Observant

INNOVATIONS

Virtual Surface

|  |  |
| --- | --- |
| Virtual Surface | |
|  | |
|  | |
| Overview |  |
| Introduction |  |
| MVC | 3.1 Design Pattern  3.2 JW01’s Overview  3.3 Trygve Reenskaug’s Formulation  3.4 VS and MVC |
| System | 4.1 Dependency Injection  4.2 Configuration  4.3 Context  4.3.1 BOOL(ean) properties  4.3.2 CHAR(acter) properties  4.3.3 DOUBLE properties  4.3.4 DWORD properties  4.3.5 FLOAT properties  4.5 Framework  4.6 Versioning |
| Application | 5.1 Interface  5.2 Configuration  5.3 Context  5.3.1 System and MVC  5.3.2 Configuration  5.3.3 Data  5.3.4 Graphic Interface  5.3.5 Element Context and Display Lists  5.3.6 Graphic Context  5.3.7 Object Configuration  5.4 Framework  5.5 API  5.6 View Callback  5.7 Development  5.8 Versioning |
| Graphics | 6.1 Context  6.2 Framework  6.3 API  6.4 Display Callback  6.5 Control Callback |
| Appendices | 1. Trygve Reenskaug – MVC Formulation 2. JW01 - MVC is all about separation of concerns 3. Hierarchical Description Language (HDL) Files 4. Application Description File 5. System Description File 6. View Description File 7. Versioning 8. C++ Conventions |
| Illustrations | 1. Virtual Surface Component Diagram 2. Class Diagram 3. System Class Diagram 4. Application (MVC) Class and (Empty) Diagrams 5. Application Interface Include Diagram 6. Application Process Diagram 7. Model Process Diagram 8. View Class Diagram 9. Control Process Diagram 10. Graphics Class Diagram 11. Element Context Class Diagram 12. Graphics Private Implementations and Components |
| Notes | 1. Please note that the all code snippets are given to illustrate process and conventions, and are not complete implementations. |

|  |  |
| --- | --- |
| 1. Overview | Virtual Surface (VS) provides a developer with a Managed Application Development Environment (MADE). This is comprised of an application development framework SDK, a Managed Application API, and a Graphics API.  VS is designed to rapidly develop graphical applications, in which all of the System and Graphics, and much of an applications "housekeeping" are managed by VS. This results in the minimum of new application design, coding and documentation being required.  VS takes its name from a computer graphics virtual 2D or 3D plane or shaped mesh primitive, created either functionally or procedurally, or by using a graphics application such as a 3D editing system.  Recently a non-graphical console based version of VS, VSX, has been created. This shares much of the same structure and code base as VS. Specific reference to VSX will indicated by prefixing paragraphs with "**VSX**". |

|  |  |
| --- | --- |
| 2. Introduction | Even the simplest graphic primitives, whether they are 2D or 3D, require a complex set of functions and programming steps to initialise, create and then draw them in the window of an application. For animation and interaction this must also all happen within a real-time framework that can handle a multitude of different system events such as a window being resized, minimised, maximised and closed; a mouse location being updated; and a key on a keyboard being pressed down and then released.  This complexity is substantially increased by the requirements of a graphics application to draw graphics primitives using a special hardware device; the Graphics Processing Unit (GPU). This device must, like the application, be initialised and a context created with properties, such as pixel width and height of an application, and the graphical capabilities of each PC. There are many system events, such as the window of an application being resized, when this context must be updated, during which time it's context described as being "lost", and when re-initialised it's context is described as "regained".  The real-time system framework, its management and interaction with an application, and the graphics API’s can be the same for every application. There are also many parts of (once developed) applications which are the same, or which are reusable, between different applications.  VS is comprised of three functionally separate parts; System, Application and Graphics. The design pattern for each of these is based on the Model, View Controller (MVC) Design Pattern, all of which are based on a real-time framework comprised of five principle functional parts; setup, setup graphics (invoked at run time and when context is "regained"), display, cleanup graphics (invoked when context is "lost") and cleanup.  The System and Graphics parts are provided, and as the View component of the Application Implementation MVC Design Pattern (see below) can be functionally the same for every application it is encapsulated within an Application API, and so in a VS based application it is only the Control and Model components that need to be developed. As mentioned above much of these can be reused or repurposed from examples or other applications.  This document describes the MVC Design Pattern, then the three functionally separate parts of VS; System, Application and Graphics. The Appendices and Diagrams are provided in a separate document so that they can be read and viewed in conjunction with this document. |

|  |  |
| --- | --- |
| 3. MVC | |
|  | |
| 3.1 Design Patterns | |
|  |  |
|  | A design pattern is a general solution to a common software problem. The term was made popular by the book Design Patterns: Elements of Reusable Object-Orientated Software.  The VS architecture uses many different design patterns and techniques, including the Pimpl idiom, Singleton Method, Monotone Method, Dependency Injection Method, and the Proxy, Adapter, Facade, Observer and as mentioned above various MVC Design Patterns.  The most important – in that it defines the whole shape of the VS architecture is the MVC Design Pattern. |
|  | https://fentyoktorina.files.wordpress.com/2011/05/mvc.jpg |
|  | Fig x: MVC diagram. |
|  | As such, before describing it’s usage in VS, this document provides two introductory texts; JW01’s Overview from Programmers Stack Exchange and Trygve Mikkjel Heyerdahl Reenskaug’s Formulation. |
|  |  |
| 3.1 Design Patterns | |
|  |  |
|  |  |
|  |  |