设置成encryption或者verification，  例如下图中6，Per在将相关数据存储到持久化内存之前应该对其进行加密或签名。读取相关数据时，将其解密或验证签名。  注：   1. 加密模块配置的加密项 2. Per配置的键值对 3. 单个键值对和加密的关联 4. 库和加密模块的关联 5. 库和加密关联时，关联的加密项 6. 库和加密关联时，关联的加密方法，这里枚举了两种方法 7. encryption 加密 8. verification 签名 9. 库和加密关联时，关联的数据库 |
| **Upstream ID** | SWS\_PER\_00210、SWS\_PER\_00211、SWS\_PER\_00449[2111A]、SWS\_PER\_00450[2111A]、RS\_PER\_00005、RS\_PER\_00010、TPS\_MANI\_03260、TPS\_MANI\_03273 |
| **Dependencies** | 加密模块 |
| **Verification method** | 测试 |
| **Verification Criteria** | 1. 上位机配置PersistencyDeploymentToCryptoKeySlotMapping的keySlotUsage设成：encryption，关联的persistency Deploment的数据库数据写入需要加密，加密使用的是Crypto key Slot关联的配置项。 2. 上位机配置PersistencyDeploymentToCryptoKeySlotMapping的keySlotUsage设成：verification，关联的persistency Element的数据库数据写入需要签名，签名使用的是Crypto key Slot关联的配置项。 3. 数据会根据keySlotUsage配置的encryption或verification进行加密或签名。encryption和verification对应的API需要加密模块提供。 4. 由于增加了加密和签名，上位机生成的清单也需要增加项，标记数据库的加密和签名方式。   persistency\_db\_manifest.json中，   1. 只实现库级别的加密和签名：   /\*\*\*库级别的加密设置\*\*\*/  "key": "keySlotUsage",  "value": {  "object[]": [  {  "key": " encryption ",  "value": {  "string": CryptoKeySlot001  }  }  ]  }  /\*\*\*库级别的签名设置\*\*\*/  "key": "keySlotUsage",  "value": {  "object[]": [  {  "key": " verification",  "value": {  "string": CryptoKeySlot011  }  }  ]  }   1. 写入数据库的数据展示形式。   KVS数据库：   1. 存储形式：   Key（密文）：“密文”  Value（密文）：“密文”  如果有签名，  SignNum（密文）：“签名码”   1. 这种只支持设置了整个库的加密策略，键值库中的所有键值对都根据库的加密策略做加密处理。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 新增，加密**P1，**签名**P3** |

### [SWRD\_Per\_00011] KVS中单个键值对的加密解密和签名验证

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00011 |
| **Type** | Valid |
| **Description** | 如果清单中配置了PersistencyDeploymentElementToCryptoKeySlotMapping.keySlotUsage设置成encryption或者verification，  例如下图中6，Per在将相关数据存储到持久化内存之前应该对其进行加密或签名。读取相关数据时，将其解密或验证签名。  注：   1. 加密模块配置的加密项 2. Per配置的键值对 3. 单个键值对和加密的关联 4. 库和加密的关联 5. 单个键值对和加密关联时，关联的加密项 6. 单个键值对和加密关联时，关联的加密方法，这里枚举了两种方法   (1) encryption 加密  (2) verification 签名  7. 单个键值对和加密关联时，关联的键值对 |
| **Upstream ID** | SWS\_PER\_00210、SWS\_PER\_00211、SWS\_PER\_00449[2111A]、SWS\_PER\_00450[2111A]、RS\_PER\_00005、RS\_PER\_00010、TPS\_MANI\_03260、TPS\_MANI\_03273 |
| **Dependencies** | 加密模块 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification method** | 测试 |
| **Verification Criteria** | 1. 上位机配置PersistencyDeploymentElementToCryptoKeySlotMapping的keySlotUsage设成：encryption，关联的persistency Deploment Element的数据库数据写入需要加密，加密使用的是Crypto key Slot关联的配置项。 2. 上位机配置PersistencyDeploymentElementToCryptoKeySlotMapping的keySlotUsage设成：verification，关联的persistency Deploment Element的数据库数据写入需要签名，签名使用的是Crypto key Slot关联的配置项。 3. 数据会根据keySlotUsage配置的encryption或verification进行加密或签名。encryption和verification对应的API需要加密模块提供。 4. 由于增加了加密和签名，上位机生成的清单也需要增加项，标明哪些键值是用的哪种加密和签名方式。或者遵循的是库的加密和签名方式。   persistency\_db\_manifest.json中，   1. 库级别的加密和签名：   /\*\*\*库级别的加密设置\*\*\*/  "key": "keySlotUsage",  "value": {  "object[]": [  {  "key": " encryption ",  "value": {  "string": CryptoKeySlot001  }  }  ]  }  /\*\*\*库级别的签名设置\*\*\*/  "key": "keySlotUsage",  "value": {  "object[]": [  {  "key": " verification",  "value": {  "string": CryptoKeySlot011  }  }  ]  }   1. 单个键的加密和签名：   /\*\*\*单个键的加密设置\*\*\*/  "key": " encryption ",  "value": {  "object[]": [  {  "key": " CryptoKeySlot002",  "value": {  "string": " KVPair10, KVPair11 "  }  },  {  "key": " CryptoKeySlot003",  "value": {  "string": " KVPair12, KVPair13 "  }  }  ]  }  /\*\*\*单个键的签名设置\*\*\*/  "key": " verification",  "value": {  "object[]": [  {  "key": " CryptoKeySlot004",  "value": {  "string": " KVPair20, KVPair21 "  }  },  {  "key": " CryptoKeySlot005",  "value": {  "string": " KVPair22, KVPair23 "  }  }  ]  }   1. 写入数据库的数据展示形式。   KVS数据库：   1. 存储形式：   Key（明文）：“密文”  Value（明文）：“密文”   1. 如果设置了单个item的加密或签名，同时也设置了整个库的策略，两种加密或签名方式可能不是一样的，单个键值对根据Element的加密或签名进行处理，库整体加密或签名时不再做了。 |
| **Risk** | 无 |
| **Change Type** | 新增**P3** |

### [SWRD\_Per\_00012] FS存储数据的加密解密和签名验证

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00012 |
| **Type** | Valid |
| **Description** | 如果清单中配置了PersistencyDeploymentToCryptoKeySlotMapping.keySlotUsage设置成encryption或verification，  例如下图红框标识，Per在将相关数据存储到持久化内存之前应该对其进行加密或签名。读取相关数据时，对其解密或对签名进行验证。 |
| **Upstream ID** | SWS\_PER\_00210、SWS\_PER\_00211、SWS\_PER\_00449[2111A]、SWS\_PER\_00450[2111A]、RS\_PER\_00005、RS\_PER\_00010、TPS\_MANI\_03260、TPS\_MANI\_03273 |
| **Dependencies** | 加密模块 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 1. 上位机配置PersistencyDeploymentToCryptoKeySlotMapping的keySlotUsage设成：encryption，关联的persistency Deploment的数据库数据写入需要加密，加密使用的是Crypto key Slot关联的配置项。 2. 上位机配置PersistencyDeploymentToCryptoKeySlotMapping的keySlotUsage设成：verification，关联的persistency Element的数据库数据写入需要签名，签名使用的是Crypto key Slot关联的配置项。 3. 数据会根据keySlotUsage配置的encryption或verification进行加密或签名。encryption和verification对应的API需要加密模块提供。 4. 由于增加了加密和签名，上位机生成的清单也需要增加项，标记数据库的加密和签名方式。   persistency\_db\_manifest.json中，   1. 只实现库级别的加密和签名：   /\*\*\*库级别的加密设置\*\*\*/  "key": "keySlotUsage",  "value": {  "object[]": [  {  "key": " encryption ",  "value": {  "string": CryptoKeySlot001  }  }  ]  }  /\*\*\*库级别的签名设置\*\*\*/  "key": "keySlotUsage",  "value": {  "object[]": [  {  "key": " verification",  "value": {  "string": CryptoKeySlot011  }  }  ]  }   1. 最后写入数据库文件的数据展示形式。 2. FS数据库： 3. 存储形式：   FS数据库文件夹下的所有文件整体加密。库文件夹下的所有文件内容以密文形式存储。   1. 文件生成对应的签名码存储到一个键值库中。 2. 在OpenFileReadWrite时，将open的文件解密到“.”开头的临时文件中，增加API closeFile(“文件路径+文件名”)，用户手动调用closeFile对文件进行加密。加密时考虑文件过大，分段加密。 3. 文件做加密影响性能。 |
| **Risk** |  |
| **Change Type** | 新增 **加密P1** **签名P3** |

### [SWRD\_Per\_00013] FS中单个文件的加密解密和签名验证

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00013 |
| **Type** | Valid |
| **Description** | 如果清单中配置了PersistencyDeploymentElementToCryptoKeySlotMapping.keySlotUsage设置成encryption或verification，  例如下图红框标识，Per在将相关数据存储到持久化内存之前应该对其进行加密或签名。读取相关数据时，对其解密或对签名进行验证。 |
| **Upstream ID** | User\_Defined\_PER\_00023、SWS\_PER\_00210[2011A]、SWS\_PER\_00211[2011A]、SWS\_PER\_00449[2011A]、SWS\_PER\_00450[2011A]、RS\_PER\_00005、RS\_PER\_00010、TPS\_MANI\_03260、TPS\_MANI\_03273 |
| **Dependencies** | 加密模块 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 1. 上位机配置PersistencyDeploymentElementToCryptoKeySlotMapping的keySlotUsage设成：encryption，关联的persistency Deploment的数据库数据写入需要加密，加密使用的是Crypto key Slot关联的配置项。 2. 上位机配置PersistencyDeploymentElementToCryptoKeySlotMapping的keySlotUsage设成：verification，关联的persistency Element的数据库数据写入需要签名，签名使用的是Crypto key Slot关联的配置项。 3. 数据会根据keySlotUsage配置的encryption或verification进行加密或签名。encryption和verification对应的API需要加密模块提供。 4. 由于增加了加密和签名，上位机生成的清单也需要增加项，标记数据库的加密和签名方式。   persistency\_db\_manifest.json中，   1. 库级别的加密和签名：   /\*\*\*库级别的加密设置\*\*\*/  "key": "keySlotUsage",  "value": {  "object[]": [  {  "key": " encryption ",  "value": {  "string": CryptoKeySlot001  }  }  ]  }  /\*\*\*库级别的签名设置\*\*\*/  "key": "keySlotUsage",  "value": {  "object[]": [  {  "key": " verification",  "value": {  "string": CryptoKeySlot011  }  }  ]  }   1. 单个文件的加密和签名：   /\*\*\*单个文件的加密设置\*\*\*/  "key": " encryption ",  "value": {  "object[]": [  {  "key": " CryptoKeySlot002",  "value": {  "string": " 1.txt, 2.txt "  }  },  {  "key": " CryptoKeySlot003",  "value": {  "string": " 3.txt, 4.txt "  }  }  ]  }  /\*\*\*单个文件的签名设置\*\*\*/  "key": " verification",  "value": {  "object[]": [  {  "key": " CryptoKeySlot004",  "value": {  "string": " 5.txt, 6.txt "  }  },  {  "key": " CryptoKeySlot005",  "value": {  "string": " 7.txt , 8.txt "  }  }  ]  }   1. 最后写入数据库文件的数据展示形式。 2. FS数据库： 3. 存储形式：   FS数据库文件夹下的单个文件整体加密。文件整体内容一密文形式存储。   1. 单个文件生成对应的签名码存储到一个键值库中。 2. 文件库中的文件可以设置单个文件的签名和/或者整个文件库的签名。两种加密方式可能不是一样的。单个文件根据Element的签名规则进行签名生成签名码，库整体签名时不再签名。 3. 在OpenFileReadWrite时，将open的文件解密到“.”开头的临时文件中，增加API closeFile(“文件路径+文件名”)，用户手动调用closeFile对文件进行加密。加密时考虑文件过大，分段加密。 4. 文件做加密影响性能。 |
| **Risk** |  |
| **Change Type** | 新增**P3** |

### [SWRD\_Per\_00014] 数据的hash验证

对于只读的Key Value Storage或File Storage，只在安装的时候做加密。

只读的Key-Value Storage或者File Storage用签名hash验证，在manifest中标记PersistencyDeploymentToCryptoKeySlotMapping.verificationHash或者PersistencyDeploymentElementToCryptoKeySlotMapping.verificationHash时，在安装时需要做哈希计算。

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00014 |
| **Type** | InValid |
| **Description** | 如果清单中有PersistencyDeploymentToCryptoKeySlotMapping.verificationHash 或PersistencyDeploymentElementToCryptoKeySlotMapping.verificationHash是有效的话，持久化集群应该使用hash来验证应该数据。 |
| **Upstream ID** | SWS\_PER\_00451[2111A]、RS\_PER\_00005、RS\_PER\_00010 |
| **Dependencies** | 加密模块 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 待加密模块确认后再定  在安装时计算hash值存储到一个键值库中每一个键值对value的后面，文件库根据文件名存储到键值库里。  OpenStorage时用hash值验证，如正确可正常使用。否则没有访问权限。 |
| **Risk** |  |
| **Change Type** | 新增**P3** |

## 冗余概念

Persistency功能集群需要保证存储的数据准确。

可以通过计算存储数据的CRC或hash value或者创建冗余备份。有效的是创建一些冗余副本。

是否使用冗余可配置。

软件设计者可以通过PersistencyInterface.redundancy去申请冗余或者使用PersistencyInterface.redundancyHandling 来预选用哪种方法。

由PersistencyDeployment.redundancyHandling.聚合的PersistencyRedundancyHandling设置。

范围是对于Key Value Storage 和keys,或者一个File Storage 和Files。

冗余的类型有三种

CRC

MOutOfN

hash值

### [SWRD\_Per\_00015] 冗余的类型 CRC

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00015 |
| **Type** | Valid |
| **Description** | Persistency为每个Key Value Storage 和每个File Storage保存冗余信息。  由PersistencyInterface的PortPrototype表示，其中PersistencyInterface.redundancy设置为redundant或redundanperelement。  如果设置了PersistencyRedundancyCrc，Persistency集群在存入数据时应该计算CRC值。读出的时候需要使用CRC去检查。  Persistency应该使用PersistencyRedundancyCrc.algorithmFamily定义的算法计算CRC值，位宽被PersistencyRedundancyCrc.length定义。  （存入数据：在存入Key Value Storage，Key Value Storage里的key, File Storage, 或者File Storage里的一个文件，依赖PersistencyDeployment. redundancyHandling.scope） |
| **Upstream ID** | SWS\_PER\_00317[2111M]、SWS\_PER\_00318[2111M]、SWS\_PER\_00439[2111A]、RS\_PER\_00008、RS\_PER\_00009、RS\_PER\_00010 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 冗余的属性配置到PersistencyKeyValueDatabase下。  上位机配置CRC，如图PersistencyRedundancyCrc    键值库文件中的键值对下会生成CRC的checksum值。  文件库会生成一个键值库文件中对应文件名和CRC的checkSum值。  CRC和M/N单独使用，不可同时配置。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 不变 |

### [SWRD\_Per\_00016] 冗余的类型 MOutOfN

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00016 |
| **Type** | Valid |
| **Description** | 如果配置了PersistencyRedundancyMOutOfN，在存入数据时需要拷贝N分冗余数据，读出时应该至少有M份相同的。  （存入数据：在存入Key-Value Storage ， Key-Value Storage里的key, File Storage, 或者File Storage里的一个文件，依赖PersistencyDeployment. redundancyHandling.scope） |
| **Upstream ID** | SWS\_PER\_00319[2111M]、RS\_PER\_00008、RS\_PER\_00009、RS\_PER\_00010 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 上位机配置MOutOfN，如图PersistencyRedundancyMOutOfN    键值库所在的文件夹下会生成N-1个数据库文件的副本。  文件库文件夹下会生成N-1个文件的副本。  CRC和M/N单独使用，不可同时配置。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 不变 |

### [SWRD\_Per\_00017] 冗余的类型 hash

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00017 |
| **Type** | Invalid |
| **Description** | 如果配置了PersistencyRedundancyHash，在存入数据时需要计算hash值，读出时使用hash值来检查数据。  （存入数据：在存入Key-Value Storage ， Key-Value Storage里的key, File Storage, 或者File Storage里的一个文件，依赖PersistencyDeployment. redundancyHandling.scope）  计算hash值可以使用PersistencyRedundancyHash.algorithmFamily定义的算法，PersistencyRedundancyHash.length.定义的位宽。如果配置了PersistencyRedundancyHash.initializationVectorLength，需要计算出包含随机数据的这个长度的初始向量，并传递给哈希算法。  计算hash值和随机数可以使用Crypto API。 |
| **Upstream ID** | SWS\_PER\_00447[2111A]、SWS\_PER\_00448[2111A]、RS\_PER\_00008、RS\_PER\_00009、RS\_PER\_00010、User\_Defined\_PER\_00029 |
| **Dependencies** |  |
| **Verification method** | 测试 |
| **Verification Criteria** |  |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 新增**P3** |

### [SWRD\_Per\_00018] 冗余可以检测数据库是否损坏

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00018 |
| **Type** | Valid |
| **Description** | Per可以使用冗余信息去检测持久内存数据的损坏。  用CRC校验或者通过冗余文件比对数据文件是否有M个相同的。 |
| **Upstream ID** | SWS\_PER\_00221、RS\_PER\_00008 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 上位机上CRC和MOutOfN单独配置，不同时出现。  一、键值库：   1. **CRC检测**   创建个副本文件，OpenKeyValueStorage时先check主文件，错了check副本文件（用副本里crc验证副本的value）  如果副本正确，用副本恢复错误的单个键的值。  如果副本也错误，会报错提示用户删掉错误的键来修改数据库，或者用户调用Reset，恢复成初始数据库。   1. **MOutOfN检测：2/3**   没有主文件的概念，  打开数据库加载数据库时，有2个相同的就加载进来，不够2个的键值被丢弃，并通知用户哪个键出现了问题。  用户也可以调Reset数据库，把数据库恢复成初始数据库。  二、文件库： 1. CRC检测： 文件库如果配置了CRC，会再建一个副本文件库（文件夹）和CRC文件库存储对应的CRC值，用户在打开文件库时，如果主文件库里的CRC对不上了，会检测副本文件夹里的数据是否正确。  如果副本文件夹数据正确，用副本来恢复主文件夹里的错误文件。  如果副本文件夹数据不正确，提示用户，返回错误kIntegrityCorrupted。  用户可以考虑用Recover或者Reset来修复文件库。  Recover会删掉错误文件，Reset用初始文件重置错误文件。 2.MOutOfN检测： 以文件夹做副本。  打开文件库时对文件夹里的文件进行检测，如果某个文件一致的不够M个，就返回错误通知用户。（当用户在用OpenFileReadWrite等方法访问这个有错误的文件时需要提示用户这个文件是被损坏的。）  用户可以考虑用Recover或者Reset来修复文件库。  Recover会删掉错误文件，Reset用初始文件重置错误文件。 |
| **Risk** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **CR** |  |
| **Change Type** | Recover文件库部分新增**P3** |

### [SWRD\_Per\_00019] 冗余检测出错误可以修复数据库

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00019 |
| **Type** | valid |
| **Description** | Per可以使用冗余信息去尽可能修复损坏的数据。  如果使用冗余信息也不能修复损坏的数据，Per返回错误kValidationFailed。  Application可以选择使用ara::per::RecoverKeyValueStorage, ara::per::KeyValueStorage::RecoverKey, ara::per::RecoverAllFiles, 或 ara::per::FileStorage::RecoverFile尽可能修复损坏的数据，或者根据清单调用ara::per::ResetKeyValueStorage, ara::per::KeyValueStorage::ResetKey, ara::per::ResetAllFiles, 或 ara::per::FileStorage::ResetFile重置数据到初始化的状态。  当 ara::per::RecoverKeyValueStorage 被调用时，Persistency 应该将 Key-Value Storage 恢复到一致的状态，包括冗余，丢失其键值对的所有更新值， 或者可能在操作后包含过时的键值对  当 ara::per::RecoverAllFiles 被调用时，Persistency 应该将 File Storage 恢复到一致的状态，包括冗余。根据可用信息，整个文件存储可能会重置为初始状态，丢失其文件的所有更新内容，或者在操作后可能包含过期文件  当 ara::per::KeyValueStorage::RecoverKey 被调用时，Persistency 将尝试将给定的密钥恢复到一致的状态，包括冗余。 根据可用信息，密钥可能会被删除，重置为 [SWS\_PER\_00477] 中所述的初始值，或者在操作后可能包含过时的值。如果该键不存在则返回kInitValueNotAvailable  当 ara::per::FileStorage::RecoverFile 被调用时，Persistency 将尝试将给定文件恢复到一致状态，包括冗余。 根据可用信息，该文件可能会被删除、重置为 [SWS\_PER\_00479] 中所述的初始状态.如果该文件不存在则返回kInitValueNotAvailable |
| **Upstream ID** | SWS\_PER\_00222、RS\_PER\_00009、SWS\_PER\_00452[2111A]、SWS\_PER\_00453[2111A]、SWS\_PER\_00454[2111A]、SWS\_PER\_00455[2111A] 、SWS\_PER\_00456[2111A] 、SWS\_PER\_00477[2111A] 、SWS\_PER\_00478[2111A] 、SWS\_PER\_00479[2111A] |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 键值库的实现是其中有一个报错就返回错误，恢复时是整个库恢复，将键值库的json文件恢复了。  文件库是有错误文件返回错误，恢复是可恢复或重置单独的文件。  恢复的时机参考SWRD\_Per\_00017的说明。  如果Per的数据损坏了，可以用如下API修复数据库。  ara::per::RecoverKeyValueStorage, ara::per::KeyValueStorage::RecoverKey, ara::per::RecoverAllFiles, 或 ara::per::FileStorage::RecoverFile尽可能修复损坏的数据。  还可以根据清单调用ara::per::ResetKeyValueStorage, ara::per::KeyValueStorage::ResetKey, ara::per::ResetAllFiles, 或 ara::per::FileStorage::ResetFile重置数据到初始化的状态。  如果恢复不了，返回错误kValidationFailed。  API使用参照第5章。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | RecoverKey和文件Recover部分新增 **P3** |

## 安装和升级Persistency Data

UCM 处理 Adaptive Application生命周期的以下阶段：

* 安装新软件
* 更新已安装的软件
* 更新成功后的处理
* 更新失败后回滚已更新的软件
* 移除已安装的软件

对于上述所有阶段，都需要在应用程序上处理存储的数据。Adaptive Application可以在安装或更新之后的验证阶段调用updatePersistency显示的触发此处理，或者在访问存储的数据时，使用Per的（OpenKeyValueStorage/OpenFileStorage）隐式的触发此处理。这两种情况下，Per将比较存储的清单的版本和当前的清单版本，并执行所需的操作。

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00020 |
| **Type** | Valid |
| **Description** | 预定义数据的优先级：  在部署数据中定义的元素(PersistencyKeyValueStorage和PersistencyFileStorage以及相关类)总是优先于应用程序设计中定义的元素(PersistencyKeyValueStorageInterface和PersistencyFileStorageInterface以及相关类)。只有当前者不存在时，才应使用后者。 |
| **Upstream ID** | SWS\_PER\_00379、RS\_PER\_00010、RS\_PER\_00012、RS\_PER\_00013 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
| **Verification method** | 测试 |
| **Verification Criteria** | 为了用户体验，简化用户配置流程现隐藏Interface配置。  所以此条不需要验证。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 不变 |

### [SWRD\_Per\_00021] 存储数据的安装

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00021 |
| **Type** | Valid |
| **Description** | 当应用程序使用OpenKeyValueStorage或OpenFileStorage打开KVS或FS时，或者当调用UpdatePersistency时，本模块程序应检查存储数据的存在性。如果没有找到存储的数据，则本模块应初始化存储的数据。 |
| **Upstream ID** | SWS\_PER\_00382、RS\_PER\_00010、RS\_PER\_00012 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification method** | 测试 |
| **Verification Criteria** | 在上位机配置一个KVS库或者FS库，在安装APP时会生成对应的KVS库或FS库。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 不变 |

#### [SWRD\_Per\_00022] 安装Key Value Storage

PersistencyKeyValueStorage相关的安装配置：

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00022 |
| **Type** | Valid |
| **Description** | 1. 在新安装的Adaptive Application清单中找到PortPrototype并创建一个键值库。以PortPrototype的短名称作为入参传给API OpenKeyValueStorage。 2. 在键值库中根据新安装或更新的Adaptive Application清单中的更新策略为keepExisting或overwrite的PersistencyKeyValueStorage.keyValuePair创建键值对。   键值库的键值对由键标识。在PersistencyKeyValueStorage中都可以定义键相同的条目。   1. 键值库中的键值对应使用PersistencyDataElement 或 PersistencyKeyValuePair的短名称作为键。 2. 键值库中的键值对的数据类型使用CppImplementationDataType定义，键值对的类型是PersistencyKeyValuePair.valueDataType引用的CppImplementationDataType。 3. 键值库中的键值对应使用PersistencyKeyValuePair.initValue的值创建，如果没有，则使用PersistencyDataRequiredComSpec.initValue的值创建。 4. 如果存在值或数据类型无效的PersistencyKeyValuePair或PersistencyDataElement，或数据类型存在冲突，则清单无效。   工具应拒绝无效的清单。 |
| **Upstream ID** | SWS\_PER\_00383、SWS\_PER\_00252、SWS\_PER\_00253、SWS\_PER\_00254、SWS\_PER\_00384、SWS\_PER\_CONSTR\_00003、RS\_PER\_00010、RS\_PER\_00012 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 1. 在上位机配置KVS键值库，根据PortPrototype的short name给OpenKeyValueStorage传入的参数。安装后会生成跟清单中的文件名一样的数据库。 2. 在上位机为键值库配置一个键值对，升级策略是keepExisting或overwrite的情况下。在安装时，把这个键值对生成到键值数据库中。 3. 键值库中的键是根据键值对的short name生成的。 4. 上位机配置键值对的数据类型根据CppImplementationDataType定义的，在键值对初始化数据时选择CppImplementationDataType定义的类型。 5. 在上位机给键值对赋初始值，配置PersistencyKeyValuePair下的initValue。 6. 数据清单中的数据跟实际的数据类型不匹配了，认为清单无效。   可以手动修改数据清单，制造错误的数据来验证。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 不变 |

PersistencyKeyValueStorageInterface相关的安装配置：

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00023 |
| **Type** | Valid |
| **Description** | 1. 在新安装的Adaptive Application清单中找到PersistencyKeyValueStorageInterface输入的PortPrototype并创建一个键值库。以PortPrototype的短名称作为入参传给API OpenKeyValueStorage。 2. 在键值库中根据新安装或更新的Adaptive Application清单中的更新策略为keepExisting或overwrite的PersistencyKeyValueStorageInterface.dataElement创建键值对。   键值库的键值对由键标识。在PersistencyKeyValueStorageInterface中都可以定义键相同的条目。   1. 键值库中的键值对应使用PersistencyDataElement 的短名称作为键。 2. 键值库中的键值对的数据类型使用CppImplementationDataType定义，键值对的类型是PersistencyKeyValuePair.valueDataType引用的CppImplementationDataType。 3. 键值库中的键值对应使用PersistencyKeyValuePair.initValue的值创建，如果没有，则使用PersistencyDataRequiredComSpec.initValue的值创建。 4. 如果存在值或数据类型无效的PersistencyDataElement，或数据类型存在冲突，则清单无效。   工具应拒绝无效的清单。 |
| **Upstream ID** | SWS\_PER\_00383、SWS\_PER\_00252、SWS\_PER\_00253、SWS\_PER\_00254、SWS\_PER\_00384、SWS\_PER\_CONSTR\_00003、RS\_PER\_00010、RS\_PER\_00012 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
|  |  |
| **Verification Criteria** | PersistencyKeyValueStorageInterface侧的配置暂不支持。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

#### [SWRD\_Per\_00024] 安装File Storage

PersistencyFileStorage相关的安装配置：

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00024 |
| **Type** | Valid |
| **Description** | 1. Per需要给Adaptive Application的清单中包含每个PortPrototype创建一个文件存储库。打开文件存储库的API OpenFileStorage用PortPrototype的short name作为入参。 2. Per需要在文件存储库中根据Adaptive Application安装或更新的清单中为更新策略类型为keepExisting或overwrite的PersistencyFileStorage.file创建一个文件。   FS中的文件由它们的名称标识。可以在PersistencyFileStorage中定义具有相同名称的文件。   1. 文件库中的文件应使用PersistencyFile.fileName作为文件名。 2. 文件库中的文件应使用PersistencyFile.contentUri所寻址的资源(在已安装的软件包内)中的内容创建，如果不存在，则由PersistencyFileElement.contentUri创建。如果均不存在，则创建空文件。 3. 如果具有相同文件名的PersistencyFileElement和PersistencyFile的短名称不同，则清单无效。   工具应拒绝无效的清单。 |
| **Upstream ID** | SWS\_PER\_00385、SWS\_PER\_00265、SWS\_PER\_00266、SWS\_PER\_00267、SWS\_PER\_CONSTR\_00004、RS\_PER\_00010、RS\_PER\_00012 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | 1. 在上位机配置的PersistencyFileStorageInterface的PortPrototype的短名作为OpenFileStorage的入参来打开文件存储库，获取句柄。 2. 在上位机上配置文件库中新增一个文件的Element，升级策略是keepExisting或overwrite，安装或升级后，新增一个文件。 3. 上位机配置的PersistencyFileElement.fileName或PersistencyFile.fileName作为文件名。 4. 从上位机上传文件在PersistencyFile.contentUri配置对应的名字，在App安装后会根据contentUri到安装的软件包中去取对应的文件存入文件库中。不会存在没有对应的文件的情况，上位机校验会报错。 5. 如果名称不匹配，清单无效 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 不变 |

PersistencyFileStorageInterface相关的安装配置：

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00025 |
| **Type** | Valid |
| **Description** | 1. Per需要给Adaptive Application的清单中包含每个PersistencyFileStorageInterface的PortPrototype创建一个文件存储库。打开文件存储库的API OpenFileStorage用PortPrototype的short name作为入参。 2. Per需要在文件存储库中根据Adaptive Application安装或更新的清单中为更新策略类型为keepExisting或overwrite的PersistencyFileStorageInterface.fileElement创建一个文件。   FS中的文件由它们的名称标识。可以在PersistencyFileStorageInterface中定义具有相同名称的文件。   1. 文件库中的文件应使用PersistencyFileElement.fileName作为文件名。 2. 文件库中的文件应使用PersistencyFile.contentUri所寻址的资源(在已安装的软件包内)中的内容创建，如果不存在，则由PersistencyFileElement.contentUri创建。如果均不存在，则创建空文件。 3. 如果具有相同文件名的PersistencyFileElement和PersistencyFile的短名称不同，则清单无效。   工具应拒绝无效的清单。 |
| **Upstream ID** | SWS\_PER\_00385、SWS\_PER\_00265、User\_Defined\_PER\_00044、SWS\_PER\_00266、SWS\_PER\_00267、SWS\_PER\_CONSTR\_00004、RS\_PER\_00010、RS\_PER\_00012 |
| **Dependencies** | 无 |
| **Verification method** | 测试 |
| **Verification Criteria** | PersistencyFileStorageInterface侧的配置暂不支持。 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
| **Risk** | 无 |
| **CR** |  |
| **Change Type** | 不变 |

### 存储数据的升级

#### [SWRD\_Per\_00026] 版本号

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00026 |
| **Type** | valid |
| **Description** | 版本号相关需求：  Executable.version：用于检测应用程序的更改  SoftwareCluster.version：用于检测存储的数据的更改  当Executable.version增加了，SoftwareCluster.version也需要增加版本  这两个版本号是StrongRevisionLabelStrings，由一个主版本(MajorVersion)，一个小版本(MinorVersion)，一个补丁版本(PatchVersion)和编译版本(BuildVersion)组成以及预发布版本的附加标签和构建元数据组成。我们假设，当版本更改时，前三个值将增加，最后一个值可能是任意的  当增加SoftwareCluster.version或Executable.version时，必须增加主版本或小版本或补丁版本。 |
| **Upstream ID** | SWS\_PER\_00378、RS\_PER\_00010、RS\_PER\_00013、RS\_PER\_00014 |
| **Dependencies** |  |
| **Verification Criteria** | 想要升级，UCM下的SoftwareCluster的version需要增加（前三位增加）：  想要升级Per数据exe的版本号需要增加（前三位有效）： |
| **Risk** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **CR** |  |
| **Change Type** | 不变 |

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00027 |
| **Type** | Valid |
| **Description** | 当应用程序使用OpenKeyValueStorage或OpenFileStorage打开KVS或FS时，或者当调用UpdatePersistency时，Per程序需要比较清单及存储的SoftwareCluster.version。如果清单中的版本高于存储的版本，则Per程序应首先创建存储数据的备份，然后更新数据。  任何时候只需要保存一组备份数据。当执行新的更新时，旧的备份数据可以被覆盖。 |
| **Upstream ID** | SWS\_PER\_00386、RS\_PER\_00010、RS\_PER\_00013 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 比较SoftwareCluster.version的工作由UCM做的。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

#### [SWRD\_Per\_00028] 升级策略

PersistencyDeployment相关的升级配置：

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00028 |
| **Type** | Valid |
| **Description** | 1. 升级策略：  * 在更新Adapt Application或清单之后，持久性集群将创建存储数据的备份，然后使用以下策略之一更新现有的存储的数据： * 现有的存储的数据保持不变(keepExisting)。 * 替换现有的存储的数据(overwrite)。 * 删除现有的存储的数据(delete)。 * 如果当前不存在的情况下，(keepExisting and overwrite)是添加新的存储数据。   更新策略可以在应用程序设计或部署期间设置，可以是针对整个KVS或FS 的（PersistencyCollectionLevelUpdateStrategyEnum选项为keepExisting或delete）或是单个键或文件的（PersistencyElementLevelUpdateStrategyEnum选项为keepExisting、overwrite、或delete）。   1. 升级策略的优先级：   为单个键或文件定义的更新策略（PersistencyDeploymentElement.updateStrategy）总是优于整个的KVS或FS定义的更新策略（PersistencyDeployment.updateStrategy）。只有当前者不存在时，才应使用后者。 |
| **Upstream ID** | SWS\_PER\_00380、RS\_PER\_00010、RS\_PER\_00012、RS\_PER\_00013 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 1. 安装一个带有预存储数据的App。   在上位机配置各种升级方法的预存储数据进行升级，看升级后数据是否跟清单一致。     1. 在上位机配置Element设置升级策略，库也设置升级策略。数据根据Element的升级策略进行升级。   如图PersistencyKeyValueDatabase\_1库的升级策略是delete，PersistencyKeyValuePair键值对的升级策略是overwrite，升级后PersistencyKeyValuePair键值对是根据overwrite的策略升级的。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

PersistencyInterface相关的升级配置：

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00029 |
| **Type** | Valid |
| **Description** | 1. 升级策略的优先级：   为单个键或文件定义的更新策略（PersistencyInterfaceElement.updateStrategy）总是优于整个的KVS或FS定义的更新策略（PersistencyInterface.updateStrategy）。只有当前者不存在时，才应使用后者。 |
| **Upstream ID** | SWS\_PER\_00380 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 为了用户体验，简化用户配置流程现隐藏Interface配置。  所以此条不需要验证。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

PersistencyDeployment和PersistencyInterface的升级优先级说明：

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00030 |
| **Type** | Valid |
| **Description** | 升级策略的优先级：  部署数据中定义的更新策略（PersistencyDeployment. updateStrategy，PersistencyDeploymentElement. updateStrategy）总是优先于应用程序设计中定义的更新策略（PersistencyInterface. updateStrategy，PersistencyInterfaceElement.updateStrategy），只有当前者不存在时，才应使用后者。 |
| **Upstream ID** | SWS\_PER\_00251、RS\_PER\_00010、RS\_PER\_00012、RS\_PER\_00013 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
| **Verification Criteria** | 为了用户体验，简化用户配置流程现隐藏Interface配置。  所以此条不需要验证。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

#### [SWRD\_Per\_00031] 升级Key-value Storage

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00031 |
| **Type** | Valid |
| **Description** | 1. 当在更新的清单中检测到PersistencyKeyValueStorageInterface键入的新PortPrototype时，Per将创建[SWS\_PER\_00383]中指定的KVS。 2. 当在更新的清单中缺少PersistencyKeyValueStorageInterface输入的PortPrototype时，Per应删除相应的KVS。 3. 当在更新清单中检测到带有新键的PersistencyKeyValueStorageInterface.dataElement 或 PersistencyKeyValueStorage.keyValuePair时，Per将在KVS中根据[SWS\_PER\_00252],[SWS\_PER\_00253],[SWS\_PER\_00254],[SWS\_PER\_00384]创建一个新条目。 4. 当一个KVS的现有的键不能与更新清单中任何的PersistencyKeyValueStorageInterface.dataElement 或者PersistencyKeyValueStorage.keyValuePair相关联，且对应于KVS的PersistencyKeyValueStorage或PersistencyKeyValueStorageInterface的更新策略为delete时，per应从KVS中删除该键所对应的条目。 5. 当KVS的键在更新清单的PersistencyKeyValueStorageInterface.dataElement 或 PersistencyKeyValueStorage.keyValuePair存在，并且更新策略为overwrite时，Per应根据[SWS\_PER\_00254][SWS\_PER\_00384]替换当前KVS的此键的类型和值为指定的新类型和值。   在PersistencyKeyValueStorageInterface和 PersistencyKeyValueStorage,中都可以定义键相同的条目，在这种情况下[SWS\_PER\_00379]适用。根据[SWS\_PER\_00251]和[SWS\_PER\_00380]确定更新策略。   1. 当KVS的现有的键在更新清单中PersistencyKeyValueStorageInterface.dataElement 或 PersistencyKeyValueStorage.keyValuePair存在，并且更新策略为delete时，per应从KVS中删除该键所对应的条目。   在更新时，使用更新策略keepExisting更新的键在更新期间不会被修改。Per不检查值，也不检查类型。 |
| **Upstream ID** | SWS\_PER\_00388、SWS\_PER\_00389、SWS\_PER\_00390、SWS\_PER\_00391、SWS\_PER\_00275、SWS\_PER\_00277、RS\_PER\_00010、RS\_PER\_00013 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 由于PersistencyKeyValueStorageInterface隐藏，所以只关注AdaptiveApplicationSwComponentType的PortPrototype。   1. 升级工程的上位机通过Port Prototype To Key Value Database Mapping关联的PortPrototypeIref的port口和键值库配置新的键值存储，升级后新增对应的KVS。 2. 升级工程的上位机删除安装工程配置的键值库，升级后相应的KVS也会被删除。 3. 在升级工程的上位机为键值库配置一个键值对，升级策略是keepExisting或overwrite的情况下。在升级时，把这个键值对生成到键值数据库中。 4. 在升级工程的上位机中为键值库配置一个键值对，升级策略是delete。在升级时，把这个键值对从键值数据库中删除。 5. 升级工程的上位机配置升级策略overwrite时，升级时，Per用升级包中的数据替换KVS库中的数据。 6. 升级工程的上位机配置升级策略delete时，升级时，Per把 KVS库中的对应键值对删掉。   更新策略是keepExisting时，相关键值对不变。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

#### [SWRD\_Per\_00032] 升级File Storage

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00032 |
| **Type** | Valid |
| **Description** | 1. 当在更新的清单中检测到PersistencyFileStorageInterface键入的新PortPrototype时，Per应创建[SWS\_PER\_00385]中指定的FS。 2. 当更新的清单中缺少PersistencyFileStorageInterface输入的PortPrototype时，per应删除相应的FS。 3. 当在更新的清单中检测到带有新文件名的PersistencyFileStorageInterface.fileElement 或PersistencyFileStorage.file时，per应在FS根据[SWS\_PER\_00265][SWS\_PER\_00266][SWS\_PER\_00267]指定的那样创建一个新文件。 4. 当FS的现有文件不能与更新清单中的任何PersistencyFileStorageInterface.fileElement或 PersistencyFileStorage.file对应，且与FS对应的PersistencyFileStorage 或 PersistencyFileStorageInterface的更新策略为delete时，Per应从FS里删除该文件。   根据[SWS\_PER\_00251]确定更新策略。   1. 当FS的现有文件可以与更新清单中的PersistencyFileStorageInterface.fileElement 或 PersistencyFileStorage.file对应时，并且更新策略为overwrite时，per应根据[SWS\_PER\_00267]中指定的新内容替换FS中的文件内容。   可以在PersistencyFileStorageInterface和 PersistencyFileStorage中定义具有相同名称的文件，在这种情况下应用[SWS\_PER\_00379]。根据[SWS\_PER\_00251]和[SWS\_PER\_00381]确定更新策略。   1. 当FS中现有的文件与更新清单的PersistencyFileStorageInterface.fileElement或 PersistencyFileStorage.file对应上，并且更新策略为delete，per应该从FS中删除该文件。   在更新期间，使用keepExisting更新的文件将不会被修改。Per不会检查现有的文件内容。 |
| **Upstream ID** | SWS\_PER\_00392、SWS\_PER\_00393、SWS\_PER\_00394、SWS\_PER\_00395、SWS\_PER\_00281、SWS\_PER\_00283、RS\_PER\_00010、RS\_PER\_00013 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 由于PersistencyFileStorageInterface隐藏，所以只关注AdaptiveApplicationSwComponentType的PortPrototype。   1. 升级工程的上位机通过PersistencyPortPrototypeToFileArrayMapping关联的PortPrototypeIref的port口和文件存储库配置新的文件存储，升级后新增对应的FS。 2. 升级工程的上位机删除安装工程配置的文件存储库，升级后相应的FS也会被删除。 3. 在升级工程的上位机为键值库配置一个文件，升级策略是keepExisting或overwrite的情况下。在升级时，把这个文件存储到文件库中。 4. 在升级工程的上位机中为文件库配置一个文件的升级策略是delete。在升级时，把这个键值对从文件库中删除。 5. 升级工程的上位机配置升级策略overwrite时，升级时，Per用升级包中的文件替换FS库中的数据。 6. 升级工程的上位机配置升级策略delete时，升级时，Per把 FS库中的对应文件删掉。   更新策略是keepExisting时，相关文件不变。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

#### [SWRD\_Per\_00033] 用户自定义升级

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00033 |
| **Type** | Valid |
| **Description** | 当应用程序使用OpenKeyValueStorage或OpenFileStorage打开KVS或FS时，或当调用UpdatePersistency时，Per程序应比较清单和存储的Executable.version。如果清单中的版本高于存储的版本，则应调用应用程序使用ara::per::RegisterApplicationDataUpdateCallback注册的函数去更新每个KVS和FS。  应用程序使用ara::per::RegisterApplicationDataUpdateCallback注册的函数可以用于手动更新Key-Value存储的Key-Value对或文件存储的文件，键值存储或文件存储由提供给该函数的ara::core::InstanceSpecifier标识。应用程序可能基于可执行文件。作为函数的第二个参数提供的存储数据的版本，以旧格式或旧类型读取存储的数据，转换数据，并使用当前版本所期望的新格式或新类型再次存储它。 |
| **Upstream ID** | SWS\_PER\_00387、RS\_PER\_00010、RS\_PER\_00013 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | //用户回调函数的实现，在用户手码部分实现  void userRegisterApplication(string name, string version)  {  cout<<"回调函数："<<name<<", "<<version<<endl;  }  int main()  {  //这部分是在生成代码的main函数的UpdatePersistency的手码区实现  // userRegisterApplication:函数指针  RegisterApplicationDataUpdateCallback(userRegisterApplication);  return 0;  } |
|  | 无 |
| **Change Type** | 不变 |

### [SWRD\_Per\_00034] 存储数据升级成功后

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00034 |
| **Type** | Valid |
| **Description** | 更新成功后，UCM将完成新的Adaptive Application.。Per不需要做任何事件，但是它可以释放上次备份分配的资源。  在安装和更新之后，持久化通常会在应用程序的验证阶段使用ara::per:: updatepersistency调用。当这成功时，应用程序将由UCM完成，然后在正常的执行模式中再次启动。在这种情况下，持久性应该删除在前一次更新期间创建的任何备份。  当应用程序使用OpenKeyValueStorage或OpenFileStorage打开KVS或FS时，或者当调用UpdatePersistency时，持久性应该比较清单中SoftwareCluster.version与存储的版本。如果两个版本相同，持久化将删除所有备份数据。 |
| **Upstream ID** | SWS\_PER\_00446、RS\_PER\_00013 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 升级成功时，数据升级成功。备份的资源目前没有释放，是每次需要新做备份动作时清掉旧的备份。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

### [SWRD\_Per\_00035] 升级失败后回退存储数据

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00035 |
| **Type** | Valid |
| **Description** | 当更新失败时，ucm将恢复到旧的Adaptive Application.和清单。然后，Per将用更新期间创建的备份替换当前使用的存储的数据。  当App 调用ara::per::OpenKeyValueStorage或ara::per::OpenFileStorage, 和ara::per::UpdatePersistency打开Key-Value Storage或 File Storage， Per需要比较manifest中的SoftwareCluster.version和存储的version，如果版本号低于存储的版本号，Per将会比较manifest和备份中存的版本号，如果版本号匹配，Per将会把Per的数据重置成备份的版本。否则，移除Key-Value Storages 和File Storages,重新安装Per数据从manifest。 |
| **Upstream ID** | User\_Defined\_PER\_00039、SWS\_PER\_00396、RS\_PER\_00014 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 如果升级失败，UCM触发版本回退，用升级前的旧版本触发升级动作，例如升级前版本号是1.0，升级的版本号是2.0，当前运行环境的版本号已经通过升级动作变成2.0了，这时升级失败回退了。UCM会切换到1.0版本来再做升级。Per比较1.0和当前环境的2.0版本，比当前版本小，再用1.0和Per备份的版本号进行比较。跟备份的版本号一致，把备份的Per数据拷贝回来到运行路径，最后Per的运行路径下是1.0版本的清单和数据。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

## 资源管理

Persistency cluster支持对单个KVS库和FS库设置库可使用的最大size和最小size。

PersistencyDeployment.minimumSustainedSize 和PersistencyDeployment.maximumAllowedSize。

可以使用ara::per::GetCurrentKeyValueStorageSize 或ara::per::GetCurrentFileStorageSize和ara::per::FileStorage::GetCurrentFileSize来获取当前占用的空间。

### [SWRD\_Per\_00037] KVS库设置最小值

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00037 |
| **Type** | Valid |
| **Description** | Per功能集群需要确保被配置的空间PersistencyDeployment.minimumSustainedSize对Key ValueStorage是有效的。  在部署期间首先分配最小size，并且在删除持久数据时永远不要将大小减少到这个值以下。 |
| **Upstream ID** | SWS\_PER\_00320、RS\_PER\_00010、RS\_PER\_00011 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
| **Verification Criteria** | 在上位机为单个键值库设置最小值，安装APP  调用GetCurrentKeyValueStorageSize(ara::core::StringView kvs)查看键值库的size是否是大于最小size。 |
| **Risk** |  |
| **Change Type** | 新增**P3** |

### [SWRD\_Per\_00038] KVS库设置最大值

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00038 |
| **Type** | Valid |
| **Description** | Per应该确保给Key-value存储分配的实际空间从没超过PersistencyDeployment.maximumAllowedSize配置的。  需要监控所有对Per存储做的写访问。 |
| **Upstream ID** | SWS\_PER\_00321[2011A]、User\_Defined\_PER\_00062、RS\_PER\_00010、RS\_PER\_00011 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 在上位机为单个键值库设置最大值，安装APP  调用GetCurrentKeyValueStorageSize(ara::core::StringView kvs)查看键值库的当前使用的size。  再向键值库写入数据，直到写入的数据达到最大size，看是否会有空间不足的错误提示。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

### [SWRD\_Per\_00039] FS库设置最小值

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00039 |
| **Type** | Valid |
| **Description** | Per功能集群需要确保被配置的空间PersistencyDeployment.minimumSustainedSize对File storage是有效的。  在部署期间首先分配最小size，并且在删除持久数据时永远不要将大小减少到这个值以下。 |
| **Upstream ID** | SWS\_PER\_00320、User\_Defined\_PER\_00061、RS\_PER\_00010、RS\_PER\_00011 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
| **Verification Criteria** | 在上位机为单个文件库设置最小值，安装APP  调用GetCurrentFileStorageSize(ara::core::StringView fs)查看文件库的size是否是大于最小size。 |
| **Risk** |  |
| **Change Type** | 新增**P3** |

### [SWRD\_Per\_00040] FS库设置最大值

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00040 |
| **Type** | Valid |
| **Description** | Per应该确保给文件存储分配的实际空间从没超过PersistencyDeployment.maximumAllowedSize配置的。  需要监控所有对Per存储做的写访问。 |
| **Upstream ID** | SWS\_PER\_00321[2111A]、User\_Defined\_PER\_00062、RS\_PER\_00010、RS\_PER\_00011 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 在上位机为单个文件库设置最大值，安装APP  调用GetCurrentFileStorageSize(ara::core::StringView fs)查看文件库当前使用的size。  再向文件库写入数据，直到写入的数据达到最大size，看是否会有空间不足的错误提示。 |
| **Risk** |  |
| **Change Type** | 不变 |

## Key Value Storages支持的数据类型

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00041 |
| **Type** | Valid |
| **Description** | KVS支持的数据类型  Per应能够将AUTOSAR\_SWS\_AdaptivePlatformTypes中描述的所有数据类型存储在KVS中。 |
| **Upstream ID** | SWS\_PER\_00302、RS\_PER\_00001 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 在上位机配置下面类型存储到KVS中  Map  数组， Vector  结构体  枚举，String  int8\_t,int16\_t, int32\_t, int64\_t  uint8\_t, uint16\_t, uint32\_t, uint64\_t  单精度：float  双精度：double  bool |
| **Risk** | 无 |
| **Change Type** | 不变 |

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00042 |
| **Type** | Valid |
| **Description** | KVS支持的数据类型  Per应该能够在KVS中存储序列化的二进制数据。  序列化的二进制数据必须表示为ara::core::Vector<ara::core::Byte>  用于应用程序存储自定义数据类型 |
| **Upstream ID** | SWS\_PER\_00303、RS\_PER\_00001 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | H |
| **Verification Criteria** | 写入读出正确。 |
| **Risk** | 无 |
| **Change Type** | 不变 |

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00043 |
| **Type** | Vaild |
| **Description** | KVS支持的数据类型  Per应能够将PersistencyKeyValueStorageInterface.dataTypeForSerialization或PersistencyKeyValueStorageInterface.dataElement在PersistencyKeyValueStorage应用设计中引用的所有CppImplementationDataTypes存储在相应的KVS中。  用于应用程序存储自定义数据类型。 |
| **Upstream ID** | SWS\_PER\_00304、RS\_PER\_00001、RS\_PER\_00010 |
| **Dependencies** | 无 |
| **ASIL** | 无 |
| **Status** | Draft |
| **Priority** | L |
| **Verification Criteria** | 由于PersistencyKeyValueStorageInterface和PersistencyKeyValueStorageInterface被隐藏. |
| **Risk** | 无 |
| **Change Type** | 不变 |

## 访问文件的附加信息

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00044 |
| **Type** | Valid |
| **Description** | 持久化集群提供了ara::per::FileStorage::GetFileInfo方法来获取存储文件的信息。  该方法返回有关：  文件创建时间(creationTime)  最后修改时间(modiationtime)  最后访问时间(accessTime)的信息  文件创建的方式和对象(fileCreationState)  最后修改的信息(fileModificationState)。 |
| **Upstream ID** |  |
| **Dependencies** |  |
| **ASIL** |  |
| **Status** |  |
| **Priority** |  |
| **Verification Criteria** |  |
| **Risk** |  |
| **Change Type** | P3 |

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00045 |
| **Type** | Valid |
| **Description** | 方法ara::per::FileStorage::GetFileInfo将收集所需的信息到一个ara::per::FileInfo结构中，并将其返回给应用程序。  如果持久化集群使用底层操作系统的文件系统，则可以从文件系统获得部分信息(如创建或访问时间)。只有当文件当前没有打开时，该信息才准确。 |
| **Upstream ID** | SWS\_PER\_00440、RS\_PER\_00004 |
| **Dependencies** |  |
| **ASIL** |  |
| **Status** |  |
| **Priority** |  |
| **Verification Criteria** |  |
| **Risk** |  |
| **Change Type** | P3 |

## 回调函数

|  |  |
| --- | --- |
| **SWRD\_ID** | SWRD\_Per\_00046 |
| **Type** | Valid |
| **Description** | 当 ara::per::RegisterRecoveryReportCallback 被调用时，Persistency 将注册提供的函数并启用报告该应用程序所有存储中的冗余问题。  通过注册的回调，在可以恢复键值存储、文件存储、键值对或文件时声明 kKeyValueStorageRecovered、kKeyRecovered、kFileStorageRecovered 或 kFileRecovered，如果没有，则声明 kKeyValueStorageRecoveryFailed、kKeyRecoveryFailed、kFileStorageRecoveryFailed 或 kFileRecoveryFailed |
| **Upstream ID** | SWS\_PER\_00480(2111A)、SWS\_PER\_00481(2111A) 、SWS\_PER\_00482(2111A) 、SWS\_PER\_00483(2111A) 、SWS\_PER\_00484(2111A) 、SWS\_PER\_00485(2111A) 、SWS\_PER\_00486(2111A) 、SWS\_PER\_00487(2111A) 、SWS\_PER\_00488(2111A) 、SWS\_PER\_00489(2111A) 、SWS\_PER\_00490(2111A)、RS\_PER\_00008 |
| **Dependencies** |  |
| **ASIL** | 无 |
| **Status** |  |
| **Priority** | L |
| **Verification Criteria** |  |
| **Risk** | 无 |
| **Change Type** | P3 |

# 非功能需求(Non-Functional Requirements)

## 制约（Constraint）

[

1. 软件运行环境：考虑以下几个方面：

接口：与其他模块的接口；

环境：使用到其他模块数据类型，AutoSAR头文件的包含关系；

1. 法律、法规和标准：如：必须按照ISO26262标准进行开发。]

### [SWRD\_ID]非功能需求1(Non-Function Requirement No.1)

|  |  |
| --- | --- |
| **SWRD\_ID** |  |
| **Type** |  |
| **Description** |  |
| **Upstream ID** |  |
| **Dependencies** |  |
| **ASIL** |  |
| **Status** |  |
| **Priority** |  |
| **Verification Criteria** |  |
| **Risk** |  |
| **Change Type** |  |

### [SWRD\_ID]非功能需求2(Non-Function Requirement No.2)

|  |  |
| --- | --- |
| **SWRD\_ID** |  |
| **Type** |  |
| **Description** |  |
| **Upstream ID** |  |
| **Dependencies** |  |
| **ASIL** |  |
| **Status** |  |
| **Priority** |  |
| **Verification Criteria** |  |
| **Risk** |  |
| **Change Type** |  |

## 性能（Performance）

[从可用性，效率，可维护性，可移植性，可扩展性和性能等角度进行分析。

1. 可用性：使用，操作相关需求
2. 效率性：技术软件的反应速度和内存使用量，如：对Rom和Ram的使用量的约束、对内存大小、硬盘大小的约束，对CPU的约束等。

如：CPU符合率不满50%等，NVM相关使用参考“NVMList”

1. 可维护性：变更解析相关、操作方式变化、运行环境的变化、接口变化、精度、时间性能等的需求发生变化时，该软件对这些变化的适应能力要求
2. 可移植性：向其他制品的展开相关要求
3. 可扩展性：派生开发要求
4. 性能：软件相关的性能：如：数据精度要求，说明该软件的输入输出的的数据精度要求，可能包含传输过程中的精度要求；

时间性能要求，包括：响应时间、更新处理时间，数据转换和传送时间要求等]

### [SWRD\_ID]非功能需求3(Non-Function Requirement No.3)

|  |  |
| --- | --- |
| **SWRD\_ID** |  |
| **Type** |  |
| **Description** |  |
| **Upstream ID** |  |
| **Dependencies** |  |
| **ASIL** |  |
| **Status** |  |
| **Priority** |  |
| **Verification Criteria** |  |
| **Risk** |  |
| **Change Type** |  |

### [SWRD\_ID]非功能需求4(Non-Function Requirement No.4)

|  |  |
| --- | --- |
| **SWRD\_ID** |  |
| **Type** |  |
| **Description** |  |
| **Upstream ID** |  |
| **Dependencies** |  |
| **ASIL** |  |
| **Status** |  |
| **Priority** |  |
| **Verification Criteria** |  |
| **Risk** |  |
| **Change Type** |  |

## 质量（Quality）

[软件相关的质量需求：]

### [SWRD\_ID]非功能需求5(Non-Function Requirement No.5)

|  |  |
| --- | --- |
| **SWRD\_ID** |  |
| **Type** |  |
| **Description** |  |
| **Upstream ID** |  |
| **Dependencies** |  |
| **ASIL** |  |
| **Status** |  |
| **Priority** |  |
| **Verification Criteria** |  |
| **Risk** |  |
| **Change Type** |  |

### [SWRD\_ID]非功能需求6(Non-Function Requirement No.6)

|  |  |
| --- | --- |
| **SWRD\_ID** |  |
| **Type** |  |
| **Description** |  |
| **Upstream ID** |  |
| **Dependencies** |  |
| **ASIL** |  |
| **Status** |  |
| **Priority** |  |
| **Verification Criteria** |  |
| **Risk** |  |
| **Change Type** |  |

# 接口说明（API）

[记录AutoSar中接口相关信息，包括：API，服务接口，错误码等。该章节内容也可以单独使用“软件接口列表”来定义，在此处引用该文件即可。]

## 接口头文件（API Header files）

[记录AutoSar中接口头文件信息]

## 接口共同数据类型（API Common Data Types）

[记录AutoSar中共同数据类型信息]

## Key Value Storage 接口定义

### [SWRD\_API\_Per\_00001] OpenKeyValueStorage

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00001 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00052] | |
| **CR** |  | |
| **Consistency** | No | |
| **Change Type** | 修改 返错和入参 | |
| **Kind:** | function | |
| **Symbol:** | OpenKeyValueStorage(const ara::core::InstanceSpecifier &kvs) | |
| **Scope:** | namespace ara::per | |
| **Syntax:** | ara::core::Result<SharedHandle<KeyValueStorage> > OpenKeyValueStorage  (const ara::core::InstanceSpecifier &kvs) noexcept; | |
| **Parameters (in):** | kvs | 由PersistencyKeyValueStorageInterface输入PortPrototype的短名称路径。 |
| **Return value:** | ara::core::Result< SharedHandle< Key ValueStorage > > | 包含SharedHandle的结果，包括下面定义的错误，或为供应商定义的错误之一。 |
| **Errors** | PerErrc::kStorageNotFound | 如果传递的InstanceSpecifier与为该可执行文件配置的任何PersistencyKeyValueStorageInterface不匹配，则返回。 |
|  | PerErrc::kPhysicalStorageFailure | 如果访问存储失败，返回。 |
|  | PerErrc::kIntegrityCorrupted（冗余异常，初始化数据时失败） | 如果由于结构完整性损坏而无法读取存储的数据，则返回。 |
|  | PerErrc::kValidationFailed（冗余异常） | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc::kEncryptionFailed（加密异常） | 如果解密存储的数据失败则返回。 |
|  | PerErrc::kResourceBusy | 如果当前正在执行updatepersistence或Reset persistency，或者对于相同的键值存储正在执行RecoverKeyValue Storage或ResetKeyValueStorage，则返回。 |
|  | PerErrc::kOutOfStorageSpace（资源管理返回） | 当可用存储空间不足以存储新增/更新的值时返回。 |
|  | PerErrc::kNotInitialized（初始化返回） | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | |
| **Description:** | 打开键值数据库时，当键值数据库正在被修改或者被别的线程（UpdatePersistency, ResetPersistency, Recover KeyValueStorage, or ResetKeyValueStorage）调用时，OpenKeyValueStorage将会返回错误kResourceBusy。  因为多个线程同时访问键值数据库时，数据库可能不能被关闭了。SharedHandle已经超出的范围，只有所有的SharedHandles都超出范围才能关闭了。 | |
| **Additional:** | 打开键值数据库，返回键值存储类的实例。  做成非标接口OpenKeyValueStorage(ara::core::StringView kvs)  代码示例及类句柄的获取说明：  auto result = OpenKeyValueStorage(StringView("dbname"));  if (!result) {  //打开库失败，错误信息为:result.Error().Message();  //相应的处理  }  auto db = std::move(result).Value();  此db即为类的指针，可以使用此db->类的相应函数 | |

### [SWRD\_API\_Per\_00002] RecoverKeyValueStorage

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00002 | |
| **Type** | Valid | |
| **Priority** | H | |
| **CR** |  | |
| **Upstream ID** | [SWS\_PER\_00333] | |
| **Consistency** | No | |
| **Change Type** | 修改 返错和参数 | |
| **Kind** | Function | |
| **Symbol** | RecoverKeyValueStorage(const ara::core::InstanceSpecifier &kvs) | |
| **Scope** | namespace ara::per | |
| **Syntax** | ara::core::Result RecoverKeyValueStorage (const ara::core::InstanceSpecifier &kvs) noexcept; | |
| **Parameters (in):** | kvs | 由PersistencyKeyValueStorageInterface输入PortPrototype的短名称路径。 |
| **Return value** | ara::core::Result< void > | 返回void，在错误的情况下，它包含任何定义的错误,或供应商的错误。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Errors:** | PerErrc::kStorageNotFound（没做） | 如果传递的InstanceSpecifier与为该可执行文件配置的任何PersistencyKeyValueStorageInterface不匹配，则返回。 |
| PerErrc::kPhysicalStorageFailure | 如果访问存储失败，返回。 |
| PerErrc::kEncryptionFailed | 如果存储的数据加密失败则返回。 |
| PerErrc::kResourceBusy | 如果当前正在执行updatepersistence或resetpersistence，或者ResetKeyValue Storage正在为相同的Key-Value Storage执行，或者相同的Key-Value Storage的SharedHandle正在使用，则返回。 |
| PerErrc::kOutOfStorageSpace | 如果在增加或者更新数据的时候存储空间不足将返回。 |
| PerErrc::kNotInitialized | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | |
| **Description** | 恢复键值存储数据库。  在冗余检查失败时可以恢复键值存储数据库来恢复key-value的存储。当存储数据库被打开或者被多个线程（UpdatePersistency, ResetPersistency, RecoverKey ValueStorage, or ResetKeyValueStorage.）修改时，会报kResourceBusy错误。此方法尽最大努力恢复所有键。恢复后，key可能是过期的，初始的数据或者也可能丢失。 | |
| **Additional:** | 调用::ara::per::RecoverKeyValueStorage("/NeuSAR/appPersistencyDemo/portKVS") //参数是port名 | |

### [SWRD\_API\_Per\_00003] ResetKeyValueStorage

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00003 | |
| **Type** | Valid | |
| **Priority** | H | |
| **CR** |  | |
| **Upstream ID** | [SWS\_PER\_00334] | |
| Consistency | No | |
| **Change Type** | 修改 返错和参数 | |
| **Kind** | Function | |
| **Symbol** | ResetKeyValueStorage(const ara::core::InstanceSpecifier &kvs) | |
| **Scope** | namespace ara::per | |
| **Syntax** | ara::core::Result ResetKeyValueStorage (const ara::core::InstanceSpecifier &kvs) noexcept; | |
| **Parameters (in):** | kvs | 由PersistencyKeyValueStorageInterface输入PortPrototype的短名称路径。 |
| **Return value** | ara::core::Result< void > | 返回void，在错误的情况下，它包含任何定义的错误,或供应商的错误。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Errors:** | PerErrc::kStorageNotFound | 如果传递的InstanceSpecifier与为该可执行文件配置的任何PersistencyKeyValueStorageInterface不匹配，则返回。 |
| PerErrc::kPhysicalStorageFailure | 如果访问存储失败，返回。 |
| PerErrc::kEncryptionFailed | 如果存储的数据加密失败则返回。 |
| PerErrc::kResourceBusy | 如果当前正在执行updatepersistence或resetpersistence，或者ResetKeyValue Storage正在为相同的Key-Value Storage执行，或者相同的Key-Value Storage的SharedHandle正在使用，则返回。 |
| PerErrc::kOutOfStorageSpace | 如果在增加或者更新数据的时候存储空间不足将返回。 |
| PerErrc::kNotInitialized | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | |
| **Description** | 重置键值存储数据库到初始状态。  ResetKeyValueStorage允许重置key-value存储到初始状态，包括在部署时根据manifest设的keys和初始的value值。  当key-value数据库被打开，或者被别的线程（UpdatePersistency, ResetPersistency, RecoverKey ValueStorage, or ResetKeyValueStorage）调用时, 会报kResourceBusy的错误。 | |
| **Additional:** | 调用::ara::per::ResetKeyValueStorage("/NeuSAR/appPersistencyDemo/portKVS")//参数是port名 | |

### [SWRD\_API\_Per\_00004] GetCurrentKeyValueStorageSize

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00004 | |
| **Type** | Valid | |
| **Priority** | H | |
| **CR** |  | |
| **Upstream ID** | [SWS\_PER\_00405]、[SWS\_PER\_00491][2111A] | |
| Consistency | No | |
| **Change Type** | 修改 | |
| **Kind** | Function | |
| **Symbol** | GetCurrentKeyValueStorageSize(const ara::core::InstanceSpecifier &kvs) | |
| **Scope** | namespace ara::per | |
| **Syntax** | ara::core::Result<uint64\_t> GetCurrentKeyValueStorageSize (const ara::core::InstanceSpecifier &kvs) const noexcept; | |
| **Parameters (in):** | kvs | 由PersistencyKeyValueStorageInterface输入PortPrototype的短名称路径 |
| **Return value** | ara::core::Result< uint64\_t > | 返回已占用的空间字节。  在错误的情况下，它包含任何定义的错误,或于供应商的错误。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Errors:** | PerErrc::kStorageNotFound | 如果传递的InstanceSpecifier与为该可执行文件配置的任何PersistencyKeyValueStorageInterface不匹配，则返回。 |
| PerErrc::kPhysicalStorageFailure | 如果访问存储失败，返回。 |
| PerErrc::kNotInitialized | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | |
| **Description** | 返回key-value已占用的空间字节。  返回的size包括所有元数据和冗余和备份所占用的空间。在如果没有其他操作键值存储在同一时间发生，返回的size值是准确的。  GetCurrentKeyValueStorageSize 应返回当前分配给 Key-Value Storage 的存储空间的总大小，包括管理数据、冗余数据和备份数据**(目前我们实现的是返回元数据所占据的空间，后续需要改成全部数据所占的空间，P1)** | |
| **Additional:** | 调用::ara::per::GetCurrentKeyValueStorageSize("/NeuSAR/appPersistencyDemo/portKVS")//参数是port名 | |

### keyvalueStorage 类

这个章节介绍的方法需要用从[SWS\_PER\_00052]获得的ara::per::KeyValueStorage对象调用。

[SWS\_PER\_00331] 修改键值存储的操作只能临时执行，以便后续的操作知道这个变化。实际只有当调用ara::per::KeyValueStorage::SyncToStorage时才更新存储。(RS\_PER\_00003)

因此，如果key-value存储被破坏了（也可能进程终止），key-value数据库不被更新，或者下次访问数据库时，APP会找到最后保持的状态，用ara::per::KeyValueStorage::DiscardPendingChanges恢复数据库到最后保存的状态。

请记住：并行访问KVS的线程需要知道,其他线程所做的更改将立即可见, ara::per::KeyValueStorage::SyncToStorage 和ara::per::KeyValueStorage::DiscardPendingChanges的影响会影响到所有线程。

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00005 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00339] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 修改 修饰符 |
| **Kind** | Function |
| **Symbol** | KeyValueStorage |
| **Scope** | namespace ara::per |
| **Syntax** | class KeyValueStorage final {...}; |
| **Header file:** | #include "ara/per/key\_value\_storage.h" |
| **Description** | 键值存储类，包含一组关联值的键 |
| **Additional:** | 代码示例及类句柄的获取说明：  auto result = OpenKeyValueStorage(StringView("dbname"));  if (!result) {  //打开库失败，错误信息为:result.Error().Message();  //相应的处理  }  auto db = std::move(result).Value();  此db即为类的指针，可以使用此db->类的相应函数。  **KeyValueStorage 改为final类，P1** |

#### [SWRD\_API\_Per\_00006] KeyValueStorage

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00006 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00322] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 不变 | |
| **Kind** | Function | |
| **Symbol** | KeyValueStorage(KeyValueStorage &&kvs) | |
| **Scope** | class ara::per::KeyValueStorage | |
| **Syntax** | KeyValueStorage (KeyValueStorage &&kvs) noexcept; | |
| **Parameters (in):** | kvs | 要移动的KeyValueStorage对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | |
| **Description** | 移动构造函数 | |
| **Additional:** |  | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00007 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00324] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 不变 |
| **Kind** | Function |
| **Symbol** | KeyValueStorage(const KeyValueStorage &) |
| **Scope** | class ara::per::KeyValueStorage |
| **Syntax** | KeyValueStorage (const KeyValueStorage &)=delete; |
| **Header file:** | #include "ara/per/key\_value\_storage.h" |
| **Description** | 拷贝构造函数，不应被使用 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00008] operator=

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00008 | |
| **Upstream ID** | [SWS\_PER\_00323] | |
| **Type** | Valid | |
| **Priority** | H | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 不变 | |
| **Kind** | Function | |
| **Symbol** | operator=(KeyValueStorage &&kvs) | |
| **Scope** | class ara::per::KeyValueStorage | |
| **Syntax** | KeyValueStorage& operator= (KeyValueStorage &&kvs) &noexcept; | |
| **Parameters (in):** | kvs | 要移动的KeyValueStorage对象 |
| **Return value** | KeyValueStorage & | 移动后的KeyValueStorage对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | |
| **Description** | 移动赋值运算符 | |
| **Additional:** |  | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00009 |
| **Type** | Valid |
| **Priority** | H |
| **CR** |  |
| **Upstream ID** | [SWS\_PER\_00325] |
| **Consistency** | Yes |
| **Change Type** | 不变 |
| **Kind** | Function |
| **Symbol** | operator=(const KeyValueStorage &) |
| **Scope** | class ara::per::KeyValueStorage |
| **Syntax** | KeyValueStorage& operator= (const KeyValueStorage &)=delete; |
| **Header file:** | #include "ara/per/key\_value\_storage.h" |
| **Description** | 拷贝赋值运算符，不应被使用 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00010] ~KeyValueStorage()

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00010 |
| **Type** | Valid |
| **Priority** | H |
| **CR** |  |
| **Upstream ID** | [SWS\_PER\_00050] |
| Consistency | Yes |
| **Change Type** | 不变 |
| **Kind** | Function |
| **Symbol** | ~KeyValueStorage() |
| **Scope** | class ara::per::KeyValueStorage |
| **Syntax** | ~KeyValueStorage () noexcept; |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | no |
| **Header file:** | #include "ara/per/key\_value\_storage.h" |
| **Description** | 析构函数 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00011] GetAllKeys

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00011 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00042]、[SWS\_PER\_00505][2111A] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 修改 增加错误返回类型 | |
| **Kind** | Function | |
| **Symbol** | GetAllKeys() | |
| **Scope** | class ara::per::KeyValueStorage | |
| **Syntax** | ara::core::Result<ara::core::Vector<ara::core::String>> GetAllKeys () const noexcept; | |
| **Return value** | ara::core::Result< ara::core::Vector< ara::core::String > > | 结果，包含可用键的列表，或为PerErrc中的定义的错误之一。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Errors:** | PerErrc::kPhysicalStorageFailure | 如果访问存储失败，返回。 |
| PerErrc::kIntegrityCorrupted | 如果由于结构完整性损坏而无法读取存储的数据，则返回。 |
| PerErrc::kValidationFailed | 如果不能确保存储数据的有效性，则返回。. |
| PerErrc::kEncryptionFailed | 如果解密数据失败将返回 |
| PerErrc::kNotInitialized | 如果在ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | |
| **Description** | 返回键值库的可用的键的列表  必须在没有key增加和删除的时候取出的列表才准确。 | |
| **Additional:** | Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  auto keysResult = db->GetAllKeys();  keysResult里返回的是key的列表  **如果临时存储为空，则返回一个空的 ara::core::Vector（P1）** | |

#### [SWRD\_API\_Per\_00012] KeyExists

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00012 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |  |
| --- | --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00043]、[SWS\_PER\_00504] | | |
| **CR** |  | | |
| Consistency | Yes | | |
| **Change Type** | 修改 接口名，增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | KeyExists(ara::core::StringView key) | | |
| **Scope:** | class ara::per::KeyValueStorage | | |
| **Syntax:** | ara::core::Result<bool> KeyExists (ara::core::StringView key) const noexcept; | | |
| **Parameters (in):** | key | 要检查的key | |
| **Return value:** | ara::core::Result< bool > | 结果，如果可以找到键，则返回true;如果不能，则返回false，或为PerErrc中的定义的错误之一。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
| PerErrc::kIntegrityCorrupted | | 如果由于结构完整性损坏而无法读取存储的数据，则返回。 |
| PerErrc::kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
| PerErrc::kEncryptionFailed | | 如果解密数据失败将返回 |
| PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | | |
| **Description:** | 校验键值库中是否存在指定的键（原来叫HasKey，修改函数名P1修改） | | |
| **Additional:** | Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  Bool bl =db->KeyExists(“key”).Value()  bl是true：存在这个key，false：不存在这个key | | |

#### [SWRD\_API\_Per\_00013] GetValue

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00013 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |  |
| --- | --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00332]、[SWS\_PER\_00496][2111A]、[SWS\_PER\_00497][2111A]、[SWS\_PER\_00498][2111A]、[RS\_PER\_00002]、[RS\_PER\_00003] | | |
| Consistency | Yes | | |
| CR |  | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | GetValue(ara::core::StringView key) | | |
| **Scope:** | class ara::per::KeyValueStorage | | |
| **Syntax:** | template <class T>  ara::core::Result<T> GetValue (ara::core::StringView key) const noexcept; | | |
| **Template param:** | T | 检索值的类型 | |
| **Parameters (in):** | key | 查找的key | |
| **Return value:** | ara::core::Result< T > | 结果，检索到的值，或为定义的错误或者为供应商的错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kKeyNotFound | | 如果提供的键不存在于键值存储中，则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果由于结构完整性损坏而无法读取存储的数据，则返回。 |
|  | PerErrc::kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc::kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc::kDataTypeMismatch | | 如果存储值的数据类型与模板类型不匹配则返回 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize之后调用此函数将返回。 |
|  | PerErrc::kDataTypeMismatch | | 如果模板不数据类型匹配存储的数据类型 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | | |
| **Description:** | Persistency首先检查临时存储中是否存在key-value对，否则直接返回kKeyNotFound。对于现有的键值对，Persistency 会检查模板是否数据类型匹配存储的数据类型，否则直接返回kDataTypeMismatch。  GetValue可能会被延迟获取由于别的线程正在RemoveAllKeys，丢弃修改，或者对同一个key进行SetValue, RemoveKey, RecoverKey, 或者ResetKey。 | | |
| **Additional:** | Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  KvsType storages;  db->GetValue("KvsPerPorts", storages);  storages里是返回的KvsPerPorts键的值  **P1** | | |

#### [SWRD\_API\_Per\_00014] SetValue

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00014 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00046]、[SWS\_PER\_000499][2111A] | | |
| **CR** |  | | |
| Consistency | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | SetValue(ara::core::StringView key, const T &value) | | |
| **Scope:** | class ara::per::KeyValueStorage | | |
| **Syntax:** | template <class T>  ara::core::Result<void> SetValue (ara::core::StringView key, const T &value) noexcept; | | |
| **Template param:** | T | 要设置的值的类型 | |
| **Parameters (in):** | key | 键 | |
| value | 值 | |
| **Return value:** | ara::core::Result< void > | 结果，为空或为PerErrc中的定义的错误之一，或供应商错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kIllegalWriteAccess | | 如果键值存储被配置为只读则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc::kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法写入，则返回。 |
|  | PerErrc::kEncryptionFailed | | 如果存储的数据加密或解密失败则返回。 |
|  | PerErrc::kDataTypeMismatch | | 如果已存储值的数据类型与模板类型不匹配，则返回。 |
|  | PerErrc::kOutOfStorageSpace | | 如果在增加或者更新数据的时候存储空间不足将返回。 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize之后调用此函数将返回。 |
|  | PerErrc::kDataTypeMismatch | | 如果模板不数据类型匹配存储的数据类型 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | | |
| **Description:** | 为键值库的一个键设置值，  如果指定键的值已经存在，并且具有与新值相同的数据类型，则会覆盖该值。如果新值的数据类型与存储值不同，则返回kDataTypeMismatch。  SetValue可能会被延迟由于别的线程正在RemoveAllKeys，同步存储，或者抛弃修改，或者对同一个key进行SetValue, GetValue, RemoveKey, RecoverKey, 或 ResetKey.  Persistency 会检查模板是否数据类型匹配存储的数据类型，否则直接返回kDataTypeMismatch。 | | |
| **Additional:** | 调用后，需要调用此类的SyncToStorage完成向物理存储介质的存储，否则进程结束后，变化的信息不会存储到物理介质。  Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  const String kStringValue = "This is string value";  db->SetValue(“key”, kStringValue);  db->SyncToStorage();  向KVS数据库中写入key这对键值对。 | | |

#### [SWRD\_API\_Per\_00015] RemoveKey

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00015 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00047]、[SWS\_PER\_00502] | | |
| **CR** |  | | |
| Consistency | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | RemoveKey(ara::core::StringView key) | | |
| **Scope:** | class ara::per::KeyValueStorage | | |
| **Syntax:** | ara::core::Result<void> RemoveKey (ara::core::StringView key) noexcept; | | |
| **Parameters (in):** | key | 要删除的键 | |
| **Return value:** | ara::core::Result< void > | 结果，为空或为PerErrc中的定义的错误之一。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kKeyNotFound | | 如果提供的键不存在于键值存储中，则返回。（P1） |
|  | PerErrc::kIllegalWriteAccess | | 如果键值存储被配置为只读则返回 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc::kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法写入，则返回。 |
|  | PerErrc::kEncryptionFailed | | 如果存储的数据加密或解密失败则返回。(P1) |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | | |
| **Description:** | 从键值库中删除指定的键和对应的值  移除键可能会被延迟当别的线程正在RemoveAllKeys，同步存储或抛弃更改操作，或者对同一个key进行SetValue, GetValue, RemoveKey, RecoverKey, 或ResetKey。 | | |
| **Additional:** | 调用后，需要调用此类的SyncToStorage完成向物理存储介质的存储，否则进程结束后，变化的信息不会存储到物理介质。  Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  db->RemoveKey(“key”);  db->SyncToStorage();  从KVS数据库中删掉key这对键值对。 | | |

#### [SWRD\_API\_Per\_00016] RecoverKey

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00016 | | |
| **Type** | Valid | | |
| **Priority** | L | | |
| **CR** |  | | |
| **Upstream ID** | [SWS\_PER\_00427] | | |
| Consistency | Yes | | |
| **Change Type** | 新增 | | |
| **Kind:** | function | | |
| **Symbol:** | RecoverKey(ara::core::StringView key) | | |
| **Scope:** | class ara::per::KeyValueStorage | | |
| **Syntax:** | ara::core::Result<void> RecoverKey (ara::core::StringView key) noexcept; | | |
| **Parameters (in):** | key | 要重置的键 | |
| **Return value:** | ara::core::Result< void > | 结果，为空或为PerErrc中的定义的错误之一。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kKeyNotFound | | 如果提供的键不存在于键值存储中，则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc::kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法写入，则返回。 |
|  | PerErrc::kEncryptionFailed | | 如果存储的数据加密或解密失败则返回。 |
|  | PerErrc::kOutOfStorageSpace | | 当可用存储空间不足恢复值时返回 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | | |
| **Description:** | 从键值库中覆盖指定的键和对应的值  这个方法是覆盖掉单个key，当冗余检查失败时。  这个方法会尽力恢复key。覆盖以后，key可能是过时了或者初始值，或者丢失了。  覆盖key可能会被延迟当别的线程正在RemoveAllKeys，同步存储或抛弃更改操作，或者对同一个key进行SetValue, GetValue, RemoveKey, RecoverKey, 或ResetKey | | |
| **Additional:** | Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  db ->RecoverKey(“key”); | | |

#### [SWRD\_API\_Per\_00017] ResetKey

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00017 | | |
| **Type** | Valid | | |
| **Priority** | M | | |
| **CR** |  | | |
| **Upstream ID** | [SWS\_PER\_00426] | | |
| Consistency | Yes | | |
| **Change Type** | 新增 | | |
| **Kind:** | function | | |
| **Symbol:** | ResetKey (ara::core::StringView key) | | |
| **Scope:** | class ara::per::KeyValueStorage | | |
| **Syntax:** | ara::core::Result<void> ResetKey (ara::core::StringView key) noexcept; | | |
| **Parameters (in):** | Key | 要重置的键 | |
| **Return value:** | ara::core::Result< void > | 结果，为空或为PerErrc中的定义的错误之一。 | |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kIllegalWriteAccess | | 如果键值存储被配置为只读则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc::kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法写入，则返回。 |
|  | PerErrc::kEncryptionFailed | | 如果存储的数据加密或解密失败则返回。 |
|  | PerErrc::kInitValueNotAvailable | | 如果没有为该键配置初始值，则返回。 |
|  | PerErrc::kOutOfStorageSpace | | 当可用存储空间不足恢复值时返回。 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | | |
| **Description:** | 将此键-值存储的键重置为其初始值。  这个方法是重置单个key到初始化的value。如果该键当前在键值存储中不可用，则会重新创建该键。  当设计和部署时没有给key初始value时，ResetKey将失败并返回kInitValueNotAvailable。  ResetKey可能会被延时当别的线程正在RemoveAllKeys，同步存储或抛弃更改操作，或者对同一个key进行SetValue, GetValue, RemoveKey, RecoverKey, 或ResetKey。 | | |
| **Additional:** | Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  db ->ResetKey (“key”);  用初始化数据覆盖数据库中key这对键值对。  P2做 | | |

#### [SWRD\_API\_Per\_00018] RemoveAllKeys

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00018 | | |
| **Upstream ID** | [SWS\_PER\_00048]、[SWS\_PER\_00503] | | |
| Consistency | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | RemoveAllKeys() | | |
| **Scope:** | class ara::per::KeyValueStorage | | |
| **Syntax:** | ara::core::Result<void> RemoveAllKeys () noexcept; | | |
| **Return value:** | ara::core::Result< void > | 结果，为空或为PerErrc中的定义的错误之一。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kIllegalWriteAccess | | 如果键值存储被配置为只读则返回。 |
|  | PerErrc:: kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc::kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法写入，则返回。 |
|  | PerErrc::kEncryptionFailed | | 如果存储的数据加密或解密失败则返回。 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | | |
| **Description:** | 删除键值库中所有的键及关联的值  RemoveAllKeys 可能会被延迟当别的线程正在RemoveAllKeys，同步存储或抛弃更改操作，或者进行SetValue, GetValue, RemoveKey, RecoverKey, 或ResetKey。 | | |
| **Additional:** | 调用后，需要调用此类的SyncToStorage完成向物理存储介质的存储，否则进程结束后，变化的信息不会存储到物理介质。  Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  db->RemoveAllKeys();  删掉数据库中的所有数据。 | | |

#### [SWRD\_API\_Per\_00019] SyncToStorage

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00019 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **CR** |  | | |
| **Upstream ID** | [SWS\_PER\_00049]、[SWS\_PER\_000494][2111A]、[RS\_PER\_00001] | | |
| Consistency | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | SyncToStorage() | | |
| **Scope:** | class ara::per::KeyValueStorage | | |
| **Syntax:** | ara::core::Result<void> SyncToStorage ()noexcept; | | |
| **Return value:** | ara::core::Result< void > | 结果，为空或为PerErrc中的定义的错误之一。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kIllegalWriteAccess | | 如果键值存储被配置为只读则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc::kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法写入，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据加密失败则返回。 |
|  | PerErrc::kOutOfStorageSpace | | 当可用存储空间不足以存储新增/更新的值时返回 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | | |
| **Description:** | 将键值库的更新刷新到物理存储  同步数据可以能被延时当别的线程正在RemoveAllKeys，抛弃更改操作，或者进行SetValue, RemoveKey, RecoverKey, 或ResetKey。.  当调用 ara::per::KeyValueStorage::SyncToStorage 时，Persistency 将永久存储自上次调用此方法或自打开键值存储以来对键值存储所做的所有更改。Persistency还应更新此调用中的任何配置冗余。 | | |
| **Additional:** | Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  const String kStringValue = "This is string value";  db->SetValue(“key”, kStringValue);  db->SyncToStorage();  同步到磁盘 | | |

#### [SWRD\_API\_Per\_00020] DiscardPendingChanges

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00020 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **CR** |  | | |
| **Upstream ID** | [SWS\_PER\_00365]、[SWS\_PER\_00495][2111A] 、[RS\_PER\_00001] | | |
| Consistency | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | DiscardPendingChanges() | | |
| **Scope:** | class ara::per::KeyValueStorage | | |
| **Syntax:** | ara::core::Result<void> DiscardPendingChanges（） noexcept; | | |
| **Return value:** | ara::core::Result< void > | 结果，为空或为PerErrc中的定义的错误之一。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc::kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc::kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/key\_value\_storage.h" | | |
| **Description:** | 丢弃所有由上次调用SyncToStorage()或者用OpenKeyValueStorage()打开key-value的存储待改变的key-value数据存储的修改。  DiscardPendingChanges可能被延时当别的线程正在RemoveAllKeys SyncToStorage或者抛弃更改操作，或者进行SetValue, GetValue ，RemoveKey, RecoverKey, 或ResetKey。.  当 ara::per::KeyValueStorage::DiscardPendingChanges 被调用时，Persistency 会将 Key-Value Storage 重置为最后的持久化状态，即最后一次调用 SyncToStorage 或打开 Key-Value Storage 后的状态 | | |
| **Additional:** | Auto result =ara::per::OpenKeyValueStorage(“/NeuSAR/appPersistencyDemo/portKVS”);  Auto db=std::move(result).Value();  const String kStringValue = "This is string value";  db->SetValue(“key”, kStringValue);  db->DiscardPendingChanges();  Bool bl =db->KeyExists(“key”).Value()  调用了DiscardPendingChanges()后发现key不存在，被丢弃了。 | | |

## File Storage 接口定义

这章列出了操作File Storage所需的所有函数和类。

这些功能可以用来访问File Storage，尽可能修复毁坏的storage，重置到部署的默认值，并得到存储分配给文件存储的数量。此外，操作者结合ara::per::OpenMode传递模式给OpenFile\* functions。

Per访问到文本模式时不会改变和解释文件内容。假如文件在文件存储编码为utf - 8(见RS\_AP\_00136]; 这也是符合约束StdCppImplementationDataType类别的字符串在[3],[constr\_1674])。也认为根据UNIX约定处理的结局,即换行(“\ n”)。

### [SWRD\_API\_Per\_00021] OpenFileStorage

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00021 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **CR** |  | | |
| **Upstream ID** | [SWS\_PER\_00116] | | |
| Consistency | No | | |
| **Change Type** | 修改 参数变更，增加错误返回类型，增加描述 | | |
| **Kind:** | function | | |
| **Symbol:** | OpenFileStorage(const ara::core::InstanceSpecifier &fs) | | |
| **Scope:** | namespace ara::per | | |
| **Syntax:** | ara::core::Result<SharedHandle<FileStorage> > OpenFileStorage  (const ara::core::InstanceSpecifier &fs) noexcept; | | |
| **Parameters (in):** | fs | 由PersistencyFileProxyInterface输入的PortPrototype的短名称路径。 | |
| **Return value:** | ara::core::Result< SharedHandle< File Storage > > | 包含SharedHandle的结果用于文件存储，或为PerErrc中定义的错误之一。 | |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kStorageNotFound | | 如果传递的InstanceSpecifier与为该可执行文件配置的任何PersistencyKeyValueStorageInterface不匹配，则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果当前正在执行updatepersistence或resetpersistence，或者当前正在为相同的文件存储执行RecoverAllFiles或ResetAllFiles，则返回。 |
|  | PerErrc:: kOutOfStorageSpace | | 添加/更新文件的可用存储空间不足时返回 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 打开文件存储.  打开文件存储将会失败返回kResourceBusy，当文件存储当前正在被别的线程做UpdatePersistency, ResetPersistency, RecoverAllFiles, 或ResetAllFiles修改  因为多线程同时访问同一个文件。当SharedHandle返回超出范围文件存储数据库不会被关闭。它只会被关闭,当所有引用SharedHandles都超出控制范围。 | | |
| **Additional:** | 代码示例及类句柄的获取说明：  auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息为:result.Error().Message();  //相应的处理  }  auto proxy = std::move(result).Value(); | | |

### [SWRD\_API\_Per\_00022] RecoverAllFiles

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00022 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **CR** |  | | |
| **Upstream ID** | [SWS\_PER\_00335] | | |
| Consistency | No | | |
| **Change Type** | 修改 参数变更，增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | RecoverAllFiles(const ara::core::InstanceSpecifier &fs) | | |
| **Scope:** | namespace ara::per | | |
| **Syntax:** | ara::core::Result<void> RecoverAllFiles  (const ara::core::InstanceSpecifier &fs) noexcept; | | |
| **Parameters (in):** | fs | 由PersistencyFileProxyInterface输入的PortPrototype的短名称路径。 | |
| **Return value:** | ara::core::Result<void > | 包含void，或为PerErrc中定义的错误之一。 | |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kStorageNotFound | | 如果传递的InstanceSpecifier与为该可执行文件配置的任何PersistencyKeyValueStorageInterface不匹配，则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据加密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果updatepersistence或resetpersistence正在执行，或者ResetAllFiles正在为同一个文件存储执行，或者同一个文件存储的SharedHandle正在使用，则返回。 |
|  | PerErrc:: kOutOfStorageSpace | | 恢复文件的可用存储空间不足时返回 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 恢复文件存储，包括所有文件。  RecoverAllFiles恢复文件存储当检查出冗余错误时。  当文件存储数据库当前是被打开或者被别的线程修改UpdatePersistency, ResetPersistency, RecoverAllFiles, 或ResetAll Files.时，将会返回kResourceBusy失败。  这个方法是最有效的恢复所有的文件。恢复以后，文件可能会过时，或者是初始数据，或可能丢失。 | | |
| **Additional:** | 调用::ara::per:: RecoverAllFiles ("/NeuSAR/appPersistencyDemo/portFS")//参数是port口  恢复文件存储，包括所有文件  依赖文件冗余。  **P2** | | |

### [SWRD\_API\_Per\_00023] ResetAllFiles

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00023 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00336] | | |
| **CR** |  | | |
| Consistency | No | | |
| **Change Type** | 修改 参数变更，增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | ResetAllFiles (const ara::core::InstanceSpecifier &fs) | | |
| **Scope:** | namespace ara::per | | |
| **Syntax:** | ara::core::Result<void> ResetAllFiles  (const ara::core::InstanceSpecifier &fs) noexcept; | | |
| **Parameters (in):** | fs | 由PersistencyFileProxyInterface输入的PortPrototype的短名称路径。 | |
| **Return value:** | ara::core::Result<void > | 包含void，或为PerErrc中定义的错误之一。 | |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kStorageNotFound | | 如果传递的InstanceSpecifier与为该可执行文件配置的任何PersistencyKeyValueStorageInterface不匹配，则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据加密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果当前正在执行updatepersistence或resetpersistence，或者当前正在为同一个文件存储执行RecoverAllFiles，或者当前正在使用同一个文件存储的SharedHandle，则返回。 |
|  | PerErrc:: kOutOfStorageSpace | | 恢复文件的可用存储空间不足时返回 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 重置文件存储，包括所有的文件。  RecoverAllFiles重置文件存储数据库到初始状态。包括部署时从manifest定义的文件初始数据。  当文件存储数据库当前是被打开或者被别的线程修改UpdatePersistency, ResetPersistency, RecoverAllFiles, 或ResetAll Files.时，将会返回kResourceBusy失败。 | | |
| **Additional:** | 调用::ara::per:: ResetAllFiles ("/NeuSAR/appPersistencyDemo/portFS")//参数是port口  用初始数据重置文件存储，包括所有的文件。 | | |

### [SWRD\_API\_Per\_00024] GetCurrentFileStorageSize

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00024 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **CR** |  | | |
| **Upstream ID** | [SWS\_PER\_00406]、[SWS\_PER\_00492][2111A] | | |
| Consistency | No | | |
| **Change Type** | 修改 参数变更，增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | GetCurrentFileStorageSize(const ara::core::InstanceSpecifier &fs) | | |
| **Scope:** | namespace ara::per | | |
| **Syntax:** | ara::core::Result<uint64\_t> GetCurrentFileStorageSize  (const ara::core::InstanceSpecifier &fs) const noexcept; | | |
| **Parameters (in):** | fs | 由PersistencyFileProxyInterface输入的PortPrototype的短名称路径。 | |
| **Return value:** | ara::core::Result < uint64\_t > | 结果包括使用的字节空间，或为PerErrc中定义的错误之一。 | |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kStorageNotFound | | 如果传递的InstanceSpecifier与为该可执行文件配置的任何PersistencyKeyValueStorageInterface不匹配，则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 返回当前文件存储数据库占用的空间。  返回的size包括元数据、冗余数据和备份数据占用的空间。返回的size只有在文件存储没有被操作时才准确。 | | |
| **Additional:** | 调用::ara::per::GetCurrentFileStorageSize("/NeuSAR/appPersistencyDemo/portFS") //参数是port口  返回当前文件存储数据库占用的空间。  GetCurrentFileStorageSize应返回当前分配给 File Storage 的存储空间的总大小，包括管理数据、冗余数据和备份数据(**目前实现没包含文件冗余和crc文件的大小， P1实现**) | | |

### [SWRD\_API\_Per\_00025] OpenMode

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00025 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00147] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 修改 变更作用域，变更描述，变更类型，变更枚举值，变更头文件 | |
| **Kind:** | enumeration | |
| **Symbol:** | OpenMode | |
| **Scope:** | namespace ara::per | |
| **Underlying type:** | uint32\_t | |
| **Syntax** | enum class OpenMode : uint32\_t {...}; | |
| **Values:** | kAtTheBeginning= 1 << 0 | 当文件打开时，操作位置到文件开始的位置，这种模式不能结合kAtTheEnd。 |
| kAtTheEnd= 1 << 1 | 当文件打开时，操作位置到文件结尾的位置，这种模式不能结合kAtTheBeginning 或kTruncate.。 |
| kTruncate= 1 << 2 | 当文件打开时，移除已经存在的内容，  这种模式不能结合kAtTheEnd。 |
| kAppend= 1 << 3 | 在写准确找到文件的尾部，应用到结尾。 |
| **Header file:** | #include "ara/per/file\_storage.h" | |
| **Description:** | **文件怎么打开定义的枚举**。值可能包含(using | and |=) 只要不相互矛盾。 | |
| **Additional:** | auto storageResult = OpenFileStorage("/NeuSAR/appPersistencyDemo/portFS");  auto storage = std::move(storageResult).Value();  constexpr auto kFileName = "file";  auto fileResult = storage->OpenFileReadWrite(kFileName, BasicOperations::OpenMode::kTruncate) | |

### [SWRD\_API\_Per\_00026] operator |

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00026 | |
| **Type** | Valid | |
| **Priority** | H | |
| **CR** |  | |
| **Upstream ID** | [SWS\_PER\_00144] | |
| Consistency | No | |
| **Change Type** | 修改 变更头文件，变更参数作用域 | |
| **Kind:** | function | |
| **Symbol:** | operator|(OpenMode left, OpenMode right) | |
| **Scope:** | namespace ara::per | |
| **Syntax:** | constexpr OpenMode operator| (OpenMode left, OpenMode right); | |
| **Parameters (in):** | left | 第一个OpenMode修饰符 |
| right | 第二个OpenMode修饰符 |
| **Return value:** | OpenMode | 返回合并的OpenMode修饰符。 |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/basic\_operations.h" | |
| **Description:** | 将两个OpenMode修饰符合并为一个 | |
| **Additional:** |  | |

### [SWRD\_API\_Per\_00027] operator|=

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00027 | |
| **Type** | Valid | |
| **Priority** | H | |
| **CR** |  | |
| **Upstream ID** | [SWS\_PER\_00434] | |
| Consistency | Yes | |
| **Change Type** | 新增 | |
| **Kind:** | function | |
| **Symbol:** | operator|=(OpenMode &left, const OpenMode &right) | |
| **Scope:** | namespace ara::per | |
| **Syntax:** | OpenMode& operator|= (OpenMode &left, const OpenMode &right); | |
| **Parameters (in):** | left | 左OpenMode修饰符 |
| right | 右OpenMode修饰符 |
| **Return value:** | OpenMode & | 返回OpenMode的修改。 |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/file\_storage.h" | |
| **Description:** | Merges an OpenMode modifier into this OpenMode.  合并成一个OpenMode修饰符到打开模式 | |
| **Additional:** |  | |

### [SWRD\_API\_Per\_00028] FileCreationState

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00028 | |
| **Type** | Valid | |
| **Priority** | H | |
| **CR** |  | |
| **Upstream ID** | [SWS\_PER\_00435] | |
| Consistency | Yes | |
| **Change Type** | 新增 | |
| **Kind:** | enumeration | |
| **Symbol:** | FileCreationState | |
| **Scope:** | namespace ara::per | |
| **Underlying type:** | uint32\_t | |
| **Syntax** | enum class FileCreationState : uint32\_t {...}; | |
| **Values:** | kCreatedDuringInstallion= 1 | APP安装或者重置Per以后文件被Per创建。 |
| kCreatedDuringUpdate= 2 | 更新过程中文件被创建。 |
| kCreatedDuringReset= 3 | 文件会被重新创建在重置文件或者重置所有文件的时候。 |
| kCreatedDuringRecovery= 4 | 损坏的文件被删除，文件会被重新创建。 |
|  | kCreatedByApplication= 5 | 文件会被创建被APP。 |
| **Header file:** | #include "ara/per/file\_storage.h" | |
| **Description:** | 枚举描述了什么情况文件被创建。 | |
| **Additional:** | **P3** | |

### [SWRD\_API\_Per\_00029] FileModificationState

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00029 | |
| **Type** | Valid | |
| **Priority** | H | |
| **CR** |  | |
| **Upstream ID** | [SWS\_PER\_00436] | |
| Consistency | Yes | |
| **Change Type** | 新增 | |
| **Kind:** | enumeration | |
| **Symbol:** | FileModificationState | |
| **Scope:** | namespace ara::per | |
| **Underlying type:** | uint32\_t | |
| **Syntax** | enum class FileModificationState: uint32\_t {...}; | |
| **Values:** | kModifiedDuringUpdate= 2 | Per更新时，文件被最后修改。 |
| kModifiedDuringReset= 3 | Per被调用ResetFile 或ResetAllFiles时，文件被最后修改。 |
| kModifiedDuringRecovery= 4 | Per损坏的数据被抛弃时，文件被修改。 |
| kModifiedByApplication= 5 | APP对文件做最后修改。 |
| **Header file:** | #include "ara/per/file\_storage.h" | |
| **Description:** | 枚举描述了什么情况文件被修改。 | |
| **Additional:** | **P3** | |

### [SWRD\_API\_Per\_00030] FileInfo

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00030 |
| **Type** | Valid |
| **Priority** | L |
| **CR** |  |
| **Upstream ID** | [SWS\_PER\_00437] |
| Consistency | Yes |
| **Change Type** | 新增 |
| **Kind:** | Struct |
| **Symbol:** | FileInfo |
| **Scope:** | namespace ara::per |
| **Syntax** | struct FileInfo {...}; |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 被GetFileInfo返回的结构包括文件额外的信息。 |
| **Additional:** | **P3** |

#### [SWRD\_API\_Per\_00031] creationTime

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00031 |
| **Type** | Valid |
| **Priority** | L |
| **Upstream ID** | [SWS\_PER\_00441] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 新增 |
| **Kind:** | variable |
| **Symbol:** | creationTime |
| **Scope:** | struct ara::per::FileInfo |
| **Type** | uint64\_t |
| **Syntax** | uint64\_t creationTime; |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 文件创建的时间是从1970-01-01 UTC午夜开始记时用nanoseconds计数。 |
| **Additional:** | **P3** |

#### [SWRD\_API\_Per\_00032] modificationTime

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00032 |
| **Type** | Valid |
| **Priority** | L |
| **Upstream ID** | [SWS\_PER\_00442] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 新增 |
| **Kind:** | variable |
| **Symbol:** | modificationTime |
| **Scope:** | struct ara::per::FileInfo |
| **Type** | uint64\_t |
| **Syntax** | uint64\_t modificationTime; |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 文件最后修改的时间是从1970-01-01 UTC午夜开始记时用nanoseconds计数。 |
| **Additional:** | **P3** |

#### [SWRD\_API\_Per\_00033] accessTime

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00033 |
| **Type** | Valid |
| **Priority** | L |
| **Upstream ID** | [SWS\_PER\_00443] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 新增 |
| **Kind:** | variable |
| **Symbol:** | accessTime |
| **Scope:** | struct ara::per::FileInfo |
| **Type** | uint64\_t |
| **Syntax** | uint64\_t accessTime; |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 文件最后访问的时间是从1970-01-01 UTC午夜开始记时用nanoseconds计数。 |
| **Additional:** | **P3** |

#### [SWRD\_API\_Per\_00034] fileCreationState

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00034 |
| **Type** | Valid |
| **Priority** | L |
| **Upstream ID** | [SWS\_PER\_00444] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 新增 |
| **Kind:** | variable |
| **Symbol:** | fileCreationState |
| **Scope:** | struct ara::per::FileInfo |
| **Type** | FileCreationState |
| **Syntax** | FileCreationState fileCreationState; |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 返回的信息：谁怎么创建的文件。 |
| **Additional:** | **P3** |

#### [SWRD\_API\_Per\_00035] fileModificationState

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00035 |
| **Type** | Valid |
| **Priority** | L |
| **Upstream ID** | [SWS\_PER\_00445] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 新增 |
| **Kind:** | variable |
| **Symbol:** | fileModificationState |
| **Scope:** | struct ara::per::FileInfo |
| **Type** | FileModificationState |
| **Syntax** | FileModificationState fileModificationState; |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 返回的信息：谁怎么修改的文件。 |
| **Additional:** | **P3** |

### [SWRD\_API\_Per\_00036] FileStorage 类

本节展示了调用[SWS\_PER\_00116]获得的ara::per::FileStorage对象可用的方法。

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00036 |
| **Type** | Valid |
| **Priority** | L |
| **Upstream ID** | [SWS\_PER\_00340] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 修改 类修饰符 |
| **Kind:** | class |
| **Symbol:** | FileStorage |
| **Scope:** | namespace ara::per |
| **Syntax:** | class FileStorage final {...}; |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 文件存储包括一组确定的标识用他们的名字 |
| **Additional:** | 代码示例及类句柄的获取说明：  auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息为:result.Error().Message();  //相应的处理  }  auto proxy = std::move(result).Value();  此proxy即为此类的指针。**FileStorage改为final类，P1** |

#### [SWRD\_API\_Per\_00037] FileStorage

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00037 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00326] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | FileStorage(FileStorage &&fs) | |
| **Scope:** | class ara::per::FileStorage | |
| **Syntax:** | FileStorage (FileStorage &&fs) noexcept; | |
| **Parameters (in):** | fs | 要移动的FileStorage对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/file\_storage.h" | |
| **Description:** | 移动构造函数 | |
| **Additional:** |  | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00038 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00328] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 不变 |
| **Kind:** | function |
| **Symbol:** | FileStorage(const FileStorage &) |
| **Scope:** | class ara::per::FileStorage |
| **Syntax:** | FileStorage (const FileStorage &)=delete; |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 拷贝构造函数，不应被使用 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00039] operator=

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00039 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00327] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | operator=(FileStorage &&fs); | |
| **Scope:** | class ara::per::FileStorage | |
| **Syntax:** | FileStorage& operator= (FileStorage &&fs) &noexcept; | |
| **Parameters (in):** | fs | 要移动的FileStorage对象 |
| **Return value:** | FileStorage & | 移动后的FileStorage对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/file\_storage.h" | |
| **Description:** | 移动赋值运算符 | |
| **Additional:** |  | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00040 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00329] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 不变 |
| **Kind:** | function |
| **Symbol:** | operator=(const FileStorage &) |
| **Scope:** | class ara::per::FileStorage |
| **Syntax:** | FileStorage& operator= (const FileStorage &)=delete; |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 拷贝赋值运算符，不应被使用 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00041] ~FileStorage

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00041 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00330] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 不变 |
| **Kind:** | function |
| **Symbol:** | ~FileStorage() |
| **Scope:** | class ara::per::FileStorage |
| **Syntax:** | ~FileStorage () noexcept; |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | no |
| **Header file:** | #include "ara/per/file\_storage.h" |
| **Description:** | 析构函数 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00042] GetAllFileNames

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00042 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00110]、[SWS\_PER\_00509] | | |
| **CR** |  | | |
| Consistency | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | GetAllFileNames() | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<ara::core::Vector<ara::core::String> > GetAllFileNames () const noexcept; | | |
| **Return value:** | ara::core::Result< ara::core::Vector< ara::core::String > > | 结果，包含可用文件列表，或为PerErrc中定义的错误之一。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 返回当前文件存储中可用的文件列表  这个列表只在没有同时被操作增加或删除时有效。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  auto rt= fs->GetAllFileNames();  keysResult返回当前文件存储中可用的文件列表  **如果临时存储为空，则返回一个空的 ara::core::Vector（P1）** | | |

#### [SWRD\_API\_Per\_00043] DeleteFile

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00043 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00111]、[SWS\_PER\_00510] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 修改 增加错误返回类型 | |
| **Kind:** | function | |
| **Symbol:** | DeleteFile(ara::core::StringView fileName) | |
| **Scope:** | class ara::per::FileStorage | |
| **Syntax:** | ara::core::Result<void> DeleteFile (ara::core::StringView fileName) noexcept; | |
| **Parameters (in):** | fileName | 文件名。可能对应配置文件中的PersistencyFile.fileName。 |
| **Return value:** | ara::core::Result< void > | 结果，为空或为PerErrc中的定义的错误之一。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
|  | PerErrc::kIllegalWriteAccess | 如果文件存储被配置为只读则返回。 |
|  | PerErrc:: kPhysicalStorageFailure | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | 如果存储的数据因结构完整性损坏而无法写入，则返回。 |
|  | PerErrc:: kEncryptionFailed | 如果存储的数据加密或解密失败则返回。(P2) |
|  | PerErrc:: kResourceBusy | 如果文件是打开的，或者相同文件名的RecoverFile或ResetFile当前正在执行，则返回 |
|  | PerErrc:: kFileNotFound | 如果提供的文件不存在于文件存储中，则返回。（P1） |
|  | PerErrc:: kNotInitialized | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | |
| **Description:** | 从该文件存储中删除指定的文件。  当文件当前打开时，该操作将失败，并带有kResourceBusyError。 | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  auto rt= fs->GetAllFileNames();  fs->DeleteFile(“filename”);  删掉文件库中filename的文件 | |

#### [SWRD\_API\_Per\_00044] FileExists

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00044 | | |
| **Type** | | Valid | |
| **Priority** | | H | |
| **Upstream ID** | [SWS\_PER\_00112]、[SWS\_PER\_00508] | | |
| **CR** |  | | |
| Consistency | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | ara::per::FileStorage::FileExists(ara::core::StringView fileName) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<bool> FileExists (ara::core::StringView fileName) const noexcept; | | |
| **Parameters (in):** | fileName | | 文件名，可能对应配置文件中的Per File.fileName |
| **Return value:** | ara::core::Result< bool > | | 一个结果，如果文件存在，则返回true，如果不存在，则返回false，或为PerErrc中的定义的错误之一。 |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 查询文件存储中是否有指定的文件存在  结果只有在没有文件被增加删除时才有效。  例如，当一个文件正在被另一个线程移除，返回的还是true，那么结果就不准确了。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  bool bl = fs-> FileExists (“filename”);  返回文件库中是否有filename这个文件。  **P2** | | |

#### [SWRD\_API\_Per\_00045] RecoverFile

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00045 | | |
| **Type** | | Valid | |
| **Priority** | | H | |
| **Upstream ID** | [SWS\_PER\_00337] | | |
| **CR** |  | | |
| Consistency | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | ara::per::FileStorage:: RecoverFile (ara::core::StringView fileName) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<void> RecoverFile (ara::core::StringView fileName) noexcept; | | |
| **Parameters (in):** | fileName | | 文件名，可能对应配置文件中的Per File.fileName |
| **Return value:** | ara::core::Result< void> | | 结果，为空或为PerErrc中的定义的错误之一。 |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据加密或解密失败则返回。 |
|  | PerErrc::kResourceBusy | | 如果文件已打开，或者正在执行具有相同文件名的DeleteFile或Reset file，则返回。 |
|  | PerErrc::kOutOfStorageSpace | | 恢复文件的可用存储空间不足时返回 |
|  | PerErrc::kFileNotFound | | 如果提供的文件不存在于文件存储中，则返回。 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 覆盖文件存储中的文件  这个方法允许在检查出冗余错误时覆盖单个文件。当文件被打开时会报kResourceBusy错误。此方法尽最大努力恢复文件。覆盖以后，文件可能会过时或者初始值，或者丢失。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  fs-> RecoverFile(“filename”);  用冗余文件恢复filename这个文件。 | | |

#### [SWRD\_API\_Per\_00046] ResetFile

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00046 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00338] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 修改 增加错误返回类型，增加描述 | | |
| **Kind:** | function | | |
| **Symbol:** | ara::per::FileStorage:: ResetFile (ara::core::StringView fileName) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<void> ResetFile (ara::core::StringView fileName) noexcept; | | |
| **Parameters (in):** | fileName | 文件名，可能对应配置文件中的Per File.fileName | |
| **Return value:** | ara::core::Result< void> | 结果，为空或为PerErrc中的定义的错误之一。 | |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据加密或解密失败则返回。 |
|  | PerErrc::kInitValueNotAvailable | | 如果没有为该文件配置初始值，则返回。 |
|  | PerErrc::kResourceBusy | | 如果文件是打开的，或者相同文件名的DeleteFile或RecoverFile正在执行，则返回。 |
|  | PerErrc::kOutOfStorageSpace | | 恢复文件时，如果可用存储空间不足或文件数量将大于配置的maxnumberfiles，则返回 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 对文件系统中的文件重置为初始化内容。  这个方法允许重置单个文件成初始化内容。如果当前文件存储数据库中不存在这个文件，将会重建。  当文件是打开的会返回错误kResourceBusy。当没有部署初始化数据时将会返回kInitValueNotAvailable。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  fs-> ResetFile(“filename”);  用初始文件恢复filename这个文件。  **P2** | | |

#### [SWRD\_API\_Per\_00047] GetCurrentFileSize

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00047 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00407]、[SWS\_PER\_00493][2111A] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | GetCurrentFileSize(ara::core::StringView fileName) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<uint64\_t>GetCurrentFileSize(ara::core::StringView fileName) const noexcept; | | |
| **Parameters (in):** | fileName | 文件名，可能对应配置文件中的Per File.fileName | |
| **Return value:** | ara::core::Result< uint64\_t > | 结果包括占用的字节空间。或为PerErrc中的定义的错误之一。 | |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法写入，则返回。 |
|  | PerErrc:: kFileNotFound | | 果提供的文件不存在于文件存储中，则返回。 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 返回当前File Storage中file占用的字节空间。  如果此时没有别的操作对文件空间，返回的size是准确的。  ara::per::FileStorage::GetCurrentFileSize 应返回传递文件的当前大小。 此大小应仅反映文件中包含的数据。 如果文件存在多个冗余实例， 应返回当前访问实例的大小 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  auto resultSize\_ = fs->GetCurrentFileSize(filename);  返回的是filename占用的字节空间 | | |

#### [SWRD\_API\_Per\_00048] GetFileInfo

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00048 | | |
| **Type** | | Valid | |
| **Priority** | | H | |
| **Upstream ID** | [SWS\_PER\_00438] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 新增 | | |
| **Kind:** | function | | |
| **Symbol:** | GetFileInfo(ara::core::StringView fileName) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<FileInfor>GetFileInfo (ara::core::StringView fileName) const noexcept; | | |
| **Parameters (in):** | fileName | | 文件名，可能对应配置文件中的Per File.fileName |
| **Return value:** | ara::core::Result< FileInfor > | | 结果包括FileInfo结构体。或为PerErrc中的定义的错误之一。 |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kFileNotFound | | 如果提供的文件不存在于文件存储中，则返回。 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 返回当前File Storage中file的额外信息。  返回的文件信息结构体里包括文件创建的时间，修改的时间，访问的时间。怎么创建的谁创建的。这些文件信息只有当文件没被打开时才准确。 | | |
| **Additional:** | **P2** | | |

#### [SWRD\_API\_Per\_00049] OpenFileReadWrite

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00049 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00375] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | OpenFileReadWrite(ara::core::StringView fileName) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<UniqueHandle<ReadWriteAccessor>>OpenFileReadWrite (ara::core::StringView fileName) noexcept; | | |
| **Parameters (in):** | fileName | 文件名, 可能对应配置文件中的Per File.fileName | |
| **Return value:** | ara::core::Result< UniqueHandle< ReadWriteAccessor > > | 一个结果，包含一个文件的UniqueHandle，或为PerErrc中定义的一个错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kIllegalWriteAccess | | 如果文件存储配置为只读则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc::kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc::kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc::kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc::kResourceBusy | | 如果文件已经打开，或者当前正在执行同名的DeleteFile、RecoverFile或ResetFile，则返回。 |
|  | PerErrc::kOutOfStorageSpace | | 创建文件时，如果可用存储空间不足或文件数量将大于配置的maxnumberfiles，则返回 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 在文件存储中打开指定的文件供读写。  文件打开时找到文件开始的位置（类似 kAtThe Beginning）  文件不存在，创建文件。  当返回UniqueHandle超出控范围时，文件将被关闭。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs->OpenFileReadWrite(kFileName);  if (!accessor) {  logger.LogError()<<"以读写方式打开文件失败，错误信息为："<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  \*ac<<::ara::core::String(kKeyStringValue);  ac->flush();  ac->fsync(); | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00050 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00113] | | |
| **CR** |  | | |
| **Consistency** | No | | |
| **Change Type** | 修改 增加错误返回类型，修改参数 | | |
| **Kind:** | function | | |
| **Symbol:** | OpenFileReadWrite(ara::core::StringView fileName, OpenMode mode) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<UniqueHandle<ReadWriteAccessor>>OpenFileReadWrite (ara::core::StringView fileName, OpenMode mode) noexcept; | | |
| **Parameters (in):** | fileName | 文件名, 可能对应配置文件中的Per File.fileName | |
| mode | 文件打开的模式 | |
| **Return value:** | ara::core::Result<UniqueHandle  <ReadWriteAccessor > > | 一个结果，包含一个文件的UniqueHandle，或为PerErrc中定义的一个错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kIllegalWriteAccess | | 如果文件存储配置为只读则返回 |
|  | PerErrc:: kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果文件已经打开，或者当前正在执行同名的DeleteFile、RecoverFile或ResetFile，则返回 |
|  | PerErrc:: kOutOfStorageSpace | | 创建文件时，如果可用存储空间不足或文件数量将大于配置的maxnumberfiles，则返回 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kInvalidOpenMode | | 如果传递的模式包含无效的模式组合，则返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 在文件存储中打开指定的文件供读写。  如果没有提供打开模式，默认是打开后到开始的位置（类似to kAtTheBeginning）。  如果文件不存在，新创建，文件UniqueHandle超出范围时文件将会被关闭。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs->OpenFileReadWrite(kFileName,BasicOperations::OpenMode(0));  if (!accessor) {  logger.LogError()<<"以读写方式打开文件失败，错误信息为："<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  \*ac<<::ara::core::String(kKeyStringValue);  ac->flush();  ac->fsync(); | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | | SWRD\_API\_Per\_00051 | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | | [SWS\_PER\_00429] | |
| **CR** | |  | |
| **Consistency** | | Yes | |
| **Change Type** | | 新增 | |
| **Kind:** | | function | |
| **Symbol:** | | OpenFileReadWrite(ara::core::StringView fileName, OpenMode mode, ara::core::Span< ara::core::Byte > buffer) | |
| **Scope:** | | class ara::per::FileStorage | |
| **Syntax:** | | ara::core::Result<UniqueHandle<ReadWriteAccessor>>OpenFileReadWrite (ara::core::StringView fileName, OpenMode mode, ara::core::Span< ara::core::Byte > buffer) noexcept; | |
| **Parameters (in):** | | fileName | 文件名, 可能对应配置文件中的PersistencyFile.fileName |
| mode | 文件打开的模式 |
|  | | Buffer | 分块译码读/写用的内存 |
| **Return value:** | | ara::core::Result<UniqueHandle  <ReadWriteAccessor > > | 一个结果，包含一个文件的UniqueHandle，或为PerErrc中定义的一个错误。 |
| **Exception Safety:** | | noexcept | |
| **Thread Safety:** | | re-entrant | |
|  | | PerErrc::kIllegalWriteAccess | 如果文件存储配置为只读则返回 |
|  | | PerErrc:: kPhysicalStorageFailure | 如果访问存储失败，返回。 |
|  | | PerErrc:: kIntegrityCorrupted | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | | PerErrc:: kValidationFailed | 如果不能确保存储数据的有效性，则返回。 |
|  | | PerErrc:: kEncryptionFailed | 如果存储的数据解密失败则返回。 |
|  | | PerErrc:: kResourceBusy | 如果文件已经打开，或者当前正在执行同名的DeleteFile、RecoverFile或ResetFile，则返回。 |
|  | | PerErrc:: kOutOfStorageSpace | 创建文件时，如果可用存储空间不足或文件数量将大于配置的maxnumberfiles，则返回。 |
|  | | PerErrc:: kNotInitialized | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | | PerErrc:: kInvalidOpenMode | 如果传递的模式包含无效的模式组合，则返回。 |
| **Header file:** | | #include "ara/per/file\_storage.h" | |
| **Description:** | | 用用户提供的buffer读或写文件存储数据库中的文件。  如果没有提供打开模式，默认是在文件的开始。（类似kAtTheBeginning）  提供的缓冲区将被用于ReadWriteAccessor实现基于块读写加快了对文件多次小的访问。  如果文件不存在则创建。  文件UniqueHandle超出范围时文件将会被关闭。 | |
| **Additional:** | | **P3** | |

#### [SWRD\_API\_Per\_00052] OpenFileReadOnly

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00052 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00376] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 修改 新增错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | ara::per::FileStorage::OpenFileReadOnly(ara::core::StringView fileName) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<UniqueHandle<ReadAccessor> > OpenFileReadOnly  (ara::core::StringView fileName) noexcept; | | |
| **Parameters (in):** | fileName | 文件名。可能对应配置文件中的Per File.fileName | |
| **Return value:** | ara::core::Result< UniqueHandle< ReadAccessor > > | 一个结果，包含一个文件的UniqueHandle，或为PerErrc中的持久性定义的一个错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果文件已经打开，或者当前正在执行同名的DeleteFile、RecoverFile或ResetFile，则返回 |
|  | PerErrc:: kFileNotFound | | 如果提供的文件不存在于文件存储中，则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 在文件存储中打开指定文件只读。  打开文件时，搜索位置设置为开头（类似kAtTheBeginning）。  文件UniqueHandle超出范围时文件将会被关闭。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor =fs->OpenFileReadOnly(kFileName);  if (!accessor) {  logger.LogError()<<"以读写方式打开文件失败，错误信息为："<<accessor.Error().Message();  return;  } | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00053 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00114] | | |
| **CR** |  | | |
| **Consistency** | No | | |
| **Change Type** | 修改 参数变更，增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | ara::per::FileStorage::OpenFileReadOnly(ara::core::StringView fileName, OpenMode mode) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<UniqueHandle<ReadAccessor>>OpenFileReadOnly (ara::core::StringView fileName,OpenMode mode) noexcept; | | |
| **Parameters (in):** | fileName | 文件名。可能对应配置文件中的Per File.fileName | |
| mode | 文件打开的模式 | |
| **Return value:** | ara::core::Result< UniqueHandle< ReadAccessor > > | 一个结果，包含一个文件的UniqueHandle，或为PerErrc中的持久性定义的一个错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果文件已经打开，或者当前正在执行同名的DeleteFile、RecoverFile或ResetFile，则返回。 |
|  | PerErrc:: kFileNotFound | | 如果提供的文件不存在于文件存储中，则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc::kInvalidOpenMode | | 如果传递的模式包含无效的模式组合，则返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 在文件存储中打开指定文件只读，定义成只读模式。  如果文件打开模式没说明，默认打开后搜索到文件开头位置（类似kAtTheBeginning）。  文件UniqueHandle超出范围时文件将会被关闭。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor =fs->OpenFileReadOnly(kFileName，BasicOperations::OpenMode(0));  if (!accessor) {  logger.LogError()<<"以读方式打开文件失败，错误信息为："<<accessor.Error().Message();  return;  } | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00054 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00430] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 新增 | | |
| **Kind:** | function | | |
| **Symbol:** | ara::per::FileStorage::OpenFileReadOnly(ara::core::StringView fileName, OpenMode mode，ara::core::Span< ara::core::Byte > buffer) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<UniqueHandle<ReadAccessor>>OpenFileReadOnly (ara::core::StringView fileName,OpenMode mode, ara::core::Span< ara::core::Byte > buffer) noexcept; | | |
| **Parameters (in):** | fileName | 文件名。可能对应配置文件中的PersistencyFile.fileName | |
| mode | 文件打开的模式 | |
|  | Buffer | 分块译码读用的内存 | |
| **Return value:** | ara::core::Result< UniqueHandle< ReadAccessor > > | 一个结果，包含一个文件的UniqueHandle，或为PerErrc中的持久性定义的一个错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果文件已经打开，或者当前正在执行同名的DeleteFile、RecoverFile或ResetFile，则返回。 |
|  | PerErrc:: kFileNotFound | | 如果提供的文件不存在于文件存储中，则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc::kInvalidOpenMode | | 如果传递的模式包含无效的模式组合，则返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 在文件存储中打开指定文件只读，用用户提供的buffer。  如果文件打开模式没说明，默认打开后搜索到文件开头位置（类似kAtTheBeginning）。  提供的缓冲区将被用于ReadWriteAccessor实现基于块的读加快了对文件多次小的访问  文件UniqueHandle超出范围时文件将会被关闭。 | | |
| **Additional:** | **P2** | | |

#### [SWRD\_API\_Per\_00055] OpenFileWriteOnly

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00055 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00377] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 修改 增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | ara::per::FileStorage::OpenFileWriteOnly(ara::core::StringView fileName) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<UniqueHandle<ReadWriteAccessor> > OpenFileWriteOnly  (ara::core::StringView fileName) noexcept; | | |
| **Parameters (in):** | fileName | 文件名。可能对应配置文件中的Per File.fileName | |
| **Return value:** | ara::core::Result< UniqueHandle< ReadWriteAccessor > > | 一个结果，包含一个文件的UniqueHandle，或为PerErrc中的持久性定义的一个错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc:: kIllegalWriteAccess | | 如果文件存储配置为只读则返回 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果文件已经打开，或者当前正在执行同名的DeleteFile、RecoverFile或ResetFile，则返回。 |
|  | PerErrc:: kOutOfStorageSpace | | 创建文件时，如果可用存储空间不足或文件数量将大于配置的maxnumberfiles，则返回 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 在文件存储中打开一个文件指定打开方式是只写。  文件是被截取的，截去顶部的，（类似kTruncate）  文件不存在则创建。文件UniqueHandle超出范围时文件将会被关闭。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs->OpenFileWriteOnly(kFileName);  if (!accessor) {  logger.LogError()<<"以只写方式打开文件失败，错误信息为："<<accessor.Error().Message();  return;  } | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00056 | | |
| **Upstream ID** | [SWS\_PER\_00115] | | |
| **Consistency** | No | | |
| **Change Type** | 修改 参数变更，增加错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | ara::per::FileStorage::OpenFileWriteOnly(ara::core::StringView fileName, OpenMode mode) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<UniqueHandle<ReadWriteAccessor>>OpenFileWriteOnly (ara::core::StringView fileName,OpenMode mode) noexcept; | | |
| **Parameters (in):** | fileName | 文件名。可能对应配置文件中的Per File.fileName | |
| mode | 文件打开的模式 | |
| **Return value:** | ara::core::Result< UniqueHandle< ReadWriteAccessor > > | 一个结果，包含一个文件的UniqueHandle，或为PerErrc中的持久性定义的一个错误。 | |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc:: kIllegalWriteAccess | | 如果文件存储配置为只读则返回 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果文件已经打开，或者当前正在执行同名的DeleteFile、RecoverFile或ResetFile，则返回。 |
|  | PerErrc:: kOutOfStorageSpace | | 创建文件时，如果可用存储空间不足或文件数量将大于配置的maxnumberfiles，则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc::kInvalidOpenMode | | 如果传递的模式包含无效的模式组合，则返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 在文件存储中打开一个文件，根据定义的打开模式打开。  如果没有提供打开模式，文件是被截取的，截去顶部的。（类似kTruncate）  文件不存在则创建。文件UniqueHandle超出范围时文件将会被关闭。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs->OpenFileWriteOnly(fileName, BasicOperations::OpenMode(0));  if (!accessor) {  logger.LogError()<<"以只写方式打开文件失败，错误信息为："<<accessor.Error().Message();  return;  } | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00057 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00431] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 新增 | | |
| **Kind:** | function | | |
| **Symbol:** | OpenFileWriteOnly(ara::core::StringView fileName, OpenMode mode, ara::core::Span< ara::core::Byte > buffer) | | |
| **Scope:** | class ara::per::FileStorage | | |
| **Syntax:** | ara::core::Result<UniqueHandle<ReadWriteAccessor>>OpenFileWriteOnly (ara::core::StringView fileName,OpenMode mode，ara::core::Span< ara::core::Byte > buffer) noexcept; | | |
| **Parameters (in):** | fileName | 文件名。可能对应配置文件中的Per File.fileName | |
| mode | 文件打开的模式 | |
|  | Buffer | 分块译码写用的内存 | |
| **Return value:** | ara::core::Result< UniqueHandle< ReadWriteAccessor > > | 一个结果，包含一个文件的UniqueHandle，或为PerErrc中的持久性定义的一个错误。 | |
| **Exception Safety:** | Noexcept | | |
| **Thread Safety:** | re-entrant | | |
|  | PerErrc:: kIllegalWriteAccess | | 如果文件存储配置为只读则返回。 |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kIntegrityCorrupted | | 如果存储的数据因结构完整性损坏而无法读取，则返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kResourceBusy | | 如果文件已经打开，或者当前正在执行同名的DeleteFile、RecoverFile或ResetFile，则返回。 |
|  | PerErrc:: kOutOfStorageSpace | | 创建文件时，如果可用存储空间不足或文件数量将大于配置的maxnumberfiles，则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc::kInvalidOpenMode | | 如果传递的模式包含无效的模式组合，则返回。 |
| **Header file:** | #include "ara/per/file\_storage.h" | | |
| **Description:** | 在文件存储中打开指定文件供写入根据用户提供的buffer。  如果没有提供打开模式，文件是被截取的，截去顶部（类似kTruncate）  提供的buffer将会用ReadWriteAccessor实现分块写，提高了速度多次对文件小块的访问。  文件不存在则创建。文件UniqueHandle超出范围时文件将会被关闭。 | | |
| **Additional:** | **P3** | | |

### [SWRD\_API\_Per\_00058] Origin枚举

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00058 | |
|  |  | |
| **Upstream ID** | [SWS\_PER\_00146] | |
| Consistency | Yes | |
| **Change Type** | 修改 修改scope，修改类型，修枚举值，修改头文件 Specification of origin used in MovePosition. | |
| **Kind:** | enumeration |  |
| **Symbol:** | Origin |  |
| **Scope:** | namespace ara::per |  |
| **Underlying type:** | uint32\_t |  |
| **Syntax** | enum class Origin : uint32\_t {...}; |  |
| **Values:** | kBeginning= 0 | 从开始的位置seek |
| kCurrent= 1 | 从当前的位置seek |
| kEnd= 2 | 从结束的位置seek |
| **Header file:** | #include ara/per/read\_accessor.h" |  |
| **Description:** | seek方向说明 |  |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor =fs->OpenFileReadOnly(kFileName);  if (!accessor) {  logger.LogError()<<"以读写方式打开文件失败，错误信息为："<<accessor.Error().Message();  return;  }  auto file = std::move(accessor).Value();  file->seek(0, ::ara::per::BasicOperations::SeekDirection::kEnd); | |

### [SWRD\_API\_Per\_00059] ReadAccessor

这章描述的方法有效的从ara::per::ReadAccessor对象调用OpenFileReadOnly，ara::per::ReadWriteAccessor对象调用OpenFileWriteOnly或者 OpenFileReadWrite。

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00059 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00342] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 修改 修改基类 |
| **Kind:** | class |
| **Symbol:** | ara::per::ReadAccessor |
| **Scope:** | namespace ara::per |
| **Syntax:** | class ReadAccessor {...}; |
| **Header file:** | #include "ara/per/read\_accessor.h" |
| **Description:** | 用于读取文件数据的类  提供了二进制和文本的模式方法对检查和获取当前位/字符(PeekByte/PeekChar, GetByte/GetChar)的方法来读取二进制/文本文件(ReadBinary/ReadText)。提供方法按行读取（ReadLine），提供检查和设置文件的当前位置(GetPosition, SetPosition, MovePosition, IsEof)，检查当前文件的size（GetSize）。 |
| **Additional:** | 代码示例及类句柄的获取说明：  见FileStorage class相应的示例获取的proxy句柄。  auto accessor = proxy->OpenFileReadOnly(StringView("filename"));  if (!accessor) {  //以只读方式打开文件失败，信息为:accessor.Error().Message();  //相应处理  }  auto ac = std::move(accessor).Value();  accessor 是ReadAccessor类的对象。 |

#### [SWRD\_API\_Per\_00060] ReadAccessor

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00060 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00413] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 新增 | |
| **Kind:** | Function | |
| **Symbol:** | ReadAccessor(ReadAccessor &&ra) | |
| **Scope:** | class ara::per::ReadAccessor | |
| **Syntax:** | ReadAccessor (ReadAccessor &&ra) noexcept; | |
| **Parameters (in):** | Ra | 要移动的ReadAccessor对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/read\_accessor.h" | |
| **Description:** | ReadAccessor的移动构造函数。 | |
| **Additional:** | **P1** | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00061 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00415] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 新增 |
| **Kind:** | Function |
| **Symbol:** | ReadAccessor(const ReadAccessor &) |
| **Scope:** | class ara::per::ReadAccessor |
| **Syntax:** | ReadAccessor (const ReadAccessor &) )=delete; |
| **Header file:** | #include "ara/per/read\_accessor.h" |
| **Description:** | ReadAccessor的拷贝构造函数，不能被使用。 |
| **Additional:** | **P1** |

#### [SWRD\_API\_Per\_00062] operator=

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00062 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00414] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 新增 | |
| **Kind:** | Function | |
| **Symbol:** | operator=(ReadAccessor &&ra) | |
| **Scope:** | class ara::per::ReadAccessor | |
| **Syntax:** | ReadAccessor& operator= (ReadAccessor &&ra) &noexcept; | |
| **Parameters (in):** | Ra | 要移动的ReadAccessor对象 |
| **Return value:** | ReadAccessor & | 移动后的ReadAccessor对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/read\_accessor.h" | |
| **Description:** | 移动ReadAccessor运算符 | |
| **Additional:** | **P1** | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00063 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00416] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 新增 |
| **Kind:** | Function |
| **Symbol:** | operator=(const ReadAccessor &) |
| **Scope:** | class ara::per::ReadAccessor |
| **Syntax:** | ReadAccessor& operator= (const ReadAccessor &)=delete |
| **Header file:** | #include "ara/per/read\_accessor.h" |
| **Description:** | ReadAccessor的拷贝运算符，不应被使用 |
| **Additional:** | **P1** |

#### [SWRD\_API\_Per\_00064] ~ReadAccessor

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00064 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00417] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 新增 |
| **Kind:** | Function |
| **Symbol:** | ~ReadAccessor() |
| **Scope:** | class ara::per::ReadAccessor |
| **Syntax:** | ~ReadAccessor () noexcept; |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | No |
| **Header file:** | #include "ara/per/read\_accessor.h" |
| **Description:** | ReadAccessor的析构函数。 |
| **Additional:** | **P1** |

#### [SWRD\_API\_Per\_00065] PeekChar

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00066 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00167] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 修改  方法名变更，新增错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | PeekChar() | | |
| **Scope:** | class ara::per::ReadAccessor | | |
| **Syntax:** | ara::core::Result<char> PeekChar () const noexcept; | | |
| **Return value:** | ra::core::Result< char > | 当前位置的字符，或为PerErrc中的持久性定义的一个错误。或为供应商定义的错误 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | no | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kIsEof | | 如果当前位置位于文件的末尾或文件为空，则返回 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | |
| **Description:** | 返回文件中当前位置的字符，当前位置不能被改变 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ara::core::String getData;  ara::per::int\_type c;  while (ac-> PeekChar () != -1) {  c = ac->get();  getData = getData + (char)c;  }  **P1** | | |

#### [SWRD\_API\_Per\_00066] PeekByte

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00066 | | | |
| **Type** | | Valid | | |
| **Priority** | | M | | |
| **Upstream ID** | [SWS\_PER\_00418] | | | |
| **CR** |  | | | |
| **Consistency** | Yes | | | |
| **Change Type** | 新增 | | | |
| **Kind:** | function | | | |
| **Symbol:** | PeekByte() | | | |
| **Scope:** | class ara::per::ReadAccessor | | | |
| **Syntax:** | ara::core::Result<ara::core::Byte> PeekByte () const noexcept; | | | |
| **Return value:** | ara::core::Result<ara::core::Byte> | | 一个Byte位，或为PerErrc中的持久性定义的一个错误。或为供应商定义的错误 | |
| **Exception Safety:** | noexcept | | | |
| **Thread Safety:** | no | | | |
|  | PerErrc::kPhysicalStorageFailure | | | 如果访问存储失败，返回。 |
|  | PerErrc:: kValidationFailed | | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kIsEof | | | 如果当前位置位于文件的末尾或文件为空，则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | | |
| **Description:** | 返回文件中当前位置的字节，当前位置不能被改变 | | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ara::core::String getData;  ara::per::int\_type c;  while (ac-> PeekByte () != -1) {//返回的当前位置的字节  }  **P2** | | | |

#### [SWRD\_API\_Per\_00067] GetChar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00067 | | | |
| **Type** | | Valid | | |
| **Priority** | | H | | |
| **Upstream ID** | [SWS\_PER\_00168] | | | |
| **CR** |  | | | |
| **Consistency** | Yes | | | |
| **Change Type** | 修改 修改接口名，新增错误返回类型 | | | |
| **Kind:** | function | | | |
| **Symbol:** | GetChar () | | | |
| **Scope:** | class ara::per::ReadAccessor | | | |
| **Syntax:** | ara::core::Result<char > GetChar ()noexcept; | | | |
| **Return value:** | ara::core::Result< char > | | 一个char，或为PerErrc中的持久性定义的一个错误。或为供应商定义的错误 | |
| **Exception Safety:** | noexcept | | | |
| **Thread Safety:** | no | | | |
|  | PerErrc::kPhysicalStorageFailure | | | 如果访问存储失败，返回。 |
|  | PerErrc:: kValidationFailed | | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kIsEof | | | 如果当前位置位于文件的末尾或文件为空，则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | | |
| **Description:** | 返回文件中当前位置的字符，推进当前位置。在一个错误的情况下,当前位置不能更改。 | | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ara::core::String getData;  ara::per::int\_type c;  while (ac-> PeekChar () != -1) {  c = ac-> GetChar ();  getData = getData + (char)c;  }  **P1** | | | |

#### [SWRD\_API\_Per\_00068] GetByte

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00068 | | | |
| **Type** | | Valid | | |
| **Priority** | | H | | |
| **Upstream ID** | [SWS\_PER\_00419] | | | |
| **CR** |  | | | |
| **Consistency** | Yes | | | |
| **Change Type** | 新增 | | | |
| **Kind:** | function | | | |
| **Symbol:** | GetByte () | | | |
| **Scope:** | class ara::per::ReadAccessor | | | |
| **Syntax:** | ara::core::Result<ara::core::Byte> GetByte () noexcept; | | | |
| **Return value:** | ara::core::Result< ara::core::Byte > | | 一个byte，或为PerErrc中的持久性定义的一个错误。或为供应商定义的错误 | |
| **Exception Safety:** | noexcept | | | |
| **Thread Safety:** | no | | | |
|  | PerErrc::kPhysicalStorageFailure | | | 如果访问存储失败，返回。 |
|  | PerErrc:: kValidationFailed | | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kIsEof | | | 如果当前位置位于文件的末尾或文件为空，则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | | |
| **Description:** | 返回文件中当前位置的字节，推进当前位置。在一个错误的情况下,当前位置不能更改。 | | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ara::core::String getData;  ara::per::int\_type c;  while (ac-> PeekByte () != -1) {  c = ac-> GetByte ();  }  **P2** | | | |

#### [SWRD\_API\_Per\_00069] ReadText

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00069 | | | |
| **Type** | | Valid | | |
| **Priority** | | H | | |
| **Upstream ID** | [SWS\_PER\_00420] | | | |
| **CR** |  | | | |
| Consistency | Yes | | | |
| **Change Type** | 新增 | | | |
| **Kind:** | function | | | |
| **Symbol:** | ReadText () | | | |
| **Scope:** | class ara::per::ReadAccessor | | | |
| **Syntax:** | ara::core::Result<ara::core::String> ReadText () noexcept; | | | |
| **Return value:** | ara::core::Result< ara::core::String > | | 一个字符串，或为PerErrc中的持久性定义的一个错误。或为供应商定义的错误 | |
| **Exception Safety:** | noexcept | | | |
| **Thread Safety:** | no | | | |
|  | PerErrc::kPhysicalStorageFailure | | | 如果访问存储失败，返回。 |
|  | PerErrc:: kValidationFailed | | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kIsEof | | | 如果当前位置位于文件的末尾或文件为空，则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | | |
| **Description:** | 返回文件中从当前位置开始读取剩下的所有字符到字符串中。当前执行位置到文件最后。如果有错误，当前位置不改变。 | | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ara::core::String getData;  ara::per::int\_type c;  while (ac-> PeekChar () != -1) {  c = ac-> ReadText ();  }  **P1** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00070 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00165] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 修改 修改接口名，修改参数，返回值。新增错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | ReadText(uint64\_t n) | | |
| **Scope:** | class ara::per::ReadAccessor | | |
| **Syntax:** | ara::core::Result<ara::core::String> ReadText (uint64\_t n) noexcept; | | |
| **Parameters (out):** | n | 读取的字符数。 | |
| **Return value:** | ara::core::Result< ara::core::String > | 读取的字符串。或为PerErrc中的持久性定义的一个错误。或为供应商定义的错误 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | no | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kIsEof | | 如果当前位置位于文件的末尾或文件为空，则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | |
| **Description:** | 从当前位置读取字符的数量。当前的位置相应的增加。如果到达了文件末尾，获取的数量可能比请求的数量少，从当前位置到文件尾。如果发生错误，当前位置不改变。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  char ch[8] = "";  readLen = ac->read(ara::core::Span<char>(ch, 7));  readLen是读出的长度，ch是读出的内容。  **P1** | | |

#### [SWRD\_API\_Per\_00071] ReadBinary

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00071 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00421] | | |
| **CR** |  | | |
| Consistency | Yes | | |
| **Change Type** | 新增 | | |
| **Kind:** | function | | |
| **Symbol:** | ReadBinary () | | |
| **Scope:** | class ara::per::ReadAccessor | | |
| **Syntax:** | ara::core::Result<ara::core:: Vector < ara::core::Byte >> ReadBinary () noexcept; | | |
| **Return value:** | ara::core::Result<ara::core:: Vector < ara::core::Byte >> | 读取Byte的vector。或为PerErrc中的持久性定义的一个错误。或为供应商定义的错误 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | no | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kIsEof | | 如果当前位置位于文件的末尾或文件为空，则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | |
| **Description:** | 从当前位置读取Byte到vector中。从当前位置到文件尾。  如果发生错误，当前位置不改变。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ara::core:: Vector < ara::core::Byte >> by;  readLen = ac-> ReadBinary(by);  readLen是读出的长度，by是读出的内容的vector。  **P3** | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00072 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00422] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 新增 | | |
| **Kind:** | function | | |
| **Symbol:** | ReadBinary (uint64\_t n) | | |
| **Scope:** | class ara::per::ReadAccessor | | |
| **Syntax:** | ara::core::Result<ara::core::Vector<ara::core::Byte>>ReadBinary(uint64\_t n)  noexcept; | | |
| **Parameters (out):** | n | 读取的bytes的数量。 | |
| **Return value:** | ara::core::Result<ara::core:: Vector < ara::core::Byte >> | 读取Byte的vector。或为PerErrc中的持久性定义的一个错误。或为供应商定义的错误 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | no | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kIsEof | | 如果当前位置位于文件的末尾或文件为空，则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | |
| **Description:** | 从当前位置开始，将一定数量的字节读入字节向量。当前的位置相应的增加。如果到达了文件末尾，获取的数量可能比请求的数量少，从当前位置到文件尾。如果发生错误，当前位置不改变。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ara::core:: Vector < ara::core::Byte >> by;  readLen = ac-> ReadBinary (20);  readLen是读出的长度，by是读出的内容的vector。  **P3** | | |

#### [SWRD\_API\_Per\_00073] ReadLine

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00073 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00119] | | |
| **CR** |  | | |
| Consistency | Yes | | |
| **Change Type** | 修改 方法名变更，返回值变更，新增错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | ReadLine(char delimiter=’\n’) | | |
| **Scope:** | class ara::per::ReadAccessor | | |
| **Syntax:** | ara::core::Result<ara::core::String> ReadLine (char delimiter=’\n’) noexcept; | | |
| **Parameters (in):** | delimiter | 用于分隔的字符 | |
| **Return value:** | ara::core::Result< ara::core::String > | 读取的字符串，或为PerErrc中的持久性定义的一个错误。或为供应商定义的错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | no | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc:: kValidationFailed | | 如果不能确保存储数据的有效性，则返回。 |
|  | PerErrc:: kEncryptionFailed | | 如果存储的数据解密失败则返回。 |
|  | PerErrc:: kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc:: kIsEof | | 如果当前位置位于文件的末尾或文件为空，则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | |
| **Description:** | 将整行字符读入字符串，推进当前位置。没有参数的话用分隔符"\\n" (ASCII 0x10)分隔每行的结尾。分隔符本身不包含在返回的字符串中。如果到达文件的末尾，则返回剩余的字符，并将当前位置设置为文件的末尾。在出现错误的情况下，当前位置不会改变。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ara::core::Array<char, 1024> buf;  readLen = ac-> ReadLine (buf, '\n');  readLen是读出的长度，buf是读出的内容的。  **P1** | | |

#### [SWRD\_API\_Per\_00074] GetSize

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00074 | |
| **Type** | Valid | |
| **Priority** | M | |
| **Upstream ID** | [SWS\_PER\_00424] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 新增 | |
| **Kind:** | function | |
| **Symbol:** | GetSize | |
| **Scope:** | class ara::per::ReadAccessor | |
| **Syntax:** | uint64\_t GetSize () const noexcept; | |
| **Return value:** | uint64\_t | 文件的当前大小(以字节为单位). |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | no | |
| **Header file:** | #include "ara/per/read\_accessor.h" | |
| **Description:** | 当前文件的大小(以字节为单位). | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs -> OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  readLen = ac-> GetSize();  readLen是当前文件的大小。  **P1** | |

#### [SWRD\_API\_Per\_00075] GetPosition

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00075 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00162] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 修改 修改方法名，返回值类型 | |
| **Kind:** | function |  |
| **Symbol:** | GetPosition() |  |
| **Scope:** | class ara::per:: ReadAccessor |  |
| **Syntax:** | uint64\_t GetPosition () const noexcept; | |
| **Return value:** | uint64\_t | 文件中的当前位置(以字节为单位)，从文件开始算起。 |
| **Exception Safety:** | noexcept |  |
| **Thread Safety:** | no | |
| **Header file:** | #include "ara/per/basic\_operations.h" | |
| **Description:** | 返回相对于文件开头的当前位置。返回的位置可能在文件的末尾。 | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs ->OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  len = ac->GetPosition();  len是文件中的当前位置。  **P1** | |

#### [SWRD\_API\_Per\_00076] SetPosition

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00076 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00163] | | |
| **CR** |  | | |
| **Consistency** | No | | |
| **Change Type** | 修改 修改接口名，返回值变更，新增错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | SetPosition(uint64\_t position) | | |
| **Scope:** | class ara::per:: ReadAccessor | | |
| **Syntax:** | ara::core::Result<void> SetPosition (uint64\_t position) noexcept; | | |
| **Parameters (in):** | position | 距离文件开始的位置，以字节为单位。 | |
| **Return value:** | ara::core::Result< void > | 返回空，在出现错误的情况下，它包含下面定义的任何错误，或者特定于供应商的错误 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | no | | |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc::kInvalidPosition | | 如果给定的位置超出了文件的末尾，则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | |
| **Description:** | 设置相对于文件开头的当前位置，如果发生错误，文件位置不变。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs ->OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ac-> SetPosition(0);  将位置设置到开始位置  **P1** | | |

#### [SWRD\_API\_Per\_00077] MovePosition

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00077 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00164] | | |
| **CR** |  | | |
| **Consistency** | No | | |
| **Change Type** | 修改 修改接口名，返回值类型 | | |
| **Kind:** | function | | |
| **Symbol:** | MovePosition(Origin origin, int64\_t offset) | | |
| **Scope:** | class ara::per::ReadAccessor | | |
| **Syntax:** | ara::core::Result<uint64\_t> MovePosition (Origin origin, int64\_t offset) noexcept; | | |
| **Parameters (in):** | origin | 移动' offset '字节的起始点 | |
| offset | 相对于' origin '的字节偏移量。kBeginning和kCurrent为正，kCurrent和kEnd为负。对于kCurrent, 0的偏移量不会改变当前位置。在kEnd的情况下，0的偏移量将把位置设置为文件的末尾。 | |
| **Return value:** | ara::core::Result<uint64\_t> | 包含从文件开头开始的新位置(以字节为单位)的结果。在出现错误的情况下，它包含下面定义的任何错误，或者特定于供应商的错误。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | no | | |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
|  | PerErrc::kInvalidPosition | | 如果结果位置小于零或超过文件末尾则返回。 |
| **Header file:** | #include "ara/per/read\_accessor.h" | | |
| **Description:** | 将文件中的当前位置相对于原点移动，如果发生错误，文件位置不变。 | | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs ->OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  int64\_t offset beg = 0;  ac-> MovePosition (ara::per::BasicOperations::SeekDirection::kBeg, beg);  将位置设置到开始位置  **P1** | | |

#### [SWRD\_API\_Per\_00078] IsEof

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00078 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00107] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 修改 修改接口名，头文件变更 | |
| **Kind:** | function | |
| **Symbol:** | IsEof() | |
| **Scope:** | class ara::per::ReadAccessor | |
| **Syntax:** | bool IsEof () const noexcept | |
| **Return value:** | bool | 如果到达文件末尾，则为True，否则为false。 |
| **Exception Safety:** | Noexcept | |
| **Thread Safety:** | no | |
| **Header file:** | #include "ara/per/read\_accessor.h" | |
| **Description:** | 检查是否到达文件末尾 | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs ->OpenFileReadOnly (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  bool bl = ac-> IsEof()  bl返回是否到达文件末尾。  **P1** | |

### [SWRD\_API\_Per\_00079] ReadWriteAccessor 类

本节展示了ara::per::ReadWriteAccessor类的对象可用的方法。

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00079 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00343] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 不变 |
| **Kind:** | class |
| **Symbol:** | ara::per::ReadWriteAccessor |
| **Scope:** | namespace ara::per |
| **Base class:** | ReadAccessor |
| **Syntax:** | class ReadWriteAccessor : public ReadAccessor {...}; |
| **Header file:** | #include "ara/per/read\_write\_accessor.h" |
| **Description:** | ReadWriteAccessor用于读写文件数据。  它提供了WriteBinary和WriteText方法，为受控的、非格式化的写入提供结果，为简单格式化的写入提供操作符<<方法。它还提供SyncToFile()来将操作系统的缓冲区刷新到存储。 |
| **Additional:** | 参见ReadAccessor class的此项说明，  proxy->OpenFileReadOnly替换成  proxy->OpenFileReadWrite或  proxy-> OpenFileWriteOnly  即可得到此类的指针。 |

#### [SWRD\_API\_Per\_00080] SyncToFile

|  |  |  |  |
| --- | --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00080 | | |
| **Type** | Valid | | |
| **Priority** | H | | |
| **Upstream ID** | [SWS\_PER\_00122] | | |
| **CR** |  | | |
| **Consistency** | Yes | | |
| **Change Type** | 修改接口名，新增错误返回类型 | | |
| **Kind:** | function | | |
| **Symbol:** | SyncToFile() | | |
| **Scope:** | class ara::per::ReadWriteAccessor | | |
| **Syntax:** | ara::core::Result<void> SyncToFile () noexcept; | | |
| **Return value:** | ara::core::Result< void > | 结果，要么为空，要么包含为PerErrc中定义的错误之一。 | |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | no | | |
|  | PerErrc::kPhysicalStorageFailure | | 如果访问存储失败，返回。 |
|  | PerErrc::kEncryptionFailed | | 如果存储的数据加密失败则返回。 |
|  | PerErrc::kOutOfStorageSpace | | 当可用存储空间不足更新后的文件大小时返回。 |
|  | PerErrc::kNotInitialized | | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/read\_write\_accessor.h" | | |
| **Description:** | 刷新并强制写缓存区到文件的持久存储中 | | |
| **Additional:** | 先调用flush并强制写缓存区到文件的持久存储中  auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs ->OpenFileReadWrite (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  char\* writeChars = (char \*)"FsTest2 write\n";  ara::per::pos\_type writeLen = ac-> WriteText (ara::core::Span<char>(writeChars, strlen(writeChars)));  ac->flush();  ac-> SyncToFile();  **P1** | | |

#### [SWRD\_API\_Per\_00081] SetFileSize

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00081 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00428] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 新增 |

|  |  |  |
| --- | --- | --- |
| **Kind:** | function | |
| **Symbol:** | SetFileSize(uint64\_t size) | |
| **Scope:** | class ara::per::ReadWriteAccessor | |
| **Syntax:** | ara::core::Result<void> SetFileSize (uint64\_t size) noexcept; | |
| **Parameters (in):** | size | 文件的新的大小 |
| **Return value:** | ara::core::Result< void > | 结果，要么为空，要么包含为PerErrc中定义的错误之一。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | no | |
| **Errors:** | PerErrc::kPhysicalStorageFailure | 如果访问存储失败，返回。 |
| PerErrc::kEncryptionFailed | 如果存储的数据加密或解密失败则返回。. |
| PerErrc::kNotInitialized | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| PerErrc::kInvalidSize | 如果新大小大于当前大小则返回。 |
| **Header file:** | #include "ara/per/read\_write\_accessor.h" | |
| **Description:** | 将文件大小减小到' size '，有效地删除文件中超出该大小的当前内容。  如果小于' size '则当前文件位置不变，否则设置为文件中的最后一个有效位置。如果' size '为0，则当前文件位置也将设置为0。 | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs ->OpenFileReadWrite (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ac-> SetFileSize(20)  **P2** | |

#### [SWRD\_API\_Per\_00082] WriteText

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00082 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00166] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 修改 修改接口名，修改参数，返回值，新增错误返回类型 | |
| **Kind:** | function | |
| **Symbol:** | WriteText(ara::core::StringView s) | |
| **Scope:** | class ara::per::ReadWriteAccessor | |
| **Syntax:** | ara::core::Result<void> WriteText (ara::core::StringView s) noexcept; | |
| **Parameters (in):** | s | 要写入的字符。 |
| **Return value:** | ara::core::Result< void > | 结果，要么为空，要么包含为PerErrc中定义的错误之一。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | no | |
| **Errors:** | PerErrc::kPhysicalStorageFailure | 如果访问存储失败，返回。 |
| PerErrc::kEncryptionFailed | 如果存储的数据加密或解密失败则返回。 |
| PerErrc::kOutOfStorageSpace | 当可用存储空间不足更新后的文件大小时返回。 |
| PerErrc::kNotInitialized | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/read\_write\_accessor.h" | |
| **Description:** | 将StringView的内容写入文件。  存储持久化数据的时间取决于持久化的实现。SyncTo文件可以用来促进持久化处理文件内容。  在出现错误的情况下，文件内容可能已损坏，当前位置可能已更改，也可能未更改。  对于每个支持的错误，文件的预期状态可以如下所示：  kPhysicalStorageFailure: 该文件的状态未知。它可能已经被完全摧毁了。  kEncryptionFailed：文件的内容和当前位置将被更新，但不能持久。持久化的文件将反映该文件的旧版本。  kOutOfStorage Space：文件的内容将被更新，但是超出配额的操作部分将被丢弃。当前位置将位于文件的末尾。  kNotInitialized: 文件的内容和当前位置没有改变。 | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs ->OpenFileReadWrite (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite, Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  char\* writeChars = (char \*)"FsTest2 write\n";  ara::per::pos\_type writeLen = ac-> WriteText(ara::core::Span<char>(writeChars, strlen(writeChars)));  ac->flush();  ac-> SyncToFile();  **P1** | |

#### [SWRD\_API\_Per\_00083] WriteBinary

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00083 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00423] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 新增 | |
| **Kind:** | function | |
| **Symbol:** | WriteBinary(ara::core::Span< const ara::core::Byte > b) | |
| **Scope:** | class ara::per::ReadWriteAccessor | |
| **Syntax:** | ara::core::Result<void>WriteBinary(ara::core::Span<const ara::core::Byte> b) noexcept; | |
| **Parameters (in):** | b | 包含要写入的字节的一段字节。 |
| **Return value:** | ara::core::Result< void > | 结果，要么为空，要么包含为PerErrc中定义的错误之一。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | no | |
| **Errors:** | PerErrc::kPhysicalStorageFailure | 如果访问存储失败，返回。. |
| PerErrc::kEncryptionFailed | 如果存储的数据加密或解密失败则返回 |
| PerErrc::kOutOfStorageSpace | 当可用存储空间不足更新后的文件大小时返回. |
| PerErrc::kNotInitialized | 如果在ara::core::Deinitialize后调用此方法将返回。 |
| **Header file:** | #include "ara/per/read\_write\_accessor.h" | |
| **Description:** | 将一个字节范围的内容写入文件  存储持久化数据的时间取决于持久化的实现。SyncTo文件可以用来促进持久化处理文件内容。  在出现错误的情况下，文件内容可能已损坏，当前位置可能已更改，也可能未更改。  对于每个支持的错误，文件的预期状态可以如下所示：  kPhysicalStorageFailure: 该文件的状态未知。它可能已经被完全摧毁了。  kEncryptionFailed：文件的内容和当前位置将被更新，但不能持久。持久化的文件将反映该文件的旧版本。  kOutOfStorage Space：文件的内容将被更新，但是超出配额的操作部分将被丢弃。当前位置将位于文件的末尾。  kNotInitialized: 文件的内容和当前位置没有改变。 | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs ->OpenFileReadWrite (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite,Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  ara::core::Result<void>WriteBinary(ara::core::Span<const ara::core::Byte> b) noexcept;  **P3** | |

#### [SWRD\_API\_Per\_00084] operator<<

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00084 | |
| **Upstream ID** | [SWS\_PER\_00125] | |
| **Consistency** | No | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | operator<<(ara::core::StringView s) | |
| **Scope:** | class ara::per::ReadWriteAccessor | |
| **Syntax:** | ReadWriteAccessor& operator<< (ara::core::StringView s) noexcept; | |
| **Parameters (in):** | s | 要写的字符串 |
| **Return value:** | ReadWriteAccessor & | ReadWriteAccessor对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | no | |
| **Header file:** | #include "ara/per/read\_write\_accessor.h" | |
| **Description:** | 将字符串写入文件  这个操作只是一个舒适的功能，非安全的关键应用。如果在此操作过程中出现错误，则静默忽略该错误 | |
| **Additional:** | auto result = OpenFileStorage(StringView("pathname"));  if (!result) {  //打开文件系统失败，错误信息;  //相应的处理  }  auto fs= std::move(result).Value();  constexpr auto kFileName = "hello.txt";  auto accessor = fs ->OpenFileReadWrite (kFileName);  if (!accessor) {  logger.LogError()<<"Failed to OpenFileReadWrite,Error:"<<accessor.Error().Message();  return;  }  auto ac = std::move(accessor).Value();  \*ac<<ara::core::String("testoperator<<(ara::core::StringView s)"); | |

## Update and Remove of Persistency Data接口定义

持久化集群允许更新、回退和删除所有安装的键值存储和文件存储。应用程序还可以注册一个回调函数，该回调函数在键值存储和文件存储更新后被调用。

### [SWRD\_API\_Per\_00085] RegisterApplicationDataUpdateCallback

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00085 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00356] | |
| **CR** |  | |
| Consistency | No | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | RegisterApplicationDataUpdateCallback(std::function  <void(const ara::core::InstanceSpecifier&storage, ara::core::String version)>  appDataUpdateCallback) | |
| **Scope:** | namespace ara::per | |
| **Syntax:** | void RegisterApplicationDataUpdateCallback(std::function  <void(const ara::core::InstanceSpecifier &storage, ara::core::String version)> appDataUpdateCallback) noexcept; | |
| **Parameters (in):** | appDataUpdateCallback | 持久化数据更新后要被持久化调用的回调函数.该函数将通过更新的键值存储或文件存储的短名称路径，以及最后一次访问持久性的可执行版本来调用。 |
| **Return value:** | None | |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | no | |
| **Header file:** | #include "ara/per/update.h" | |
| **Description:** | 用持久性注册应用程序数据更新回调  如果需要更新存储的应用程序数据，则持久化将调用提供的回调函数。此决定基于可执行版本。  最后一次访问持久性的版本作为回调函数的参数提供，InstanceSpecifier也引用更新后的键值存储或文件存储，根据这些信息，应用程序可以决定哪些更新是实际需要的，例如，可以支持从任何旧版本迁移，每个版本都需要不同的步骤。  提供的函数将从updatepersistence()、OpenKeyValueStorage()或OpenFileStorage()的上下文中调用。 | |
| **Additional:** |  | |

### [SWRD\_API\_Per\_00086] UpdatePersistency

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00086 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |  |
| --- | --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00357] | | |
| **CR** |  | | |
| Consistency | Yes | | |
| **Change Type** | 修改 修改返错 | | |
| **Kind:** | function | | |
| **Symbol:** | UpdatePersistency() | | |
| **Scope:** | namespace ara::per | | |
| **Syntax:** | ara::core::Result<void> UpdatePersistency () noexcept; | | |
| **Return value:** | ara::core::Result< void > | | 结果，要么为空，要么包含为PerErrc中定义的错误之一。 |
| **Exception Safety:** | noexcept | | |
| **Thread Safety:** | no | | |
| **Errors:** | PerErrc::kPhysicalStorageFailure | 在更新操作期间访问存储失败，则返回。 | |
| PerErrc::kIntegrityCorrupted | 如果存储的数据因结构完整性损坏而无法读取，则返回。 | |
| PerErrc::kValidationFailed | 如果不能确保存储数据的有效性，则返回。 | |
| PerErrc::kEncryptionFailed | 如果在更新操作期间对存储数据的加密或解密失败，则返回。 | |
| PerErrc::kResourceBusy | 返回如果ResetPersistency目前正在执行,或者RecoverKeyValueStorage或重置KeyValueStorage目前正在执行的任何键值存储,或者如果RecoverAllFiles或ResetAll文件目前正在执行任何文件存储,或键值存储或文件存储的SharedHandle目前正在使用。 | |
| PerErrc::kOutOfStorageSpace | 当可用存储空间不足时返回 | |
| PerErrc::kNotInitialized | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 | |
| **Header file:** | #include "ara/per/update.h" | | |
| **Description:** | 安装新清单后，更新所有持久化文件存储和键值存储。  此方法可用于在验证阶段更新应用程序的持久数据。 | | |
| **Additional:** | UpdatePersistency(); | | |

### [SWRD\_API\_Per\_00087] ResetPersistency

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00087 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | L |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00358] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 修改 修改返错 | |
| **Kind:** | function | |
| **Symbol:** | ResetPersistency() | |
| **Scope:** | namespace ara::per | |
| **Syntax:** | ara::core::Result<void> ResetPersistency () noexcept; | |
| **Return value:** | ara::core::Result< void > | 结果，要么为空，要么包含为PerErrc中定义的错误之一。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | no | |
| **Errors:** | PerErrc::kPhysicalStorageFailure | 在更新操作期间访问存储失败将返回。 |
| PerErrc::kResourceBusy | 返回如果UpdatePersistency目前正在执行,或者RecoverKeyValueStorage或重置KeyValueStorage目前正在执行的任何键值存储,或者如果RecoverAllFiles或ResetAll文件目前正在执行任何文件存储,或键值存储或文件存储的SharedHandle目前正在使用 |
| PerErrc::kNotInitialized | 如果在ara::core::Initialize之前或ara::core::Deinitialize之后调用此函数将返回。 |
| **Header file:** | #include "ara/per/update.h" | |
| **Description:** | 重置所有文件存储和键值存储，完全删除其内容  下次调用OpenFileStorage或Open KeyValueStorage时，将重新创建文件存储和键值存储。 | |
| **Additional:** | ResetPersistency(); **P3** | |

## Redundancy Handling接口定义

Per支持键值存储、文件存储、以及 键值对和文件的冗余存储。当访问键值存储或文件存储时，遇到数据错误可以使用冗余数据来修复，存储数据中的错误将被隐式修复，只有当冗余修复失败时，持久性才会返回错误。

为了能够跟踪存储错误是否已经使用可用冗余修复，应用程序可以注册以下回调函数。

### [SWRD\_API\_Per\_00088] RecoveryReportKind 枚举

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00088 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | L |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00432] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 新增 | |
| **Kind:** | enumeration | |
| **Symbol:** | RecoveryReportKind | |
| **Scope:** | namespace ara::per | |
| **Underlying type:** | uint32\_t | |
| **Syntax:** | enum class RecoveryReportKind : uint32\_t {...}; | |
|  | kKeyValueStorageRecoveryFailed= 1 | 键值存储已损坏，存在的有效副本数量不足。存储包含短名称路径的键值存储，报告的元素为空。reportedInstances包含受影响的键值存储副本的索引。 |
| kKeyValueStorageRecovered= 2 | 键值存储已损坏，但存在足够数量的有效副本，存储包含短名称路径的键值存储，报告的元素为空，reportedInstances包含受影响的键值存储副本的索引。 |
| kKeyRecoveryFailed= 3 | 一组键值对已损坏，存在的有效副本数量不足，storage包含键值存储的短名称路径。reportedElements包含受影响的键列表，reportedInstances包含受影响的键值存储或键副本的索引。 |
| kKeyRecovered= 4 | 一组键值对已损坏，但存在足够数量的有效副本，storage包含键-值存储的短名称路径，reportedElements包含受影响的键列表，reportedInstances包含受影响的键-值存储或键副本的索引。 |
| kFileStorageRecoveryFailed= 5 | 文件存储已损坏，存在的有效副本数量不足。storage包含文件存储的短名称路径，reports Elements为空，reportedInstances包含受影响的文件存储副本的索引。 |
|  | kFileStorageRecovered= 6 | 文件存储已损坏，但存在足够数量的有效副本。storage包含文件存储的短名称路径，reports Elements为空，reportedInstances包含受影响的文件存储副本的索引。 |
| kFileRecoveryFailed= 7 | 一组文件已损坏，存在的有效副本数量不足。storage包含文件存储的短名称路径，reports Elements包含受影响的文件名列表，reportedInstances包含受影响的文件存储或文件副本的索引。 |
| kFileRecovered= 8 | 一组文件已损坏，但存在足够数量的有效副本。storage包含文件存储的短名称路径，reports Elements包含受影响的文件名列表，reportedInstances包含受影响的文件存储或文件副本的索引。 |
| **Header file:** | #include "ara/per/recovery.h" |  |
| **Description:** | 定义恢复操作. |  |
| **Additional:** | **P3** |  |

### [SWRD\_API\_Per\_00089] RegisterRecoveryReportCallback

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00089 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00433] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 新增 | |
| **Kind:** | function | |
| **Symbol:** | RegisterRecoveryReportCallback(std::function<void(const ara::core::InstanceSpecifier &storage,ara::per::recoveryReportKind recoveryReportKind, ara::core::Vector<ara::core::String > reportedElements, ara::core::Vector< uint8 > reportedInstances)> recovery ReportCallback) | |
| **Scope:** | namespace ara::per | |
| **Syntax:** | void RegisterRecoveryReportCallback (std::function< void(const ara::core::InstanceSpecifier &storage, ara::per::recoveryReportKind recoveryReportKind, ara::core::Vector< ara::core::String > reportedElements, ara::core::Vector< uint8 > reportedInstances)> recoveryReportCallback) noexcept; | |
| **Parameters (in):** | recoveryReportCallback | 持久化调用的回调函数，用于报告存储数据中使用可用冗余纠正的错误。将使用受影响的键值存储或存储中的文件存储的短名称路径调用该函数，并在参数recoveryReportKind、已报告的元素和reportedInstances中显示已纠正的内容。 |
| **Return value:** | None | |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | no | |
| **Header file:** | #include "ara/per/recovery.h" | |
| **Description:** | 注册一个具有持久性的恢复报告回调。  该回调可用于安全意识强的应用程序中，以检测与持久化数据的正确性和存储的可靠性相关的持久性操作. | |
| **Additional:** | **P3** | |

## Handle Classes接口定义

本章是持久化集群的API中使用的句柄类的定义。ara::per::SharedHandle(通过typenameT模板化)用于提供对ara::per::KeyValueStorage或ara::per::FileStorage的共享访问，而ara::per::UniqueHandle(通过typenameT模板化)用于提供对文件存储的ara::per::ReadAccessor或ara::per::ReadWriteAccessor的非共享访问。

### [SWRD\_API\_Per\_00090] SharedHandle class

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00090 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00362] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 不变 | |
| **Kind:** | class | |
| **Symbol:** | SharedHandle | |
| **Scope:** | namespace ara::per | |
| **Syntax：** | template <typename T> class SharedHandle final {...}; | |
| **Template param:** | typename T | – |
| **Header file:** | #include "ara/per/shared\_handle.h" | |
| **Description:** | 文件存储或键值存储的句柄。  SharedHandle由函数OpenFileStorage()和OpenKeyValueStorage()返回，并可以根据需要在线程之间传递。  它提供了允许OpenFileStorage()和OpenKeyValueStorage()的线程安全实现所必需的抽象。 | |
| **Additional:** |  | |

#### [SWRD\_API\_Per\_00091] SharedHandle

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00091 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00367] | |
| **CR** |  | |
| **Consistency** | No | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | SharedHandle(SharedHandle &&sh) | |
| **Scope:** | class ara::per::SharedHandle | |
| **Syntax:** | SharedHandle (SharedHandle &&sh) noexcept; | |
| **Parameters (in):** | sh | 要移动的SharedHandle对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/shared\_handle.h" | |
| **Description:** | 移动构造函数  源句柄对象无效，不能再使用。  操作符bool()应使用在使用句柄对象的任何其他操作符之前检查句柄对象的状态。 | |
| **Additional:** |  | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00092 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00369] | |
| **CR** |  | |
| **Consistency** | No | |
| **Change Type** | 修改参数 | |
| **Kind:** | function | |
| **Symbol:** | SharedHandle(const SharedHandle &sh) | |
| **Scope:** | class ara::per::SharedHandle | |
| **Syntax:** | SharedHandle (const SharedHandle &sh) noexcept; | |
| **Parameters (in):** | sh | 被拷贝的SharedHandle对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/shared\_handle.h" | |
| **Description:** | SharedHandle的拷贝构造函数. | |
| **Additional:** |  | |

#### [SWRD\_API\_Per\_00093] operator=

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00093 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00368] | |
| **CR** |  | |
| **Consistency** | No | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | operator=(SharedHandle &&sh) | |
| **Scope:** | class ara::per::SharedHandle | |
| **Syntax:** | SharedHandle& operator= (SharedHandle &&sh) &noexcept; | |
| **Parameters (in):** | sh | 要移动的SharedHandle对象。 |
| **Return value:** | SharedHandle & | 移动的SharedHandle对象。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/shared\_handle.h" | |
| **Description:** | 移动赋值运算符  源句柄对象无效，不能再使用。  操作符bool()应使用在使用句柄对象的任何其他操作符之前检查句柄对象的状态。 | |
| **Additional:** |  | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00094 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00370] | |
| **CR** |  | |
| **Consistency** | No | |
| **Change Type** | 修改参数 | |
| **Kind:** | function | |
| **Symbol:** | operator=(const SharedHandle &sh) | |
| **Scope:** | class ara::per::SharedHandle | |
| **Syntax:** | SharedHandle& operator= (const SharedHandle &sh) &noexcept; | |
| **Parameters (in):** | sh | 要移动的SharedHandle对象。 |
| **Return value:** | SharedHandle & | 移动的SharedHandle对象。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/shared\_handle.h" | |
| **Description:** | 拷贝运算符 | |
| **Additional:** |  | |

#### [SWRD\_API\_Per\_00095] operator bool

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00095 |
| **Upstream ID** | [SWS\_PER\_00398] |
| **Consistency** | Yes |
| **Change Type** | 不变 |
| **Kind:** | function |
| **Symbol:** | operator bool() |
| **Scope:** | class ara::per::SharedHandle |
| **Syntax:** | explicit operator bool () const noexcept; |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | re-entrant |
| **Header file:** | #include "ara/per/shared\_handle.h" |
| **Description:** | 处理状态。  如果句柄代表模板类的有效对象，则为True;如果句柄为空，则为False（例如操作以后）  使用bool()之外的其他空句柄操作符将导致未定义的行为。 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00096] operator->

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00096 | |
| **Upstream ID** | [SWS\_PER\_00363] | |
| **Consistency** | Yes | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | operator->() | |
| **Scope:** | class ara::per::SharedHandle | |
| **Syntax:** | T\* operator-> () noexcept; | |
| **Return value:** | T \* | – |
| **Exception Safety:** | noexcept |  |
| **Thread Safety:** | re-entrant |  |
| **Header file:** | #include "ara/per/shared\_handle.h" |  |
| **Description:** | 变量 ->运算符 |  |
| **Additional:** |  |  |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00097 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00364] | |
| **Consistency** | Yes | |
| **Change Type** | 修改返回值 | |
| **Kind:** | function | |
| **Symbol:** | operator->() | |
| **Scope:** | class ara::per::SharedHandle | |
| **Syntax:** | const T\* operator-> () const noexcept; | |
| **Return value:** | const T \* | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/shared\_handle.h" | |
| **Description:** | 常量 ->运算符 | |
| **Additional:** |  | |

#### [SWRD\_API\_Per\_00098] operator\*

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00098 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00402] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 不变 | |
| **Kind:** | function |  |
| **Symbol:** | operator\*() |  |
| **Scope:** | class ara::per::SharedHandle |  |
| **Syntax:** | T& operator\* () noexcept; |  |
| **Return value:** | T & | – |
| **Exception Safety:** | noexcept |  |
| **Thread Safety:** | re-entrant |  |
| **Header file:** | #include "ara/per/shared\_handle.h" | |
| **Description:** | 变量解除引用操作. |  |
| **Additional:** |  |  |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00099 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00403] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 修改返回值 | |
| **Kind:** | function | |
| **Symbol:** | operator\*() | |
| **Scope:** | class ara::per::SharedHandle | |
| **Syntax:** | const T& operator\* () const noexcept; | |
| **Return value:** | const T & | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/shared\_handle.h" | |
| **Description:** | 常量解除引用操作. | |
| **Additional:** |  | |

### [SWRD\_API\_Per\_00100] UniqueHandle 类

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00100 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00359] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 不变 | |
| **Kind:** | class | |
| **Symbol:** | UniqueHandle | |
| **Scope:** | namespace ara::per | |
| **Syntax:** | template <typename T> class UniqueHandle final {...}; | |
| **Template param:** | typename T | – |
| **Header file:** | #include "ara/per/unique\_handle.h" | |
| **Description:** | ReadAccessor或者ReadWriteAccessor的句柄。  当调用下面的函数时返回UniqueHandle。  OpenFileReadOnly(),OpenFileWriteOnly (),OpenFileReadWrite(). | |
| **Additional:** | Accessor的句柄。 当调用下面的函数时返回 OpenFileReadOnly(), OpenFileReadWrite(),OpenFileReadWrite() | |

#### [SWRD\_API\_Per\_00101] UniqueHandle

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00101 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00371] | |
| **CR** |  | |
| **Consistency** | No | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | UniqueHandle(UniqueHandle &&uh) | |
| **Scope:** | class ara::per::UniqueHandle | |
| **Syntax:** | UniqueHandle (UniqueHandle &&uh) noexcept; | |
| **Parameters (in):** | uh | 要移动的UniqueHandle对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/unique\_handle.h" | |
| **Description:** | UniqueHandle的移动构造函数.  源句柄对象无效，不能再使用。  操作符bool()应使用在使用句柄对象的任何其他操作符之前检查句柄对象的状态。 | |
| **Additional:** |  | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00102 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |
| --- | --- |
| **Upstream ID** | [SWS\_PER\_00373] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 修改参数 |
| **Kind:** | function |
| **Symbol:** | UniqueHandle(const UniqueHandle &) |
| **Scope:** | class ara::per::UniqueHandle |
| **Syntax:** | UniqueHandle (const UniqueHandle &)=delete; |
| **Header file:** | #include "ara/per/unique\_handle.h" |
| **Description:** | 拷贝构造函数，不应该被使用。 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00103] operator=

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00103 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00372] | |
| **CR** |  | |
| **Consistency** | No | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | operator=(UniqueHandle &&uh) | |
| **Scope:** | class ara::per::UniqueHandle | |
| **Syntax:** | UniqueHandle& operator= (UniqueHandle &&uh) &noexcept; | |
| **Parameters (in):** | uh | 要移动的UniqueHandle对象。 |
| **Return value:** | UniqueHandle & | 移动的UniqueHandle对象。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/unique\_handle.h" | |
| **Description:** | 移动赋值运算符  源句柄对象无效，不能再使用。  操作符bool()应使用在使用句柄对象的任何其他操作符之前检查句柄对象的状态。 | |
| **Additional:** |  | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00104 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |
| --- | --- |
| **Upstream ID** | [SWS\_PER\_00374] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 修改参数 |
| **Kind:** | function |
| **Symbol:** | operator=(const UniqueHandle &) |
| **Scope:** | class ara::per::UniqueHandle |
| **Syntax:** | UniqueHandle& operator= (const UniqueHandle &)=delete; |
| **Header file:** | #include "ara/per/unique\_handle.h" |
| **Description:** | 拷贝赋值运算符，不应该被使用. |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00105] operator bool

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00105 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |
| --- | --- |
| **Upstream ID** | [SWS\_PER\_00399] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 不变 |
| **Kind:** | function |
| **Symbol:** | operator bool() |
| **Scope:** | class ara::per::UniqueHandle |
| **Syntax:** | explicit operator bool () const noexcept; |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | re-entrant |
| **Header file:** | #include "ara/per/unique\_handle.h" |
| **Description:** | 句柄状态。  如果句柄代表了模板类的一个有效对象，则为True，如果句柄为空(例如在移动操作之后)则为False。  除了空句柄的bool()之外，其他操作符将导致未定义的行为。 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00106] operator->

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00106 | |
| **Upstream ID** | [SWS\_PER\_00360] | |
| Consistency | Yes | |
| **Change Type** | 不变 | |
| **Kind:** | function |  |
| **Symbol:** | operator->() |  |
| **Scope:** | class ara::per::UniqueHandle |  |
| **Syntax:** | T\* operator-> () noexcept; |  |
| **Return value:** | T \* | – |
| **Exception Safety:** | noexcept |  |
| **Thread Safety:** | re-entrant |  |
| **Header file:** | #include "ara/per/unique\_handle.h" |  |
| **Description:** | 变量 ->运算符 | |
| **Additional:** |  | |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00107 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |
| --- | --- |
| **Upstream ID** | [SWS\_PER\_00361] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 修改返回值 |

|  |  |  |
| --- | --- | --- |
| **Kind:** | function | |
| **Symbol:** | operator->() | |
| **Scope:** | class ara::per::UniqueHandle | |
| **Syntax:** | const T\* operator-> () const noexcept | |
| **Return value:** | const T \* | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/unique\_handle.h" | |
| **Description:** | 常量 ->运算符 | |
| **Additional:** |  | |

#### [SWRD\_API\_Per\_00108] operator\*

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00108 |
| **Upstream ID** | [SWS\_PER\_00400] |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |
| --- | --- |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 不变 |
| **Kind:** | function |
| **Symbol:** | operator\*() |
| **Scope:** | class ara::per::UniqueHandle |
| **Syntax:** | T& operator\* () noexcept; |
| **Return value:** | T & |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | re-entrant |
| **Header file:** | #include "ara/per/unique\_handle.h" |
| **Description:** | Non-constant dereference operator. （\*运算符） |
| **Additional:** |  |

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00109 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00401] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 修改返回值 | |
| **Kind:** | function | |
| **Symbol:** | operator\*() | |
| **Scope:** | class ara::per::UniqueHandle | |
| **Syntax:** | const T& operator\* () const noexcept; | |
| **Return value:** | const T & | – |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/unique\_handle.h" | |
| **Description:** | Constant dereference operator（\*运算符）. | |
| **Additional:** |  | |

## Errors接口定义

持久化集群实现了基于ara::core::Result的错误处理。持久集群支持的错误在下面列出。

### [SWRD\_API\_Per\_00110] PerErrc 枚举

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00110 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00311] | |
| **CR** |  | |
| Consistency | No | |
| **Change Type** | 修改 新增error | |
| **Kind:** | enumeration | |
| **Symbol:** | PerErrc | |
| **Scope:** | namespace ara::per | |
| **Underlying type:** | ara::core::ErrorDomain::CodeType | |
| **Syntax:** | enum class PerErrc : ara::core::ErrorDomain::CodeType {...}; | |
|  | kStorageNotFound= 1 | 没有配置key-Value Storage或File Storage的存储位置在AUTOSAR model中。 |
| kKeyNotFound= 2 | 在key-value存储中没有找到key。 |
| kIllegalWriteAccess= 3 | 文件配置成只读，打开文件写或者修改或者同步key失败。 |
| kPhysicalStorageFailure= 4 | 访问物理地存储是发生错误。例如，因为损坏的文件系统或损坏的硬件，或由于没有足够的访问权限。 |
| kIntegrityCorrupted= 5 | 无法建立存储的结构完整性，可能是key-value存储或者文件的元数据结构损坏了。 |
| kValidationFailed= 6 | 对于单个键、整个键值存储、单个文件或整个文件存储，冗余度量的验证失败。 |
| kEncryptionFailed= 7 | 单个密钥、整个键值存储、单个文件或整个文件存储的加密或解密失败。 |
| kDataTypeMismatch= 8 | 提供的数据类型与存储的数据类型不匹配。. |
| kInitValueNotAvailable= 9 | 没有可用的初始值进行初始化。 |
| kResourceBusy= 10 | 资源忙，无法进行该操作。 |
| kOutOfStorageSpace= 12 | 已分配的存储超过最大值。 |
| kFileNotFound= 13 | File Storage中文件没找到。 |
| kNotInitialized= 14 | Persistency中的方法在ara::core::Initialize()之前或者ara::core::Deinitialize()之后被调用。 |
| kInvalidPosition= 15 | SetPosition试图移动到一个无法到达的位置（例如，小于零或大于文件的当前大小） |
|  | kIsEof= 16 | 应用程序试图从文件的末尾或空文件中读取。 |
| kInvalidOpenMode= 17 | 由于请求的OpenModes组合无效，打开文件失败。 |
| kInvalidSize= 18 | 试图设置一个大于当前文件大小的新size。 |
| **Header file:** | #include "ara/per/per\_error\_domain.h" | |
| **Description:** | The enumeration values 0 - 255 are reserved for AUTOSAR assigned errors, the stack provider is free to define additional errors starting from 256.  Per定义的错误，枚举从0~255预留给AUTOSAR分配的错误，堆栈提供程序可以自由定义从256开始的额外的错误。 | |
| **Additional:** | **P1** | |

### [SWRD\_API\_Per\_00111] GetPerDomain

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00111 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00352] | |
| **CR** |  | |
| **Consistency** | Yes | |
| **Change Type** | 修改 修改返回值类型 | |
| **Kind:** | function | |
| **Symbol:** | GetPerDomain() | |
| **Scope:** | namespace ara::per | |
| **Syntax:** | constexpr const ara::core::ErrorDomain& GetPerDomain () noexcept; | |
| **Return value:** | const ara::core::ErrorDomain & | 全局PerErrorDomain对象。 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/per\_error\_domain.h" | |
| **Description:** | 返回全局的PerErrorDomain对象. | |
| **Additional:** | **P2** | |

### [SWRD\_API\_Per\_00112] MakeErrorCode

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00112 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |  |
| --- | --- | --- |
| **Upstream ID** | [SWS\_PER\_00351] | |
| **CR** |  | |
| Consistency | No | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | MakeErrorCode(PerErrc code, ara::core::ErrorDomain::SupportDataType data) | |
| **Scope:** | namespace ara::per | |
| **Syntax:** | constexpr ara::core::ErrorCode MakeErrorCode (PerErrc code, ara::core::ErrorDomain::SupportDataType data) noexcept; | |
| **Parameters (in):** | code | 错误码 |
| data | 与错误相关的数据 |
| **Return value:** | ara::core::ErrorCode | ErrorCode对象 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | re-entrant | |
| **Header file:** | #include "ara/per/per\_error\_domain.h" | |
| **Description:** | 创建错误码 | |
| **Additional:** |  | |

### [SWRD\_API\_Per\_00113] PerException 类

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00113 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |
| --- | --- |
| **Upstream ID** | [SWS\_PER\_00354] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 不变 |
| **Kind:** | class |
| **Symbol:** | PerException |
| **Scope:** | namespace ara::per |
| **Base class:** | ara::core::Exception |
| **Syntax:** | class PerException : public Exception {...}; |
| **Header file:** | #include "ara/per/per\_error\_domain.h" |
| **Description:** | Per模块的异常的类型 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00114] PerException

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00114 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00355] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 不变 | |
| **Kind:** | function | |
| **Symbol:** | PerException(ara::core::ErrorCode errorCode) | |
| **Scope:** | class ara::per::PerException | |
| **Syntax:** | explicit PerException (ara::core::ErrorCode errorCode) noexcept; | |
| **Parameters (in):** | errorCode | 错误码. |
| **Exception Safety:** | noexcept | |
| **Header file:** | #include "ara/per/per\_error\_domain.h" | |
| **Description:** | 构造函数：构造一个包含错误代码的新的Per异常对象 | |
| **Additional:** |  | |

### [SWRD\_API\_Per\_00115] PerErrorDomain 类

错误处理需要一个ara::core::ErrorDomain，它可以用来检查通过ara::core::Result返回的错误。

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00115 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00312] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 不变 |
| **Kind:** | class |
| **Symbol:** | PerErrorDomain |
| **Scope:** | namespace ara::per |
| **Base class:** | ara::core::ErrorDomain |
| **Syntax:** | class PerErrorDomain final : public ErrorDomain {...}; |
| **Unique ID:** | 0x8000’0000’0000’0101 |
| **Header file:** | #include "ara/per/per\_error\_domain.h" |
| **Description:** | 为per定义错误域 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00116] Errc

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00116 |
| **Type** | Valid |
| **Priority** | H |
| **Upstream ID** | [SWS\_PER\_00411] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 新增 |
| **Kind:** | type alias（类型别名） |
| **Symbol:** | Errc |
| **Scope:** | class ara::per::PerErrorDomain |
| **Derived from:** | PerErrc |
| **Syntax:** | using Errc = PerErrc; |
| **Header file:** | #include "ara/per/per\_error\_domain.h" |
| **Description:** | 错误码值枚举的别名 |
| **Additional:** | **P1** |

#### [SWRD\_API\_Per\_00117] Exception

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00117 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |
| --- | --- |
| **Upstream ID** | [SWS\_PER\_00412] |
| **CR** |  |
| Consistency | Yes |
| **Change Type** | 新增 |
| **Kind:** | type alias |
| **Symbol:** | Exception |
| **Scope:** | class ara::per::PerErrorDomain |
| **Derived from:** | PerException |
| **Syntax:** | using Exception = PerException; |
| **Header file:** | #include "ara/per/per\_error\_domain.h" |
| **Description:** | 异常基类的别名. |
| **Additional:** | **P1** |

#### [SWRD\_API\_Per\_00118] PerErrorDomain

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00118 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |
| --- | --- |
| **Upstream ID** | [SWS\_PER\_00313] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 不变 |
| **Kind:** | function |
| **Symbol:** | PerErrorDomain() |
| **Scope:** | class ara::per::PerErrorDomain |
| **Syntax:** | PerErrorDomain () noexcept; |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | no |
| **Header file:** | #include "ara/per/per\_error\_domain.h" |
| **Description:** | 构造函数，创建一个PerErrorDomain实例 |
| **Additional:** |  |

#### [SWRD\_API\_Per\_00119] Name

|  |  |
| --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00119 |

|  |  |
| --- | --- |
| **Type** | Valid |
| **Priority** | H |

|  |  |
| --- | --- |
| **Upstream ID** | [SWS\_PER\_00314] |
| **CR** |  |
| **Consistency** | Yes |
| **Change Type** | 修改 修改返回值类型 |
| **Kind:** | function |
| **Symbol:** | Name() |
| **Scope:** | class ara::per::PerErrorDomain |
| **Syntax:** | const char\* Name () const noexcept override; |
| **Return value:** | const char \* 错误域的名称。 |
| **Exception Safety:** | noexcept |
| **Thread Safety:** | re-entrant |
| **Header file:** | #include "ara/per/per\_error\_domain.h" |
| **Description:** | 返回错误域的名称。. |
| **Additional:** |  |

[SWS\_PER\_00353] PerErrorDomain::Name将返回以null结尾的字符串"Per"(RS\_PER\_00001)

#### [SWRD\_API\_Per\_00120] Message

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00120 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00315] | |
| Consistency | Yes | |
| **Change Type** | 修改 修改返回值 | |
| **Kind:** | function | |
| **Symbol:** | Message(CodeType errorCode) | |
| **Scope:** | class ara::per::PerErrorDomain | |
| **Syntax:** | const char\* Message (CodeType errorCode) const noexcept override; | |
| **Parameters (in):** | errorCode | 错误码 |
| **Return value:** | const char \* | 与错误码关联的消息 |
| **Exception Safety:** | noexcept | |
| **Thread Safety:** | no | |
| **Header file:** | #include "ara/per/per\_error\_domain.h" | |
| **Description:** | 返回错误码关联的信息。 | |
| **Additional:** |  | |

#### [SWRD\_API\_Per\_00121] ThrowAsException

|  |  |  |
| --- | --- | --- |
| **SWRD\_ID:** | SWRD\_API\_Per\_00121 | |
| **Type** | Valid | |
| **Priority** | H | |
| **Upstream ID** | [SWS\_PER\_00350] | |
| **CR** |  | |
| Consistency | Yes | |
| **Change Type** | 修改 修改参数类型 | |
| **Kind:** | function | |
| **Symbol:** | ThrowAsException(const ara::core::ErrorCode &errorCode) | |
| **Scope:** | class ara::per::PerErrorDomain | |
| **Syntax:** | void ThrowAsException (const ara::core::ErrorCode &errorCode) const override; | |
| **Parameters (in):** | errorCode | 抛出错误. |
| **Return value:** | None | |
| **Thread Safety:** | no | |
| **Header file:** | #include "ara/per/per\_error\_domain.h" | |
| **Description:** | 抛异常关联的错误码。 | |
| **Additional:** |  | |

附录A- 信息定义

|  |  |  |
| --- | --- | --- |
| 类别 | 结构 | 备注 |
| 需求 | SWRD\_{需求类型}\_{功能简称}\_流水号  功能简称：参见下面功能简称列表  需求类型：功能需求为空，非功能需求为NF,接口为API  流水号：从00001开始的5位自然数 | 例：  SWRD\_Nvm\_00001  SWRD\_NF\_Nvm\_00001  SWRD\_API\_Nvm\_00001 |

|  |  |
| --- | --- |
| 功能简称列表（aCore） | 说明 |
| DM\_DEM | Diagnostics management模块的诊断事件管理 |
| DM\_DCM | Diagnostics management模块的诊断通信管理 |
| DM\_DCM\_DOIP | Diagnostics management模块的DO/IP相关功能 |
| CoreTypes | 核心数据类型 |
| CM\_SOMEIP | Communication management模块的SOME/IP相关功能 |
| CM\_DDS | Communication management模块的DDS相关功能 |
| CM\_CommunicationGroup | Communication management模块的通信组相关功能 |
| CM\_SHM | Communication management模块的共享内存相关功能 |
| CM\_IPC | Communication management模块的IPC相关功能 |
| CM\_Raw | Communication management模块的raw data streaming相关功能 |
| CM\_TLS | Communication management模块的TLS相关功能 |
| CM\_S2S | Communication management模块的S2S相关功能 |
| CM\_E2E | Communication management模块的E2E相关功能 |
| UCM\_Master | Update and config management模块的主站相关功能 |
| UCM\_Server | Update and config management模块的从站相关功能 |
| LT | Log and trace模块相关功能 |
| PHM | Platform health management模块相关功能 |
| Per | Persistency模块相关功能 |
| SM | State management模块相关功能 |
| Crypto | Cryptography模块相关功能 |
| EM | Execution mangement模块相关功能 |
| NM | Network management模块相关功能 |
| TS | Time synchronization模块相关功能 |

说明：根据项目情况可自己定义，增加功能简称

|  |  |  |
| --- | --- | --- |
| 安全等级(ASIL) | 解释说明 | 备注 |
| ASIL A | 根据S – Severity(严重度)  E – Exposure（暴露度） C – Controllability（可控性） 排定功能安全等级。详细理解可以参考26262标准文件。 | 如果有关于ASIL等级的特殊解释说明，请记录在此 |
| ASIL B |  |
| ASIL C |  |
| ASIL D |  |
| QM(A) | 从ASIL A到ASIL D 中拆分出来，拆分的标准，参考功能安全体系文件《功能安全需求分解指南\_FS.pdf》 |  |
| QM(B) |  |
| QM(C) |  |
| QM(D) |  |
| ASIL A(A) |  |
| ASIL A(B) |  |
| ASIL A(C) |  |
| ASIL A(D) |  |
| ASIL B(B) |  |
| ASIL B(C) |  |
| ASIL B(D) |  |
| ASIL C(C) |  |
| ASIL C(D） |  |
| ASIL D(D) |  |

|  |  |  |
| --- | --- | --- |
| 优先级（Priority） | 解释说明 | 备注 |
| H | 高优先级 | 例：被依赖的需求优先级设置为H级别 |
| M | 中优先级 | 例： |
| L | 低优先级 | 例：其余功能均设置为L级别 |

|  |  |  |
| --- | --- | --- |
| 状态  （Status） | 状态说明 | 备注 |
| Draft | 草稿 | 例：表示新建 |
| In Review | 评审中 | 例：表示处于评审中 |
| Approved | 批准 | 例：表示通过评审 |
| Released | 发布 | 例：表示通过客户确认 |
| Modified | 修改 | 例：表示正在检讨修改中 |

|  |  |  |
| --- | --- | --- |
| 类型  （Type） | 状态说明 | 备注 |
| Valid | 有效 | 例：表示需要对应 |
| InValid | 不适用 | 例：表示不做对应 |
| TBD | 检讨中 | 例：表示正在检讨中 |

说明：根据项目情况可自己定义，但需要明确

|  |  |  |
| --- | --- | --- |
| 变更类型  (Change Type) | 解释说明 | 备注 |
| 新增 | 相对已建立的第一版需求基线（含Base项目的需求基线），如果是新增的需求，选择此项 | 如果有关于每个变更类型的特殊解释说明，请记录在此 |
| 修改 | 相对已建立的第一版需求基线（含Base项目的需求基线），发生了修改的需求 |  |
| 不变 | 相对已建立的base项目的需求基线，复用了base项目的需求，填此类型，如没有Base项目需求基线，不应填此类型。 |  |
| 删除 | 相对已建立的第一版需求基线（含Base项目的需求基线），如果是删除的需求，选择此项。 |  |

说明：根据项目情况可自己定义，但需要明确

附录B- 配置信息

|  |  |  |  |
| --- | --- | --- | --- |
| 配置信息 | 说明 | 范围 | 备注 |
| API configuration class |  | 1、2、3 |  |
|  |  |  |  |
|  |  |  |  |

东软秘密,未经许可不得扩散

软件需求说明书 第16页

东软秘密，未经许可不得扩散