

LGE Internal Use Only

pjt\_SDD\_0x\_ComponentName v1.0

LGE VS [OEM Name] [Project Name]

**Software Detailed Design**

For SW Component Name

20YY.MM.DD

Gildong Hong

**XY Team**

**CV Company / XY Division**

Revision History

|  |
| --- |
| Document histories are arranged in the order that the most recent histories are at the top, and the first histories are at the bottom.  The version mark is ‘target version + Draft indicator’.  eg v1.0a : first draft targeting v1.0 ; Yes, v1.0b : second draft targeting v1.0  Yes v1.0 : v1.0 approved version |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Comment | Author | Approver |
|  |  |  |  |  |
| 1.0a | 2016-01-20 | First draft | GD.Hong |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **About This Template**   * • Document Name: LGE\_VS\_SWDC\_T02\_SW Detailed Design (SDD)\_Type2  |  |  |  |  |  | | --- | --- | --- | --- | --- | | Version | Date | Comment | Author | Approver | | 1.1 | 2021-06-15 | Updated security notice of this template  (Before: LGE Confidential->After: LGE Internal Use Only) Security level related note (the last sentence in red color below) | VC Smart SW Process Team | VS SW Process Team Leader | | 1.0 | 2019-03-29 | Initial Release | VC Smart SW Process Team | VS SW Process Team Leader | |
| * This template is a basic form for a Software Detailed Design document and consists of guidelines (light green boxes) and examples. * Before writing the document, read and understand the General Guidelines. * After writing the document, all guidelines (light green boxes) should be deleted. * The notice “LGE Internal Use Only” is for this template itself. The document which use this template needs to be classified as suitable security level according to its content. |

|  |
| --- |
| **General Guidelines**  • SDD is written as an independent document for each component determined by SAD. Refer to [12] when changing the configuration.  • The definitions of SW Element, SW Component, and SW Unit used in this document are as follows.  - SW Element: Among the Sys elements defined in SyAD, the elements implemented as SW  - SW Component: As the lowest level element defined in SAD, it becomes the unit of SDD.  - SW Unit: SW component of a small unit that cannot be further divided. function in c.  • There are no restrictions on Design Tool. (The examples in this template were created using EA) |
| * Body font: Arial * Add table number and figure number (insert caption automatically): In Word, select Figure or Table in [Reference] > [Insert Caption] > [Label] and select OK * Insert a caption below the figure: Figure 1 figure title, Figure 2 figure title, .. * Insert captions above tables: Table 1 table title, Table 2 table title, .. |

Table of Contents

|  |
| --- |
| If you right-click on the table of contents below and select the [Update Field] menu, the change is reflected. |

[1 Introduction １](#_Toc4488535)

[1.1 Purpose １](#_Toc4488536)

[1.2 Scope １](#_Toc4488537)

[1.3 Audience １](#_Toc4488538)

[1.4 Conventions １](#_Toc4488539)

[1.5 Acronyms / Glossary １](#_Toc4488540)

[1.6 Related Documents １](#_Toc4488541)

[2 External Interfaces 2](#_Toc4488542)

[2.1 Context Diagram 2](#_Toc4488543)

[2.2 External interface list 2](#_Toc4488544)

[3 Static Design 3](#_Toc4488545)

[4 Dynamic Design 5](#_Toc4488546)

[4.1 State Design 5](#_Toc4488547)

[4.2 Object Diagram 6](#_Toc4488548)

[4.3 Interaction Design 6](#_Toc4488549)

[4.3.1 [Interface 1] interface 6](#_Toc4488550)

[4.3.2 [Interface 2] interface 7](#_Toc4488551)

[4.4 Other Diagrams 7](#_Toc4488552)

[5 Algorithm Design 8](#_Toc4488553)

[5.1 [Class 1] class 8](#_Toc4488554)

[5.2 [Class 2] class 9](#_Toc4488555)

[6 Global Variables 10](#_Toc4488556)

[6.1 Define 10](#_Toc4488557)

[6.2 Type definition 10](#_Toc4488558)

[6.3 Data definition 10](#_Toc4488559)

[7 Design Alternatives 11](#_Toc4488560)

[7.1 Title A 11](#_Toc4488561)

Figures

[Figure 2.1 Context diagram for ComponnetX 2](#_Toc4488562)

[Figure 3.1 Class diagram for [Component 1] component 3](#_Toc4488563)

[Figure 4.1 State diagram 5](#_Toc4488564)

[Figure 4.2 Sequence diagram for [Interface 1] interface 6](#_Toc4488565)

[Figure 5.1 Detailed class diagram with attribute and method 8](#_Toc4488566)

[Figure 7.1 alternative Design #1 11](#_Toc4488567)

**Tables**

[Table 2.1 External interface list 2](#_Toc4488568)

[Table 3.1 Classes that [Component 1] component consists of 4](#_Toc4488569)

[Table 4.1 Description of each state 5](#_Toc4488570)

[Table 4.2 State transitions (optional) 5](#_Toc4488571)

# Introduction

## Purpose

This document specifies the Software Detailed Design (SDD) for the AppManager including the static design, dynamic design, and algorithm design.

## Scope

* This document covers for XY.
* This document applies XY Model.
* This document doesn’t deal with XX.

## Audience

The target audience of this document is:

* Requirement engineer, Project manager, Software architect, Component developer
* SW integration Test engineers

## Conventions

Unit :

UML notation : 2.0

## Acronyms / Glossary

|  |
| --- |
| Abbreviations and terms used in the text are written in alphabetical order. |

|  |  |
| --- | --- |
| Acronym | Description |
| IHU | IVI Head Unit |
| SAD | Software Architectural Design |
| SDD | Software Detailed Design |
|  |  |

|  |  |
| --- | --- |
| Glossary | Description |
|  |  |

## Related Documents

|  |
| --- |
| Describe the list of documents referenced when preparing this document.  Includes all parent documents. Order: [n] Author, document name, document number, version |

Documents related to this document include:

[1] LGE, Customer Requirements Specifications, pjt\_CRS, v1.0

[2] LGE, System Requirements Specifications, pjt\_SyRS\_feature, v1.0

[3] LGE, System Architecture Design, pjt\_SyAD, v1.0

[4] LGE, Software Requirements Specifications, pjt\_SRS\_feature, v1.0

[5] LGE, System Architecture Design, pjt\_SAD\_00, v1.0

# External Interfaces

This is the software architectural design of OOO. OOO consists of Remote Trip Statistics, Remote Battery Charge, Remote Honk&Flash, ….

## Context Diagram

<Based on HLD, organize messages/APIs allocated to components (exposed to the outside). It describes in detail the meaning of the parameters, the range of valid values, and the meaning of the return value. If possible, organize messages by port.>



Figure 2.1 Context diagram for ComponnetX

## External interface list

<Based on HLD, organize messages/APIs allocated to components (exposed to the outside). It describes in detail the meaning of the parameters, the range of valid values, and the meaning of the return value. If possible, organize messages by port.>

Table 2.1 External interface list

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| # | Port | Interface | Description | Parameters (range) | Return (range) | Error code | SRS\_ID |
| 1 |  | DoBarrier |  | void | Void |  |  |
| 2 |  | OnFinalize |  | Int (0~10) | Boolean(1:OK, 0:fail) |  |  |
| 3 |  | OnInitialize |  |  |  |  |  |
| 4 |  | OnMessage |  |  |  |  |  |

# Static Design

|  |
| --- |
| **Component internal design can be expressed using class diagram, and it is okay to use other diagrams that can effectively express this according to the characteristics of each component.**  **This chapter identifies the classes composing components and defines the relationship between them. In this chapter, only classes are identified, and functions/operations/methods are identified through Chapter 4.**  **If the interface between components is defined through SAD interaction design, it can be implemented by inheriting the interface defined in the component as a class.**  **- Object-oriented languages such as C++/Java: Identification of classes composing components**  **- C: The file composing component can be modeled as a class.**  **Note: File in C language is modeled by mapping it to Class.** |

The class diagram for the [Component 1] component is shown below:



Figure 3.1 Class diagram for [Component 1] component

The [Component 1] stores the commands such as resuming and pausing Application. The [Component 1] schedules them whether to execute or skip the commands at a specific time. This command scheduling is necessary if the command comes faster than the processing capability.

The [Component 1] component is realized by using the command pattern. In command pattern, there are two main classes; the CommandInvoker class that stores the incoming commands, and invokes them according to the policy, and the Command class that abstracts each command so as for the CommandInvoker class deals with them in general manner. Specific commands are realized by inheriting the [Class 1] class.

The classes identified above are described in the below table:

Table 3.1 Classes that [Component 1] component consists of

|  |  |  |
| --- | --- | --- |
| # | Class | Descriptions |
| 1 | CommandInvoker | This class is the abstract class which is the super class for all concrete command. This class follows the Command Pattern. |
| 2 | [Class 2] | … |
| 3 | [Class 3] | … |
| 4 | … | … |
| 5 | … | … |
| 6 | … | … |

As shown

# Dynamic Design

|  |
| --- |
| Design the internal operation behavior of the design target component. Dynamic Design related to this component can be described through State Diagram or Sequence Diagram. Select and describe the diagram that can effectively express the component to be implemented. |

## State Design

|  |
| --- |
| For each internal class of component, the design for class state change is described in State Design.  Below is an example of a State Diagram. |

The [Class 1] class under schedule shall conform to the following state:



Figure 4.1 State diagram

The descriptions for each state are as follows:

Table 4.1 Description of each state

|  |  |
| --- | --- |
| States | Descriptions |
| Initial | This is the initial state. |
| Created | … |
| … | … |

|  |
| --- |
| Create a table for state transition. (Only used when adding content that is not in the state diagram) |

The state transitions for the state diagram are as follows:

Table 4.2 State transitions (optional)

|  |  |  |  |
| --- | --- | --- | --- |
| Current State | Event/Action | Next State | Descriptions |
| Initial | n/a | Created | This is the initial state transition. |
| Created | Enque/Stored the command | Queued | Created command is stored at the queue. |
| … | … | … | … |

## Object Diagram

<(optional) The internal structure of the component is expressed using an object diagram.>

<Define/explain the relationship between objects created during operation and, if necessary, ports, etc.>

## Interaction Design

|  |
| --- |
| Design how classes identified in Static Design interact. Interaction is designed in units of Objects in which Classes identified and defined in Static Design are instantiated.  It is divided into subsections for each external interface identified in Section 2.2, and internal processing is expressed as a sequence diagram. The scope of the sequence diagram is specified up to adjacent components and internal Class/Object.  The goal is to identify all public/private operations of each class/object through the sequence diagram. |

### [Interface 1] interface



Figure 4.2 Sequence diagram for [Interface 1] interface

|  |
| --- |
| At the end of each step description, write SRS\_ID in parentheses. Step number 1.0 is used for external interface, and 1.1 is used for internal processing. The steps corresponding to the pre-condition are attached as 0.1, 0.2. |

1.0 The [Interface 1] () method of the CommandInvoker is called by … (SRS\_ID)  
 1.1 …

1.2 …

|  |
| --- |
| Precautions when expressing the interaction between classes in a sequence diagram  1. The start of the sequence must coincide with the External Interface call. In the above example, the interface name [Interface 1] and Step 1.0 match.  2. External components and internal objects have different colors.  3. To repeat the same sequence, use a loop.  4. Frequently used routines use references to avoid repetitive design. |

### [Interface 2] interface

## Other Diagrams

< If necessary, express the design using activity diagrams, etc.>

# Algorithm Design

|  |
| --- |
| After the interaction between Class/Object is designed in Chapter 4, Operation is identified within Classes in the process. (Optional) Attach the class diagram so that the functions inside each class show Attribute/Operations. Even if there is no class diagram, the contents can be understood through the sub-sections, so only insert pictures when necessary. Add comments if necessary.  Note: File in C language is written according to Class, and function in C language is written according to Method. |

The [Component 1] Component is realized by using the command pattern. As a normal command pattern, it has the CommandInvoker class, and the [Class 1] class that acting as a baseclass for all concrete command class. Currently, there exists five concrete subclass of the [Class 1] class; ApplicationResume class, ApplicationStop class, ApplicationPause class, ApplicationLaunch class, FactoryReset class. Each of them represents the command.

When the command is created, then it shall be added to the CommandInvoker that stores the added command and executes them at the later time.



Figure 5.1 Detailed class diagram with attribute and method

< The following describes the attributes and operations belonging to each class. Describe in detail the meaning of the operation parameters, the range of valid values, and the meaning of the return value.>

## [Class 1] class

**Attribute**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Visibility | Name | Type | Description (range) |
| 1 | Public |  |  |  |
| 2 | Private |  |  |  |

**Operations**

**<** **Detailed design for all operation/method. If algorithm expression is required, Flowchart or Pseudocode can be used. Statechart diagram in case of managing state, and call graph in case of complex function call relationship.>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Visibility** | **Operation** | **Description** | **Inputs/Outputs/Error code** | **SRS\_ID** |
| 1 | Public | AddCommand | This method adds command to the CommandInvoker. The added command is to be stored and will be executed later according to the policy. | Input   * Application\* app: the added command |  |
| Algorithm |  | | | |
| 2 | Private | [Method 1-2] | … | … |  |
| Algorithm |  | | | |

< When using EA, you can use the document generation function to organize by function as follows.>

##### entry\_observer

**Full Name:** main:: entry\_observer

**Visibility:** Public

**Description:** Thread for system operation. (SRS\_ID: UC1.1\_3)

**Parameters:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| thread\_input | ULONG | thread entry parameter.  0UL ~ 4294967295UL |

**Return Type:** void

**Function Algorithm (Pseudo Code):**

1. Call the functions related to initialization.

2. Execute the infinite loop for system operation.

3. Return x (x: 1~100)

## [Class 2] class

# Global Variables

< Global data should be avoided as much as possible, but if absolutely necessary, define global data. Clearly define the meaning or range of a value.>

## Define

#define CH\_PACKET\_NUM 10

#define CH\_BANNER\_NUM 10

## Type definition

typedef struct

{

UINT08 numOfVCh; /\* 0x00: Number of Virtual Channels \*/

UINT08 PSIP\_mode; /\* 0x01: PSIP mode for this p-Channel \*/

UINT16 TS\_ID; /\* 0x02: Transport Stream ID \*/

CM\_VCH\_INFO\_T \*pVchInfo; /\* 0x04: Pointer to virtual channels \*/

} CH\_PAC\_T;

## Data definition

CH\_PAC\_ARRAY\_T gChPacket[CH\_PACKET\_NUM];

CH\_BAN\_ARRAY\_T gChBanner[CH\_BANNER\_NUM];

# Design Alternatives

<n ASPICE 3.0, SDD, like SAD, requires a design alternative. In the case of a new design, it is evaluated in terms of interoperability, interaction, criticality, technical complexity, risks, testability, etc., and in the case of a change design, the difference from the previous design (previous design is alternative#1, later design is alternative#2) from this point of view evaluated as In case of reuse, it is evaluated whether the component is suitable for the purpose.>

## Title A

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **QA Type** | **Quality Attribute Scenario** | | | | | | | | | | **Priority** | |
| QA type | Describe the scenario (SRS\_ID) | | | | | | | | | | High | |
| Business Goal | Describe the business goals that are affected by the scenario | | | | | | | | | | | |
| **Source** | **Stimulus** | | **Artifacts** | | | **Environment** | | **Response** | | | **Measure** | |
| **source of stimulation** | **stimulus that causes a response in the system** | | System components that respond to stimuli | | | **The environment in which the stimulus that causes the system to respond occurs** | | Response of system components to stimuli | | | Response metric | |
| Architectural Decisions and Reasoning | List design alternatives.  Alternative #1: …  Alternative #2: …  Choice : #1  Rationale: Describes the design decisions related to the scenario that affect the quality attribute response. Discuss the rationale for making the design decision. | | | | | | | | | | | |
| Allocated SW Component | SW components related to the design | | | | | | | | | | | |
| Trade-off  Analysis | Alternative | time to market | | Cost | Risk | | System qualities | | Reuse | Use proven technologies | | Performance |
| #1 | **Yes** | | **Yes** | No | | NO | | **Yes** | Unknown | | No |
| #2 | No | | No | **Yes** | | **Yes** | | Unknown | Yes | | Unknown |

< Draw a picture of the selected design alternative (mandatory) and alternative design alternative (optional).>

Figure 7.1 alternative Design #1