LGE Internal Use Only

LGE VS [OEM Name] [Project Name]

**Software Detailed Design**

[SW Component Name]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **About This Template**   * Template Name: LGE\_VS\_SWDC\_T01\_Software Detailed Design(SDD) * Management Department: VS SW Process Team * Revision History  |  |  |  |  |  | | --- | --- | --- | --- | --- | | Version | Date | Comment | Author | Approver | | 2.5 | 2016-04-18 | Initial Release | VC Smart SW Process Team | VS SW Process Team Leader | | 2.6 | 2018-01-30 | Added “Data Design” chapter. | VC Smart SW Process Team | VS SW Process Team Leader | | 2.7 | 2019-02-12 | Update due to annual organization restructuring (VC --> VS) | SH.Lee  (VS SW Process Team) | VS SW Process Team Leader | | 2.9 | 2021-08-25 | (\* actually updated in ’19.03.29 by TJ.Park)  - Updated ‘Read me first’, ‘About this template’  - Document structure | Soo.Yoon  (VS SW Process Unit) | SW Process Unit Leader | | 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) | |

|  |
| --- |
| **Read me first**   * 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. * Since the content composition of this template may not be 100% suitable for all projects, it should be tailored according to the characteristics of the project. * The definitions of SW Element (or SW Module), SW Component, and SW Unit used in this document are as follows.   - SW Element: Among the System Elements defined in SysAD, Elements implemented in SW have a hierarchical level in the SW architecture.  ASPICE defines SW Element as follows.  “The software is decomposed into elements of the software architecture across appropriate hierarchical levels down to the software components (the lowest level elements of the software architecture).”  - SW Component: As the lowest level element defined in SAD (SW Architecture Design), it becomes the unit of SDD (SW Detailed Design) document.  - SW Unit: The smallest unit constituting SW (e.g. Function in C language)   * Following documents are referenced.   + CMMISM for Systems Engineering/Software Engineering, Version 1.02   + IEEE Recommended Practice for Software Design Descriptions   + Automotive SPICE Process Assessment / Reference Model 3.0   + UML 2 AND THE UNIFIED PROCESS Second Edition   + LGE VR Checklist   + LGE SW quality audit check list   + Template and guideline previously released by LGE VS Smart division. * 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 !! You should not miss these.**   1. **Ground Rule**  * Any kinds of design tools can be used for SDD. This template uses EA for diagrams. * SDD is documented by component unit. Depending on the size and characterisrics of the project, one document file can include all the components or separate document files can cover each component individually. * Acronyms for the related SW Engineering documents.   + SyRS: System Requirements Specifications (Requirement ID: SyRS ID)   + SyAD: System Architectural Design   + SRS: SW Requirements Specifications (Requirement ID: SRS ID)   + SAD: SW Architectural Design   + SDD: SW Detailed Design  1. **Document Format**   These are general rules for document format, which are applied to this template.   * Figures and tables have captions to find and check them easily. * Create Caption: [References] > [Insert Caption] > [Label] > Select [Figure] or [Table]. * Put the figure caption under the figure. Figure 1 Title, Figure 2 Title,.. * Put the table caption above the table. Table 1 Title, Table 2 Title,..  1. **Document Versioning, File Naming Rule**   Follow the “Work product Naming Rule” from “LGE\_Smart\_Guideline\_Configuration Management”. |

About This Document

Document Information

|  |  |
| --- | --- |
| Issuing authority | VS OOO Team |
| Configuration ID | Configuration Item ID of CMBook |
| **Status of document** | In Progress / Approved / Released |

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| The document history is organized in the order that the most recent history is at the top and the first history is at the bottom. | | | | |
| Version | Date | Comment | Author | Approver |
| #.# | YYYY-MM-DD | OOOOOOOOOOOOOOO | OOO | OOO |
| 1.1 | 2016-02-20 | Modified xxx notation of C&C Diagram in 2.2 C&C View. | Gildong Hong | Sam Shin |
| 1.0 | 2016-01-20 | Initial Release | Gildong Hong | Sam Shin |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

|  |
| --- |
| The table of contents should be generated automatically. If you right-click on the table below and select [Update Field] menu, the changed table of Contents will be automatically reflected |

[1 Introduction 8](#_Toc80799243)

[1.1 Purpose 8](#_Toc80799244)

[1.2 Scope 8](#_Toc80799245)

[1.3 Audience 8](#_Toc80799246)

[1.4 Conventions 8](#_Toc80799247)

[1.5 Acronyms / Glossary 9](#_Toc80799248)

[1.6 Related Documents 9](#_Toc80799249)

[2 Overview 10](#_Toc80799250)

[3 [Component 1] Component 12](#_Toc80799251)

[3.1 Data Design 12](#_Toc80799252)

[3.2 External Design 13](#_Toc80799253)

[3.3 Internal Design 15](#_Toc80799254)

[3.3.1 Static Design 15](#_Toc80799255)

[3.3.2 Dynamic Design 17](#_Toc80799256)

[3.3.2.1 State Design 17](#_Toc80799257)

[3.3.2.2 Interaction Design 18](#_Toc80799258)

[3.3.2.2.1 [Interface 1] interface 19](#_Toc80799259)

[3.3.2.2.2 [Interface 2] interface 20](#_Toc80799260)

[3.3.3 Algorithm Design 20](#_Toc80799261)

[3.3.3.1 [Class 1] class 21](#_Toc80799262)

[3.3.3.1.1 [Method 1-1] method 22](#_Toc80799263)

[3.3.3.1.2 [Method 1-2] method 23](#_Toc80799264)

[3.3.3.2 [Class 2] class 23](#_Toc80799265)

[3.3.3.2.1 [Method 2-1] method 23](#_Toc80799266)

[3.3.3.2.2 [Method 2-2] method 23](#_Toc80799267)

[4 [Component 2] Component 24](#_Toc80799268)

[4.1 Data Design 24](#_Toc80799269)

[4.2 External Design 24](#_Toc80799270)

[4.3 Internal Design 24](#_Toc80799271)

[4.3.1 Static Design 24](#_Toc80799272)

[4.3.2 Dynamic Design 24](#_Toc80799273)

[4.3.2.1 State Design 24](#_Toc80799274)

[4.3.2.2 Interaction Design 24](#_Toc80799275)

[4.3.2.2.1 [Interface 3] interface 24](#_Toc80799276)

[4.3.2.2.1 [Interface 4] interface 24](#_Toc80799277)

[4.3.3 Algorithm Design 24](#_Toc80799278)

[4.3.3.1 [Class 3] class 24](#_Toc80799279)

[4.3.3.1.1 [Method 3-1] method 24](#_Toc80799280)

[4.3.3.1.2 [Method 3-2] method 24](#_Toc80799281)

[4.3.3.2 [Class 4] class 24](#_Toc80799282)

[4.3.3.2.1 [Method 4-1] method 24](#_Toc80799283)

[4.3.3.2.2 [Method 4-2] method 24](#_Toc80799284)

Figures

|  |
| --- |
| If you right-click on the table below and select [Update Field] menu, the changed table of Figures will be automatically reflected. |

[Figure 1 Software Architectural Design 10](#_Toc80799285)

[Figure 2 External Interaction Design for [Component 1] component 13](#_Toc80799286)

[Figure 3 External Interaction Design for [Component 1] component 14](#_Toc80799287)

[Figure 4 Class diagram for [Component 1] component 16](#_Toc80799288)

[Figure 5 State diagram 17](#_Toc80799289)

[Figure 6 Sequence diagram for [Interface 1] interface 19](#_Toc80799290)

[Figure 7 Detailed class diagram with attribute and method 21](#_Toc80799291)

[Figure 8 [Method 1-1] method flow chart 22](#_Toc80799292)

[Figure 9 [Method 1-2] method flow chart 23](#_Toc80799293)

**Tables**

|  |
| --- |
| If you right-click on the table below and select [Update Field] menu, the changed table of Figures will be automatically reflected. |

[Table 1 SW Component Descriptions 11](#_Toc80799294)

[Table 2 Data Design 12](#_Toc80799295)

[Table 3 Classes that [Component 1] component consists of 16](#_Toc80799296)

[Table 4 Description of each state 18](#_Toc80799297)

[Table 5 State transitions 18](#_Toc80799298)

[Table 6 Interface design for [Component 1] 19](#_Toc80799299)

[Table 7 Method specifications 21](#_Toc80799300)

# Introduction

## Purpose

|  |
| --- |
| Describe the use of this document. |

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

## Scope

|  |
| --- |
| Describe the scope of contents covered in this document. |

This document identifies the class consisting of each component from SAD, and describes the behaviors of those classes to accomplish the requirements upon them.

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

## Audience

|  |
| --- |
| Describe the target audiences (main stakeholders) of this document. |

The target audience of this document is:

* Software architect who will evaluate the design of the software
* Component developer who will implement the design in actual code
* Native application developers who need to interact with AppManager
* XXX project participants who want to understand the low level design of the AppManager
* Test engineers who verify this component

## Conventions

|  |
| --- |
| Indicate what you want to emphasize or what is helpful. |

NOTE

useful notes

## Acronyms / Glossary

|  |
| --- |
| Describe the abbreviations and explanations used in this document.  The table should be written in alphabetical order of abbreviations / terms. In addition to the acronym, a description is required. |

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

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

## Related Documents

|  |
| --- |
| List the referenced documents when writing this document.  Describe the titles of the referenced documents. If there are multiple versions of the document, also describe the version name of the referenced document. |

Documents related to this document include:

* IHU\_MAIN SRS (Software Requirement Specifications) v1.0
* IHU\_MAIN SAD (Software Architectural Design) v1.0

# Overview

|  |
| --- |
| This chapter explains the target of detailed design. Within the SAD’s Module View below, red boxes represent components. SW detailed design is documented by this component unit. Depending on the size and characteristics of the project, one document file can include all the components assigning one chapter to one component or separate document files can cover each component individually. |

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

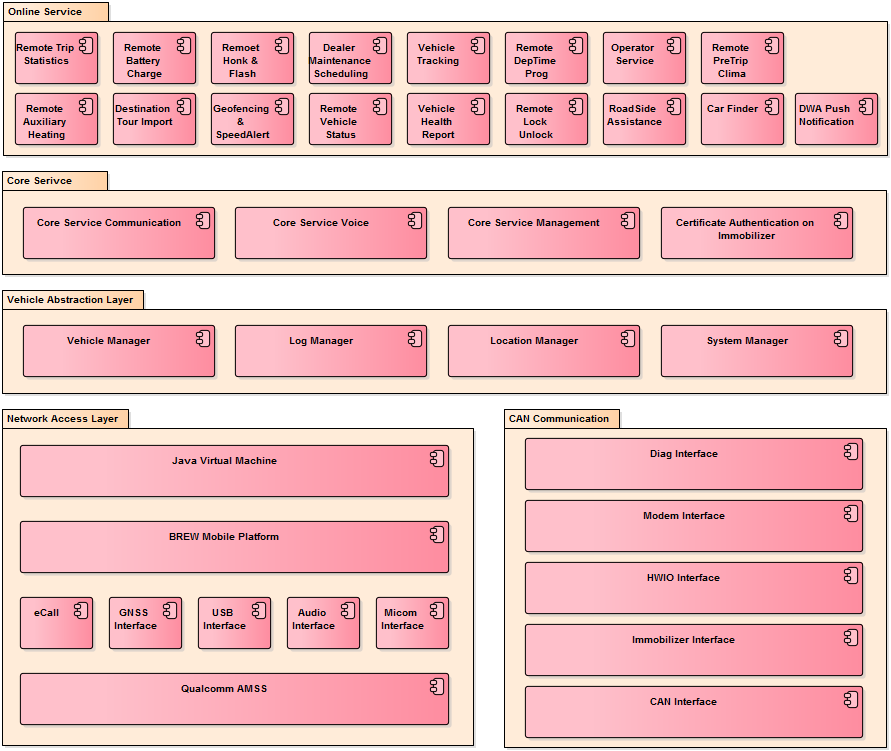


Figure 1 Software Architectural Design

|  |
| --- |
| This chapter explains above components and their SW requiremens briefly.  SAD and SDD use same SW Component ID maps. SW Component ID secures the traceability among SRS-SAD-SDD.  Both functional and non-functional requirements should be fully covered at this table. For SRS ID, be sure to have all the non-functional requirements as well as functional requirements. |

|  |  |  |  |
| --- | --- | --- | --- |
| **SW Component ID** | **SW Component Name** | **Description** | **SRS ID** |
| SW Component ID | SW Component name | Component Responsibility | Requirements ID associated with Component |

Below table describes the software components of OOO.

Table 1 SW Component Descriptions

|  |  |  |  |
| --- | --- | --- | --- |
| SW Component ID | SW Component Name | Description | SRS ID |
| XXX\_SWC-001 | Remote Trip Statistics | Vehicle’s trip information (mileage, average consumption) | SRS-XXX\_100 |
| XXX\_SWC-002 | Remote Battery Charge | To control e-vehicle’s battery charging function remotely | SRS-XXX\_101 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# [Component 1] Component

|  |
| --- |
| Every component that is identified at SAD should go to the detailed design phase from this chapter. One component occupies one chapter. |

## Data Design

|  |
| --- |
| The variables needed to process input / output data passed through the component's interface or internally to process and store data should be defined. In general, a global variable to be used in a component should be defined here.  Type: Defines the variable type. For user-defined types, the details should be added to the Description.  Name: Define the variable name according to the naming rule.  Description: Describe the data stored in the variable, and a range of the data value.  Remark: Describe if there are restrictions on using variables. (Applied model variant etc.) |

Table 2 Data Design

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Name** | **Description** | **Remark** |
| Structure | SwcBluVariable | Store status data related BLU  typedef struct \_swc\_blu {  uint8 ui8VariantType; // range: 1~5  uint16 ui16Lcd1PwmDutyCycle; // range: 0~32768  boolean bLcd1BluEnStat; // BLU\_EN Pin Stat  boolean bLcd1BluCanEnCtrl; // DISP\_C\_BackLgt\_Rq & DISP\_IC\_Rq\_Pr2 Value  boolean bLcd1BluDeserialEnCtrl; // Lock Status Value  boolean bLcd1BluFpgaEnCtrl; // Safety Status Value  uint16 ui16Lcd2PwmDutyCycle; // range: 0~32768  …  …  } SwcBluVariable; |  |

## External Design

|  |
| --- |
| Relationship between the target component and other components that directly interact with it is described. Sequence Diagram and its description are introduced here.  (Example 1) Within the following example, blue box represents the target component (Component 1) and gray box represents outer components. |

When the IHU starts up, the Application Launcher decides the boot sequence of Applications. The sequence may depend on LUC, country variant, car variant, etc. After deciding the boot sequence, the App Launcher starts each Application accordingly.



Figure 2 External Interaction Design for [Component 1] component

* Step 1.0: MICOM sends the boot context data during the start up. It may include the LUC, country info, car variants, etc. Depending those information, the boot sequence may vary.
* Step 1.1: DBus Manager notifies the Application Loader for the MMUS event. (**SRS ID OOO**)
* Step 1.2: …
* Step 1.3: …
* Step 1.4: …
* Step 1.5: …
* Step 1.6: …
* Step 1.7: …
* Step 1.8: …
* Step 1.9: …

|  |
| --- |
| (Example 2)  Within the following example, gray-colored area means the target component (Component 1) and other boxes represent outer components.  Depth of the interaction design can vary depending on each project’s circumstances and characteristics. Set the level or standard that fit each project and apply them. |

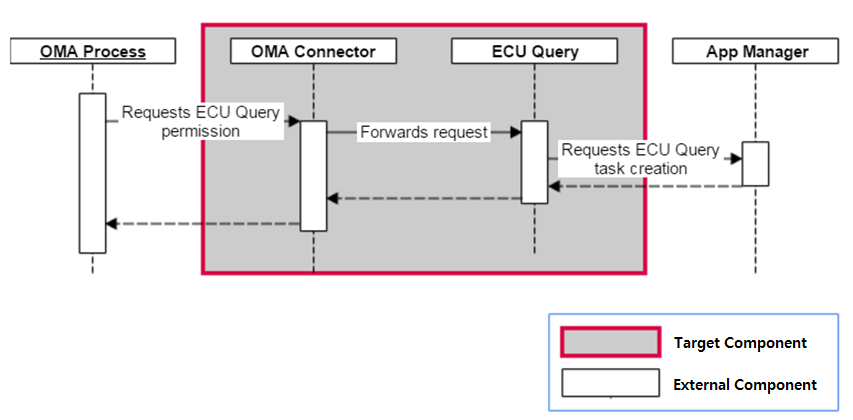


Figure 3 External Interaction Design for [Component 1] component

## Internal Design

|  |
| --- |
| This chapter covers the internal design of the target component. |

### Static Design

|  |
| --- |
| Identify the internal classes of the target component. Example of this template defines classes at Static Design. Algorithm Design identifies and designs the algorithm of the methods of those classes. Depending on the characteristics of project and component, scope and depth (public or private) of method identification/design vary.  Internal design of component uses Class Diagram. Other diagrams can be applied also, depending on the characteristics of each component.   - C++/Java: Identify the classes and member functions consisting the target component - C: Identify the files and functions consisting the target 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 class diagram for the [Component 1] component is shown below:



Figure 4 Class diagram for [Component 1] component

|  |
| --- |
| Use a table below to expain the classes of Class Diagram. . |

|  |  |
| --- | --- |
| **Class** | **Descriptions** |
| Class Name | Class’ Responsibility |

The classes identified above are described in the below table:

Table 3 Classes that [Component 1] component consists of

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

As shown

### Dynamic Design

|  |
| --- |
| The internal behavior of the target component should be designed. Dynamic design related to this Component can be described through State Diagram or Sequence Diagram. A diagram that can effectively represent the Component you want to implement should be selected and described. |

#### State Design

|  |
| --- |
| Explain the state transition of internal classes of target component. Following is an example of using State Diagram. |

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



Figure 5 State diagram

|  |
| --- |
| The following table describes the states identified in the above state diagram. |

|  |  |
| --- | --- |
| **States** | **Descriptions** |
| State type | Meaning of each state |

The descriptions for each state are as follows:

Table 4 Description of each state

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

|  |
| --- |
| Create a table of state transitions. |

|  |  |  |  |
| --- | --- | --- | --- |
| Current State | Event/Action | Next State | Descriptions |
| Current State | Input Event/Action | State to be transitioned | State Description of the transition |

The state transitions for the state diagram are as follows:

Table 5 State transitions

|  |  |  |  |
| --- | --- | --- | --- |
| 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. |
| … | … | … | … |
| … | … | … | … |
| … | … | … | … |
| … | … | … | … |

#### Interaction Design

|  |
| --- |
| Design how the identified classes interact in Static Design. In Static Design, the interaction should be designed in the unit of the object identified and defined as the Instance.  It is designed in detail based on the Interface List identified in the SAD, so the SAD's Interface List table is created in the same manner. Detailed design should be done for all interfaces, and if the interface is changed in SDD, related contents of SAD should also be changed. |

|  |  |  |  |
| --- | --- | --- | --- |
| SW Component | Interface Name | Type | Parameters |
| Component Name | Interface Name | Use one of the values below. (Add if necessary)  call : Local function call  dbus : Communication using dbus | - Define each Input / Output  - Parameters range, scale, design criteria, etc. |

Table 6 Interface design for [Component 1]

|  |  |  |  |
| --- | --- | --- | --- |
| SW Component | Interface Name | Type | Parameters |
| [Component 1] | [Interface 1] | call | n/a |
| [Interface 2] | call | In:   * [Class 1] : command to be added * bExecute: whether to execute the command immediately or not. If set to true, the command is executed immediately. |

#### [Interface 1] interface

|  |
| --- |
| Describe how the identified classes interact in the form of a Sequence Diagram. |



Figure 6 Sequence diagram for [Interface 1] interface

- Step 1.0: The [Interface 1]() method of the CommandInvoker is called by …  
- Step 1.1: …

- Step 1.2: …

|  |
| --- |
| Precautions when expressing interactions between classes as a sequence diagram   1. The start of the sequence must match the interface call. In the example above, the interface name [Interface 1] matches Step 1.0. 2. If you need to interact with classes outside of "AppManager”, you can also distinguish the class box by displaying it in a color other than blue. (No external class in the example above) 3. If there is repetition of the same sequence, uses a loop.. |

#### [Interface 2] interface

### Algorithm Design

|  |
| --- |
| Once the interactions between classes have been designed, it should be also described what methods the classes should have in them. The class of the component, the attributes and methods belonging to the internal class should be described..  The struct / union / enum in C language should be written in accordance with the class, and the function in C language should be written in accordance with the 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 7 Detailed class diagram with attribute and method

#### [Class 1] class

|  |
| --- |
| The methods of each internal class of Component should be explained using the following table.. |

|  |  |  |
| --- | --- | --- |
| **Method** | **Descriptions** | **Input/Output** |
| Method Name | Brief description of the method | Explanation of Input and Output |

Table 7 Method specifications

|  |  |  |
| --- | --- | --- |
| **Method** | **Description** | **Inputs/Outputs** |
| 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 |
| [Method 1-1] | … | … |
| [Method 1-2] | … | … |

#### [Method 1-1] method

|  |
| --- |
| Detailed design is required for all the methods mentioned in the table above. However there can be a deviation according to OEM.  You can use a Flowchart or Pseudocode if you need an algorithm representation to describe the detailed design inside a particular method. In the example below, we use flowchart to describe the internal logic. It can also be described as a state chart diagram tomanage state, or as a call graph for complex function call relationships. |

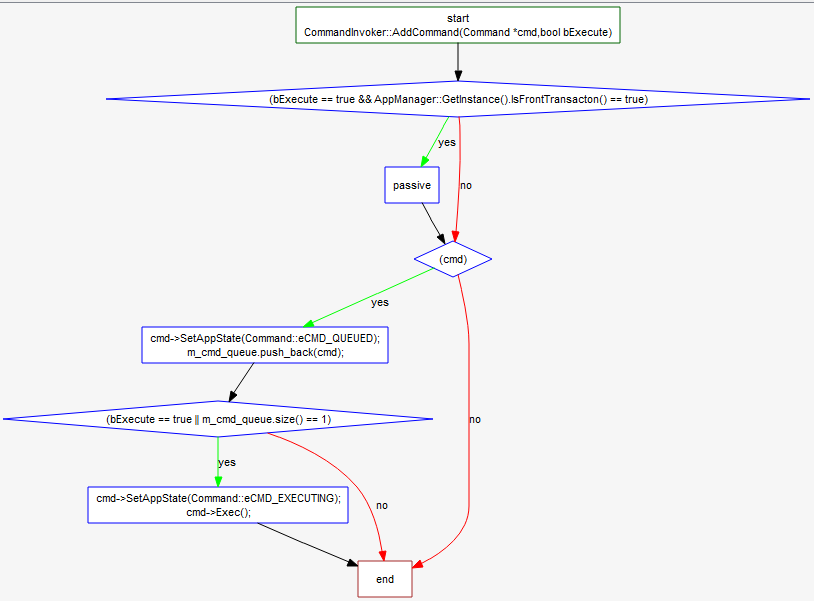


Figure 8 [Method 1-1] method flow chart

#### [Method 1-2] method

순서도

Figure 9 [Method 1-2] method flow chart

#### [Class 2] class

#### [Method 2-1] method

#### [Method 2-2] method

# [Component 2] Component

## Data Design

## External Design

## Internal Design

### Static Design

### Dynamic Design

#### State Design

#### Interaction Design

#### [Interface 3] interface

#### [Interface 4] interface

### Algorithm Design

#### [Class 3] class

#### [Method 3-1] method

#### [Method 3-2] method

#### [Class 4] class

#### [Method 4-1] method

#### [Method 4-2] method