Mark* V1e and Mark V1eS
Virtual Controllers
User Guide

Nov 2020
These instructions do not purport to cover all details or variations in equipment, nor to provide for every possible contingency to be met during installation, operation, and maintenance. The information is supplied for informational purposes only, and GE makes no warranty as to the accuracy of the information included herein. Changes, modifications, and/or improvements to equipment and specifications are made periodically and these changes may or may not be reflected herein. It is understood that GE may make changes, modifications, or improvements to the equipment referenced herein or to the document itself at any time. This document is intended for trained personnel familiar with the GE products referenced herein.

GE may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not provide any license whatsoever to any of these patents.

Public Information – This document contains non-sensitive information approved for public disclosure.

GE provides the following document and the information included therein as is and without warranty of any kind, expressed or implied, including but not limited to any implied statutory warranty of merchantability or fitness for particular purpose.

For further assistance or technical information, contact the nearest GE Sales or Service Office, or an authorized GE Sales Representative.

Revised: Nov 2020
Issued: Sept 2009


* Indicates a trademark of General Electric Company and/or its subsidiaries. All other trademarks are the property of their respective owners.

We would appreciate your feedback about our documentation. Please send comments or suggestions to controls.doc@ge.com
## Document Updates

<table>
<thead>
<tr>
<th>Rev</th>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td><strong>Virtual Controller Features</strong></td>
<td>Added NVRAM support</td>
</tr>
<tr>
<td></td>
<td><strong>Product and System Limitations</strong></td>
<td>Removed “it does not support NVRAM storage”</td>
</tr>
<tr>
<td></td>
<td><strong>NVRAM Save and Restore</strong></td>
<td>New section</td>
</tr>
<tr>
<td>K</td>
<td><strong>Header and Library Files</strong></td>
<td>Added support for consolidated 32-bit and 64-bit products</td>
</tr>
<tr>
<td></td>
<td><strong>Mark VIe Multi-core Support</strong></td>
<td>New section to support Multi-core products</td>
</tr>
<tr>
<td>J</td>
<td><strong>Simulation System Components</strong></td>
<td>Added content providing alarm system requirements concerning the supported number of new alarms that can be processed but explained that the virtual controller is not limited to these requirements</td>
</tr>
<tr>
<td></td>
<td><strong>Configure Microsoft Loopback Adapter</strong></td>
<td>Added the procedure to configure the Loopback Adapter to add an additional virtual controller to the simulation</td>
</tr>
</tbody>
</table>
## Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>EGD</td>
<td>Ethernet Global Data</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-Machine Interface, usually a computer with CIMPLICITY software</td>
</tr>
<tr>
<td>PDH</td>
<td>Plant Data Highway</td>
</tr>
<tr>
<td>SDI</td>
<td>System Data Interface</td>
</tr>
<tr>
<td>TMR</td>
<td>Triple modular redundant, uses three sets of controllers and I/O</td>
</tr>
<tr>
<td>UDH</td>
<td>Unit Data Highway</td>
</tr>
</tbody>
</table>
Safety Symbol Legend

**Warning**
Indicates a procedure or condition that, if not strictly observed, could result in personal injury or death.

---

**Caution**
Indicates a procedure or condition that, if not strictly observed, could result in damage to or destruction of equipment.

---

**Attention**
Indicates a procedure or condition that should be strictly followed to improve these applications.
1 Introduction

This document describes the virtual controller product and Application Programming Interface (API) installed in a larger simulation platform. *Virtual controller* is a generic term used to refer to Mark* VIe and Mark VIeS Virtual Controller products. The Mark VIe and Mark VIeS Virtual Controllers are part of the ControlST* DVD.

The virtual controller simulates Mark VIe and VIeS control system responses in a software environment. Virtual controller software and Human-Machine Interfaces (HMI) are designed to be integrated into a full-scope, high-fidelity simulator to provide training to both operators and maintenance personnel. The virtual controller connectivity with the ToolboxST* application allows live control code data to be viewed. For a complete simulation, one or several HMIs communicate with the virtual controller application. The HMIs include the CIMPLICITY* application and project-specific graphic files, which allow duplication of a plant HMI and eliminates translation errors.

**Note** Virtual controller and HMI software are enabled using a USB license key that is provided with the software package. To prevent disrupting the simulation, the virtual controller stops within thirty minutes after the licence key is not detected (or has been removed).

<table>
<thead>
<tr>
<th>Supported Hardware and Operating Systems (OS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OS</strong></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>CPU</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>HD</td>
</tr>
<tr>
<td>Monitor</td>
</tr>
<tr>
<td>Drive</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>USB</td>
</tr>
</tbody>
</table>

**Note** Verify that all ControlST applications required to be installed on the same computer are supported on the selected OS, especially a 64-bit OS.
1.1 Virtual Controller Features

- Uses the same application code as an actual controller.
- Runs the same Control Sequence Program as an actual controller.
- As with an actual controller, uses Ethernet Global Data (EGD) and System Data Interface (SDI) for block scheduling and communication protocols.
- Any major changes to application code cannot invalidate previous collections of controller state information.
- Virtual controller process is repeatable and deterministic in nature (for example, consistent response) provided the furnished simulation model is repeatable and deterministic.
- All time stamps implemented by the simulator are driven by simulation time, not host target time.
- Provides an API to directly access the variable space, so that an external software model can drive I/O.
- All variables are identified symbolically. They are available for inspection and modification to external software.
- Supports Run, Stop, Pause, Step, Set/Clear Break, Watch, Save/Restore IC, Overrides, and Backtrack commands, as well as utility functions such as simulation time management and performance metrics.
- Can be run synchronously (the Simulation Executive sends a command to the virtual controller to step one or more frames) or asynchronously (virtual controller runs independently of the Simulation Executive at the frequency determined by the controller configuration using simulation time).
- EGD and SDI communication remains active when the virtual controller is paused. EGD production and reception continue when the application code is paused to prevent the HMI screens displaying blank. However, EGD commands (from either a CIMPLICITY project or other EGD command clients) are not performed when the virtual controller is paused.
- Supports process alarm detection and messaging.
- Unexpected disturbances in the control system do not occur due to any instructor actions.
- Supports two methods of implementing Fast Time and Slow Time in the virtual controller: variable step rate and variable step size. When it is running synchronously, the Simulation Executive can implement fast time and slow time by varying the frequency of the step commands to the virtual controller or by varying the number of frames to run for each step command.
- The process alarm summary, historical process data, or any other data dynamically generated is not suspended while the simulation system is paused.
- Suspends Trender operation when the application is paused, then continues without losing any data.
- Captures alarm queue status with the Save command; restores the alarm queue with the Restore command.
- Triggers Sequence of Events (SOE) using an API to simulate contact transitions at 1 ms intervals without affecting the application state.
- Supports dynamic binding of EGD, which allows updates to the consumed EGD configuration dynamically when a corresponding producer changes (Mark VIe Virtual Controller only).
- Supports the Action Capture and Replay (ACR) feature, which allows recording and replaying of commands from a CIMPLICITY project, the ToolboxST and WorkstationST* applications, LiveView, and Trender.
- The FOUNDATION Fieldbus (FF) macrocycle delay can be simulated for FF device-related I/O in a Mark VIe configuration.
- Provides controller-to-controller communication in the virtual controller.
- Supports NVRAM storage for saved variables, totalizers, and forced variables. (Not supported in all products, refer to the section NVRAM Save and Restore.)
1.2 Typical Simulator Components

A typical full scope power plant simulator consists of four major components:

- Simulation Executive
- Plant model (process dynamics)
- Plant virtual controller – Mark VIe and Mark VIeS Virtual Controllers (simulation of one or multiple controllers)
- Operator interface – HMI

Additionally, the ToolboxST application is used in the actual plant for programming and troubleshooting the virtual controller. The following illustrates the typical division of responsibility between GE and the simulator vendor for a project where the virtual controller is integrated into a larger plant simulator.

<table>
<thead>
<tr>
<th>Item</th>
<th>Simulator Vendor</th>
<th>GE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation Executive</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Plant Model</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CIMPLICITY HMI</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Mark VIe and Mark VIeS Virtual Controllers</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>ToolboxST application</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>WorkstationST application</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
2 System Architecture

This chapter defines the architecture of the simulation system, including GE-supplied components and typical system layout. It also discusses the limitations of the virtual controller and requirements of a Simulation Executive into which the virtual controller is integrated.

2.1 Simulation System Components

The virtual controller consists of four .exe files among multiple libraries. For each new controller project opened and started, these four processes simulate the new controller.

GESimApp.exe is the heart of the virtual controller that runs the application code.

GESimSdi.exe provides an interface with the ToolboxST application.

GESimEgd.exe provides EGD services to the virtual controller.

GESimAlm.exe provides alarm services to the virtual controller.

The simapi.dll (also called the Simulator API) is a dynamic link library (.dll) that provides setup and communication of all external programs of the simulation environment. This .dll uses all application programming interfaces (APIs) and sets up I/O by specifying an identification number that is known between the plant model and the virtual controller. The data is made known to the virtual controller through the API. The .dll exports the necessary functions, which third-party software or a graphical user interface (GUI) can use to run and control the virtual controller.

The ToolboxST is a Windows-based software application used for configuring and maintaining the Mark VIe and VIeS control systems. The Trender can be used for control software analysis and provides sequencing diagnostics. Refer to the ToolboxST for Mark Control Platforms User Guide (GEH-6700).

The WorkstationST application allows you to download the ToolboxST configuration to a workstation to ensure that configured WorkstationST features start and stop accordingly. It also provides access to control and diagnostic information for WorkstationST functions, as well as historical Alarm and Recorder data.

The supported number of new alarms that can be processed in the Mark VIe and virtual Mark VIe alarm systems is as follows:

- A burst load of 400 alarm transitions are supported in a single frame without missing a transition in the internal queue.
- A constant load of 100 alarm transitions per second are supported without missing a transition in the internal queue.

However, since the virtual controller runs in Windows, which is not a real-time operating system, the frame rate is not exact. Taking advantage of this fact, the virtual controller is designed to increase the frame length to process the actual number of alarm transitions in a frame.
2.2 System Layout

In the following configuration, the virtual controller is installed directly on the simulation computer. One or multiple HMI(s) are provided to simulate the control room HMIs.

The simulation computer requires an Ethernet network interface card dedicated to the GE simulation HMI; this network corresponds to the plant’s Unit Data Highway (UDH). The simulation vendor provides, installs, and configures the network interface card according to GE requirements.

2.3 Product and System Limitations

The virtual controller runs only simplex, non-redundant software. It does not restart automatically. To restart, you must stop and start the virtual controller. Historical process data, command and event logging, or any other data dynamically generated as a result of running a simulation is not cleared or reset if the simulation is started, restarted, or restored to another state, as would be the case with an initial condition.

Mark VIe hardware diagnostic alarms are not functional because there is no real I/O hardware. There is no support for Distributed Control System (DCS) link testing, communication (for example, Modbus® and PROFIBUS), or verification. It does not support OPC UA.

---

**Attention**

The Controller Backup Option in the Mark VIe Component Editor must be set to Automatic and the Download Backup File check box must be selected in the Download Mark VIe Controller Wizard or the backup files will not be downloaded to the virtual controller. The backup files are required for the virtual controller to run properly.
2.4 Simulation Executive Requirements

The Simulation Executive is responsible for concurrently stopping, starting, running, and pausing the basic processes, including:

- Plant model (process dynamics)
- Virtual controller

*Note* These requirements are essential to the Simulation Executive.

The virtual controller uses an API where process variables (analog and digital) can be specified for read/write operations through shared memory. The Simulation Executive configures and populates the I/O to the virtual controller. It also provides a Graphical User Interface (GUI) to map plant model I/O to virtual controller I/O.

The virtual controller uses an API where the state of the simulator is saved to disk, including specification of a file name(s). The Simulation Executive manages these files, and assures that they are synchronized with simultaneously saved states of all other simulation processes. The Simulation Executive also manages any file-naming conventions, name collisions, access violations, ID numbers, and such. It saves all models and virtual controller states, and any other simulation settings required to reproduce the desired simulation state when a restore operation is invoked.
3 **Installation and Configuration**

Virtual controllers are typically installed on a computer (referred to as the simulation computer) that is different from a GE-supplied HMI. This simulation computer typically holds the modeling environment and the third-party Simulation Executive software. The other system components and the ToolboxST and WorkstationST applications are installed on the HMI and configured.

This chapter defines installation requirements for the virtual controller and discusses a simulation configuration where all the different system components are required to run on a single computer.

Before installing:

- If BlackICE service is running on the simulation computer, it needs to be stopped and the *startup type* set as disabled.
- Uninstall any previous versions of the virtual controller

The virtual controller installation, which includes the various virtual controller binary files and the API, is available on the ControlST DVD. The ToolboxST and WorkstationST applications must be installed on one computer, with a virtual controller installed on a separate computer. For instructions to install software from the ControlST DVD, refer to the instructions provided in the *ToolboxST for Mark Controls Platforms User Guide* (GEH-6700).

---

**Warning**

The virtual controller should never be installed on a computer that has network connectivity to the UDH of an actual control system. This causes the virtual controller to interfere with the actual Mark VIe control resulting in unintended control actions.

---

The virtual controller can also be used in a system layout where various simulation components (virtual controller, the ToolboxST or WorkstationST applications, or a CIMPLICITY project) are installed on the same computer. Such a system layout requires a specific additional configuration (refer to the section *Configure WorkstationST Service*). Also, this system layout allows you to configure the controller IP addresses to a *Microsoft Loopback Adapter*.
### 3.1 Configure ToolboxST System

#### Prerequisites

- Successfully installed the ControlST Software Suite V07.00.00C or later
- Successfully set up the PROFICY* License Key

➢➢➢ To configure a ToolboxST system

1. From the Start menu, select All Programs, GE ControlST, ToolboxST, and ToolboxST.

2. From the New System dialog box, enter the system name, browse to the location, and click OK. The System Editor displays. If needed, the GatewayIPAddress or SubnetMask can be modified (this is the network address for the ToolboxST application, not the Virtual Controller).

3. From the Tree View, right-click the system item and select Insert New, Controller, and Mark VIe Controller.

![Screenshot of ToolboxST configuration](image)

**Note** The Mark VIeS Controller can also be selected if planning to use that version of Virtual Controller.

4. From the Mark VIe Creation Wizard, enter a name and click Next and Finish.

5. From the toolbar, click the Save System icon.

6. From the Tree View or the Summary View, double-click the controller to display the Component Editor.

7. From the Component Editor, General tab, Property Editor, select the controller hardware Platform.

**Note** The Platform type must be the same as what is used for HardwareType when you Create the Virtual Controller.
8. Configure the IP address.
   a. Select Network Adapter 0.
   b. Enter the IP Address of the Virtual Controller. This is the same address used for the Loopback Adapter.

9. Create and configure application blocks as needed.
   a. From the Software tab, right-click Programs and select Add Program.
   b. Right-click Prog1 and select Add Task.
c. From the **Library View**, select and drag a block to the **Summary View**, then right-click the block and select **Edit Block Pins**.

d. From the **Edit Block Pin Connections** dialog box, use the drop-down lists to change values, then click **Close**.
e. From the **Summary View**, double-click the **OUTPUT** pin to display the **Connect Pin** dialog box.

![Connect Pin Dialog Box](image)

f. From the **Connect Pin** dialog box, select **Global Variable**, then select the **Create Variable** check box. Enter the variable name and click **OK**.

![Connect Pin Dialog Box](image)

g. From the **Tree View**, select **Variables**. From either the **Data Grid** or the **Property Editor**, configure the **EGD Page**.

![Tree View and Property Editor](image)

10. Perform a **Build** and verify that the device was saved.

11. From Windows Explorer, create a folder for the Mark Vle virtual controller files.

---

**Note** For more information on configuring the controller, refer to the **ToolboxST for Mark Controls Platforms User Guide** (GEH-6700) and any of the block library user guides that are available with your ControlST Software Suite installation.
3.1.1 Create Virtual Controller

Prerequisites

- The simulation computer has been successfully setup with an IP address and loopback adapter.
- The system and component have been correctly configured in the ToolboxST application.
- A Build has been successfully performed in the ToolboxST application.

Create the following directories under a root directory of the simulator computer, \( X:\Simulation\Controllers \) (\( X \) refers to the drive on the computer):

- \( X:\Simulation\Controllers \): holds all virtual controller folders. For example, if three virtual controllers with project names G1, G2 and G3 are being created, these virtual controller folders are created under \( X:\Simulation\Controllers \).
- \( X:\Master \): holds all application code files. For example, a .tcw file with three Mark VIe controllers G1, G2, and G3 could be stored in a project folder under \( X:\Master \).

➢ To create a virtual controller

1. From the Start menu, select All Programs, GE ControlST, Mark VIe Virtual Controller, and Mark VIe Virtual Controller.

   Note To add a Mark VIeS virtual controller, select Mark VIeS Virtual Controller.

2. From the Mark VIe Virtual Controller File menu, select Create to display the Create Project dialog box.

3. Provide the information to create the virtual controller:

   a. Enter a Project Name.

   b. Navigate to the Project Path (the folder created on the simulation computer that will store the Virtual Controller files).

   c. Select the Hardware Type. This must match the Platform configured in the ToolboxST application.

   d. Enter the IP Address. This must be the same IP Address used for the Loopback Adapter.
4. Configure the **Ext symbol file** by navigating to the folder on your computer where you saved your system (.tcw) file. From the controller’s device subfolder, open the **Output** folder, select the .syl file, and click **Open**. Click **OK** to save the Create Project.

5. Click the **Start Application** icon. When the Windows firewall message displays, click **Yes**.

---

**Attention**

If after clicking Start, a *Failed to bind to (IP address)* message displays, do not try to continue. There is something wrong with your network adapter configuration. Investigate the IP Address settings used for the Loopback Adapter, the ToolboxST Controller and WorkstationST Devices, and the virtual controller to verify that the addresses are the same.

6. From the ToolboxST **Component Editor**, perform a **Build** and **Download** the virtual controller.
7. From the virtual controller application toolbar, click the **Stop a running Simulation** icon.

8. Click the **Start Application** icon, then the **Run Application** icon.

   **Note**  When the simulator is started for the first time, the log window displays a message that runtime is not loaded.

   **Note**  If the WorkstationST Alarm Server has stopped receiving alarms, this may cause the virtual controller to not start properly. If this occurs, open a Task Manager and try closing all Windows processes related to the WorkstationST application. Then restart the virtual controller and the WorkstationST application.

9. From the Mark Vle Component Editor, click the **Go On/Offline** icon. Live values displayed in green indicate that the Mark Vle controller is communicating with the virtual Mark Vle control.
3.1.2 Configure WorkstationST Device

This configuration is required in a system layout where all simulation components run on the same computer. Without this configuration, the virtual controller cannot communicate with other simulation components. Every IP address, for which a virtual controller is connected, needs to be added to a WorkstationST Component Editor.

➢➢➢ To configure WorkstationST service

1. From the ToolboxST System Editor, right-click the system item and select Insert New and WorkstationST.
2. Provide a name, select all default settings, and click Finish. The new component displays in the Tree View.
3. Double-click the WorkstationST component to display the Component Editor.
4. From the General tab, select Network Adapter 0, then from the Property Editor, set the IP Address.

If the WorkstationST application is run on the same computer as the virtual controller, an exclude address must be set for the virtual controller.

From the General tab, select General, then from the Property Editor, select Exclude Addresses and enter an exclude IP address for the Mark VIe virtual controller.

[Image]

Note The same IP Address is used for the Loopback Adapter.

5. From the Tree View, select Features.
6. From the Data Grid, set OPC DA Server to True.

Note If the Historian is installed, set Historian to True as well.

7. Perform a Build and Download.
3.2 Simulation Computer Configuration

3.2.1 Configure Microsoft Loopback Adapter

➢➢ To configure a Microsoft Loopback Adapter

**Note** In Windows 10, this may be called the Microsoft KM-TEST Loopback Adapter. Refer to the Microsoft Windows 10 procedure in the following location: [https://technet.microsoft.com/en-us/library/cc708322(v=ws.10).aspx](https://technet.microsoft.com/en-us/library/cc708322(v=ws.10).aspx).

1. From the Windows 7 **Start** menu, launch the **Device Manager**.
2. From the **Device Manager**, select **Action**, then select **Add Legacy Hardware**.
3. When the **Add Hardware** wizard displays, click **Next**.
4. Select **Install the hardware that I manually select from a list (Advanced)** and click **Next**.
5. From the **Common hardware types:** list, select **Network adapters** and click **Next**.
6. From the **Manufacturer** drop-down list, select **Microsoft** and **Microsoft [KM-TEST] Loopback Adapter**. Then, click **Next** and **Finish**.
7. From the **Device Manager**, verify that the **Microsoft [KM-TEST] Loopback Adapter** is listed under the Network adapters.

➢➢ To configure the Microsoft Loopback Adapter to add an additional virtual controller to the simulation: add the IP address for the additional virtual controller to the existing Loopback Adapter configuration.
3.2.2 Configure IP Addresses

The simulation computer must have a unique IP Address for the virtual controller.

Note This procedure assumes that you are using the Microsoft Loopback Adapter.

➢➢ To set up an IP Address on the simulation computer for use with the virtual controller

1. From system tray, right-click the network connections icon and select Open Network and Sharing Center.

2. Click Change adapter settings.

3. Right-click the Microsoft [KM-TEST] Loopback Adapter, and select Properties.


5. Select Use the following IP address:, enter a new IP address for your local computer to be used with the virtual controller (for example 192.168.x.x), and click OK.

6. Click Advanced and configure a second IP address in the same subnet, for example:
   one for workstationST
   one for the controller
When all simulation components are installed on the same computer, a Microsoft Loopback Adapter replaces an Ethernet connection. This allows the different simulation components to interact on the same computer without a physical connection.
3.2.3 Download Product and Application Code

As with a real controller, the virtual controller initially has no runtime or application code.

➢ To download runtime and application code

1. Open the controller project by locating the appropriate .sim file.

2. From the Application menu, select Start.

Note: When the simulator is started for the first time, the log window displays a message that runtime is not loaded.

3. From the ToolboxST application, Mark VIe Component Editor, click the Build Configuration (hammer) icon. If the Build command was successful, click the Download (red arrow) icon.

4. When the Download Mark VIe Controller wizard displays, click Next, then click Abort.

Note: Because the virtual controller does not simulate all devices in the actual control software, error messages might display. Check that the error messages correspond to hardware not being simulated. Clear the Scan I/O check box to avoid scanning the I/O when downloading to the virtual controller.

5. From the wizard page, select the desired controller(s) and click Next.

6. Confirm the download by clicking Next until the Download Status dialog box displays.

7. When the Download is complete, close the dialog box.

8. From the Mark VIe Virtual Controller screen, stop the virtual controller then restart.

Note: To perform a restart, stop the virtual controller, start it again, then run the application.
4 Simulation Platforms Integration

This chapter describes how to integrate the virtual controller into other third-party simulation platforms. The simulator API allows outside applications to run instances of the virtual controller with various programs without the SimulatorUI application. This allows larger environments with a Simulation Executive to run the virtual controller instances, establish I/O contracts, and control virtual controller functions. In this environment, one or more process models can provide the inputs to the virtual controller.

The simulator API is a C++ .dll in the form of simapi.dll, located in the virtual controller installed folder. This folder also contains a directory named Include. At the center of the Simulator API is the Simulator class found in a header file called SimulatorAPI.h. This class contains the API functions to the Simulator. For each controller being simulated, an instance of this class must be created. Create a client class with an instance of the Simulator class as a member. Supporting functions and information can then be kept with the controller instance. An array of these client classes can be created, with each element containing one Simulator instance.

Two supporting files through which additional features are exposed are diagserver.dll and cmdserver.dll. The ability to receive error and event messages generated by the virtual controller is provided by the diagserver.dll. The cmdserver.dll and the simapi.dll files allow for capture and replay of EGD commands.

4.1 Project Configuration Files

The virtual controller .sim file is the project configuration file. This file is used by the Simulator API to configure and start the simulation, and is typically created when using the Simulator UI application. To create internal lookup tables for symbol management, three related files are potentially used: a .syl file used to build the syl table, and two .xml files used to build the sym table. The two .xml files contain information for named variables only, and the .syl file contains information about both named variables and all internal states. The distinction between the two becomes important when Save and Restore functions are performed.


4.2 Process/Plant Model Integration

When the StartSimulation() API function runs, the four GESim.exe files for that instance of the virtual controller are started. The simFileName parameter is the path for the appropriate .sim project file. The StartSimulation takes optional arguments of whether to set the simulation time internally to the current computer time or use the time passed to it by the Simulation Executive. If this option is used, the seconds and nanoseconds parameters are used to set the internal time.

---

Note The Include folder contains the header file, which lists functions and defines the API.

---

The StartSimulation() function returns an integer, named result in this example. If a non-zero value is returned, it flags an error that occurred in performing the operation. These flags descriptors are defined in the header file SimulatorErrs.h. This header file is located in the Include folder within the installed folder of the virtual controller. If any API function returns zero (SIM_ERR_EOK), the operation was successful.

4.2.1 I/O Registration

After the simulation starts, you must create the read/write contracts for the I/O that takes place between the virtual controller and the process/plant model. When a contract is created, an integer contract ID is returned. This gives the virtual controller a list of variables names in the controller in a specific order. This contract ID is used for all future reading or writing of these variables.

The I/O contracts perform an I/O transaction between the virtual controller and the process/plant model. CreateReadContract() takes two parameters. The first is an array of C-strings (char arrays), and the second is the number of elements in this array. This array is a list of all variables to be placed in this contract. These C-strings contain the variable names. You must verify that these variables exist in the controller, and in the proper case (upper or lower). The API function ReadVariable() verifies the existence of the variable in the virtual controller. The variable name, and a reference to its value, is processed, and the function returns an integer. If the result is zero, the variable exists in the virtual controller. If the result is non-zero, the variable name is tried again in another case.

4.2.2 Simulator I/O

The ReadData and WriteData functions are members of the simulator class. The contractID of the sought write/read contract is selected and input as the contractID. The buffer processed is a pointer to a shared memory resource between the process/plant model and the virtual controller. The Simulation Executive must create a shared memory block that is properly allocated to hold all values within a contract. Writing to the virtual controller involves copying all variable values for the given contract into the shared memory, and performing the WriteData() function, and applying the proper contractID and a pointer to the shared memory resource. Reading from the virtual controller involves performing the ReadData() function, again passing in the proper contractID, as well as the pointer to the proper shared memory resource. This function causes the virtual controller to copy the appropriate variable values into the shared memory resource. It is required to read from the shared memory, saving the values into the data structure that the process/plant model uses.

---

Note There can be numerous contracts between a simulator and model, and there can be numerous virtual controllers (controllers) for one model, or even multiple models for one instance of the simulator.

---

For systems that include FOUNDATION Fieldbus devices, the macrocycle delay for device-related signals can be included in the I/O data communication between the process/plant model and the virtual controller. This feature will be enabled when the parameter ffmacrocycles is set to True in the sim file created by the V04.06 virtual controller (the default value is True). You can use the older sim file by either adding this parameter or using LoadPrjSettings() API. Set the parameter mincludeMacrocycles to True and perform the Save command using SavePrjSettings() API. When this feature is enabled, ChangeBaseFramePeriod() API will accept only standard frame periods (10, 20, 40, 80, 160, or 320). ChangeBaseFramePeriod() API will not work with any other value.
The use of this shared memory, along with the read/write contracts and API function functions, sends I/O back and forth between the virtual controller and the process/plant model. Since the host environment typically controls stepping a simulation, pausing suspends that stepping loop. If the simulation is stopping, the EndSimulation() function is sent to the simulator, telling it to stop and close its processes.

4.2.3 Identify I/O Variables

A report can be generated to identify which I/O variables are used to create this exchange of data. I/O is sent between the virtual controller and the process/plant model through shared memory (including read/write contracts and API functions).

To identify I/O variables

1. From the ToolboxST System Editor Tree View, double-click a Mark VIe component.
2. From the View menu, select Reports, IO, and Variable.

Example I/O Variable Report
3. From the **Report** menu, select **Change Columns**.

![I/O Variable Report for charfivetwo_G12](image)

**I/O Variable Report for charfivetwo_G12**

<table>
<thead>
<tr>
<th></th>
<th>Change Columns...</th>
<th>Ctrl+F5</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Apply Filter...</td>
<td>Ctrl+F6</td>
</tr>
<tr>
<td>01</td>
<td>Remove Filter</td>
<td>Ctrl+F7</td>
</tr>
<tr>
<td>01</td>
<td>Find...</td>
<td>Ctrl+F</td>
</tr>
</tbody>
</table>

4. From the **Selectable Columns** list, select a column and use the arrow key to move it to the **Viewable Columns** list.

![Select I/O Variable Report Columns](image)

**Select I/O Variable Report Columns**

**Selectable Columns**
- Config Type
- Data Type
- Engr High
- Engr Low
- Entry No
- Port Name
- Raw High
- Raw Low
- SeqDIEvents
- Signal Invert
- Sub-Assembly
- Units

**Viewable Columns**
- I/O Variable
- Direction
- Screw Number
- Screw Name
- TB Name
- TB Hardware Form
- Screw Note
- Wire Number
- Description
- Device Tag
- Cable Number
- Interposing TB
- Interposing TB

5. Save this report as a `.csv` file, then filter and sort it in an Excel spreadsheet. View the report in the ToolboxST application to determine the direction of any variable, either input or output.

### 4.2.4 Run Simulation

After the virtual controller starts, and the I/O contracts have been made, the simulator can be run. This is done by either performing the Run() function or the Step() function on a schedule under the control of the hosting simulation environment.

The virtual controller synchronizes the event loop to real time, then waits for real time to catch up to simulation time before advancing. The virtual controller is then stepped, data is read, the model object response is computed, and the response is written to the virtual controller.

### 4.2.5 Initial Conditions

The simulation allows you to take a snapshot of the entire system being simulated and save the state of each element, then restore that state on demand.
4.2.5.1 Named Save and Restore Functions

When the SaveSimulation() function is performed, the Simulator API saves the initial conditions in a .var file and a .vsm file. These two files are saved in the project folder with the same name as the .sim file. The Simulation Executive copies, renames, and deletes these files. Using the RestoreSimulation() function, SaveSimulation enables a full simulation restore, even if the application code has had major differences applied since the save, but only if the virtual controller successfully loads its .syl file. The SaveSimulation and RestoreSimulation may take several seconds, depending on CPU, simulation size, and other factors. The RestoreSimulation() function expects to find a .vsm and .var file in the project directory with the same name as the .sim file. To manage multiple saved states, the Simulation Executive must copy the appropriate .var and .vsm files to the project folders for each virtual controller to be restored, and rename them to match the name of the device (.sim file name). The SaveSimulation/RestoreSimulation functions can be called with the file name as an argument to avoid the file management that must be performed by the Simulation Executive.

The RestoreSimulation takes optional arguments of whether to set the simulation time internally to the current computer time or use the time passed to it by the Simulation Executive. If this option is used, the seconds and nanoseconds parameters are used to set the internal time. The RestoreSimulation function provides the option to not restore the control constants.

4.2.5.2 Binary Save and Restore Functions

The SaveVarDef() function can be performed in situations where major differences in the .tcw file between the save and the restore are not anticipated. In this case, unlike SaveSimulation, only the .var file is saved. This function enables the BacktrackSimulation() function, which periodically saves the simulation. To manage multiple saved states, the Simulation Executive must copy the appropriate .var file to the project folders for each virtual controller to be restored, and rename it to the name that matches the device name (.sim file name). The SaveVarDef/BacktrackSimulation can be called with the file name as an argument to avoid the file management that must be performed by the Simulation Executive.

Regardless of how a simulation is saved (SaveSimulation or SaveVarDef), the virtual controller can generate two additional files, used by the virtual controller to re-create the alarm queue when the simulation is restored. If there are alarms in the alarm queue, a .alm file is created.

*Note* The virtual controller does not generate a .alm file if the alarm queue is empty. As with the .var and .vsm files, the Simulation Executive must manage copying, renaming, and deleting these files.

The BacktrackSimulation takes optional arguments of whether to set the simulation time internally to the current computer time or take the time passed to it. If this option is used, the seconds and nanoseconds parameters are used to set the internal time.

4.2.5.3 Override

Overriding a variable forces a variable to a desired state. When a variable in the Mark VIe Simulator is overridden, the virtual controller ignores any other inputs to that value.

A variable must be in a simulated state to be overridden and a value must be inserted. These two steps are performed using the SimulateVariable and SimulateVarValue functions. Performing the UnSimulateVariable function returns the variable to a state for accepting normal values. The ForceVariable API function also allows variables to be overwritten. The Unforcevariable API function returns it to its prior state. The health of a variable can be forced or unforced to a defective or good condition using the ForceVarHealth and UnforceVarHealth API functions.
### 4.2.5.4 Sequence of Events

The Sequence of Events (SOEs) records state transitions of the discrete inputs at a 1 ms resolution. They can be generated in the virtual controller through an API exposed in the SimulatorApi.h. The API takes two arguments: an array of pointers to cSOEStructure objects, and the number of such objects in the array. These numbers must match for proper operation. The cSOEStructure class has members as follows:

- Signal Name
- State
- Offset in ms
- Invert

The signal name is used internally to generate the SOE on that particular signal if it is enabled as an SOE. If the variable is not present in the application or not enabled as an SOE, an error is logged, and the API continues to generate SOE for the subsequent variables in the list.

The state indicates whether the SOE is due to a transition from 0 to 1 or 1 to 0. Passing a value of 1 in the state parameter means that the transition is from 0 to 1. Conversely, passing a 0 means that the state transition is from 1 to 0.

The offset in ms takes the offset from the frame boundary at which the SOE is to be time stamped. The given offset is added to the simulation time and the SOE is time stamped at this time. If the offset is greater than the frame rate, it is clamped to the frame rate.

The invert parameter allows the Simulation Executive to model the SOE in the same lines as in the actual Mark VI. If the state is 1 and the invert is 1, the SOE is raised as a transition from 1 to 0 instead of 0 to 1. Results for different combinations is as follows:

<table>
<thead>
<tr>
<th>Start</th>
<th>Invert</th>
<th>Transition from</th>
<th>Transition To</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The SOE generated using the above API do not affect the application state, and only simulate the SOE. If the SOE are generated at the frame boundaries, there could be a discrepancy in the state of the variable shown in the generated SOE when viewed through a CIMPLICITY project or the WorkstationST Alarm Viewer, as well as the state of the variable when trended. To avoid this mismatch, the same value must be passed during the I/O transfers and when generating SOE.

### 4.2.5.5 NVRAM Save and Restore

When the StartSimulation function is called, all variables are given initial values as defined in the downloaded application.

Some Virtual Controller products support NVRAM emulation, where the values of saved variables, totalizers, and forced variables are written to disk during an EndSimulation function call. The next StartSimulation function call reloads those values, replacing the initial values from the downloaded application. Subsequent calls to RestoreSimulation or BackTrackSimulation functions will replace those NVRAM emulation values. The products that support NVRAM emulation are:

- Mark VIe Virtual Controller (32+64), V06.11.00C and higher
- Mark VIeS Virtual Controller, V06.05.00C and higher

**Note** Forced variables are NOT supported as non-volatile in Mark VIeS Virtual Controller.
4.3 Header and Library Files

The following are the relevant header files and library files that are part of the virtual controller. These files are located at the following paths:

- On 32-bit computers: X:\Program Files\GE Energy\<product>\Vxx.xx.xx
- On 64-bit computers: X:\Program Files (x86)\GE Energy\<product>\Vxx.xx.xx

where X refers to the drive on which the ControlST products are installed. Vxx.xx.xx refers to the version of the virtual controller.

The <product> refers to one of the following products:

<table>
<thead>
<tr>
<th>&lt;product&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark VIe Virtual Controller</td>
<td>Mark VIe Virtual Controller (32-bit only)</td>
</tr>
<tr>
<td>Mark VIe Virtual Controller X64</td>
<td>Mark VIe Virtual Controller (64-bit only)</td>
</tr>
<tr>
<td>Mark VIe Virtual Controller (32+64)</td>
<td>Mark VIe Virtual Controller, consolidated 32-bit and 64-bit</td>
</tr>
<tr>
<td>Mark VIeS Virtual Controller</td>
<td>Mark VIeS Virtual Controller (Safety Product)</td>
</tr>
</tbody>
</table>

4.3.1 Header Files

Classes and structures present in the header files for external integrators are as follows.

File :: SimulatorApi.h
Location :: X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Include\SimulatorApi.h

This header file consists of a class declaration for the cProjectInfo class. The data members of this class hold information relating to the simulation configuration that the user wishes to create. Detailed comments are provided next to each data member inside the header file.

File :: SimulatorErrs.h
Location :: X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Include\SimulatorErrs.h

This header file contains common error codes returned by virtual controller APIs. Most of the virtual controller APIs that run successfully return a zero value, unless they return a negative error code. An error code returned from an API can be passed as an argument to GetErrorString() API function to get a corresponding error string.

File :: SymbolTable.h
Location :: X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Include\SymbolTable.h

This header file provides a symbol table entry structure as it is stored in the virtual controller. All the variables created by the user in the application code are downloaded into the simulation folder as .xml files. During initialization, the virtual controller parses these .xml files, and fills the data members of a new object, belonging to the cExtraSymInfo class, for each variable. All variables currently loaded into the virtual controller can be retrieved by passing a buffer as input argument to the GetSymbolInfo() API or GetExtraSymbolInfo() API. The size of the buffer passed should be equal to (Total Number of Symbols) * sizeof(cSymInfo) for the former and (Total Number of Symbols) * sizeof(cExtraSymInfo) for the latter. The total number of symbols is retrieved by using GetSymbolCount() API. The cSymInfo provides information about the variable name, the token, and the array size. The cExtraSymInfo provides information about the units and the high and low limits of the variable in addition to the information provided by the cSymInfo. Whether the variable is an intrinsic or not, and whether high and low limits hold default or user-entered values is retrieved by bit-masking the mStateOfVar variable. Details of usage are provided in the SimulatorApi.h file.
This header file provides the seven data types supported by the virtual controller. This file consists of an enum structure that supports the data types. The following table provides an equivalent data type for each of the virtual controller data types.

<table>
<thead>
<tr>
<th>Virtual controller data type</th>
<th>Equivalent data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL_TYPE</td>
<td>bool</td>
</tr>
<tr>
<td>INT_TYPE</td>
<td>short</td>
</tr>
<tr>
<td>DINT_TYPE</td>
<td>long</td>
</tr>
<tr>
<td>REAL_TYPE</td>
<td>float</td>
</tr>
<tr>
<td>LREAL_TYPE</td>
<td>double</td>
</tr>
<tr>
<td>UINT_TYPE</td>
<td>unsigned short</td>
</tr>
<tr>
<td>UDINT_TYPE</td>
<td>unsigned long</td>
</tr>
</tbody>
</table>

This header file provides the class that uses the Action Capture and Replay (ACR) feature. The user needs to create a command server object of the class cCmdServer defined in this header file for recording and replaying CIMPLICITY and SDI commands.

This header file provides the class that reads the diagnostic messages from the virtual controller. The user needs to create a Diagserver object of the class cDiagServer defined in this header file for capturing these diagnostic messages. The user needs to pass a callback function pointer to the constructor while creating the object.

This header file contains character strings that provide information about the version.

This header file contains a POSIX compliant-time specification structure.
4.3.2 Library Files

The library files are located at the following path: X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Lib
where X refers to the drive on which the ControlST products are installed. Vxx.xx.xx refers to the version.

The single environment products (32-bit or 64-bit) all use the same library name as they will be unique within that product directory. The consolidated products (32-bit and 64-bit) use an -x64 suffix on the end of the library file name for the 64-bit libraries.

Three .lib files are part of the virtual controller installation. These library files are to be linked by external integrators to use the virtual controller features. These files are as follows.

- **File** :- simapi.lib
  - **Location** :- X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Lib\simapi.lib

  This library file must be linked with the application being developed to use the APIs exported by the virtual controller.

- **File** :- cmdserver.lib
  - **Location** :- X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Lib\cmdserver.lib

  This library file must be linked with the application being developed to use the Action Capture and Replay feature.

- **File** :- DiagServer.lib
  - **Location** :- X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Lib\DiagServer.lib

  This library file must be linked with the application being developed to read diagnostic messages sent by the virtual controller.

The 64-bit libraries in the consolidated 32-bit and 64-bit products add -x64 to the library name, resulting in library names of:

- **File** :- simapi-x64.lib
  - **Location** :- X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Lib\simapi-x64.lib

- **File** :- cmdserver-x64.lib
  - **Location** :- X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Lib\cmdserver-x64.lib

- **File** :- DiagServer-x64.lib
  - **Location** :- X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Lib\DiagServer-x64.lib
4.4 Diagnostics and Log Files

The virtual controller publishes all diagnostic messages over a server that third-party simulation applications can access. Two types of published messages include:

- Diagnostic event messages
- Diagnostic error messages

Capture diagnostics messages that the virtual controller publishes on the diagnostics server as follows.

Include file...

```
X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Include\DiagServer.h
```

in the user code

Create an object of class cDiagServer in the application, which takes a function pointer as an argument. Since displaying captured diagnostic messages is in the scope of external simulation applications, the function pointer must point to a user-written function that displays all captured data. The messages are published as a character buffer, which includes the following information:

- Tag event or error
- Time ASCII form of tm time structure
- Process name one of four processes
- Event description message description

Sample code is as follows:

```
DisplayMessage (char *Buf)
{
    // User written display method
    const char End[] = "\r\n";
    CString strBuffer = Buf;
    strBuffer.Remove('n');
    CString CLogView::m_strLog = ""
    m_strLog += strBuffer + End;
    // Display in external software GUI
}
pDiagServer = new cDiagServer(DisplayMessage);
```

The Command Server Init(InstanceCount) function is now performed. Since external simulation applications might interact with multiple virtual controllers, the Init function takes the instance count of the current virtual controller as an input parameter. This captures the messages from the correct controller.

Example code is as follows:

```
pDiagServer->Init(gSimulatorCount); //Initialize the Command Server
```

This completes the diagnostics server initialization. From this point on, the user-written display method is accessed every time the virtual controller publishes a diagnostic message.

The virtual controller also writes formatted diagnostic messages to one of two 1-MB log files (GESim1.log and GESim2.log). When the first log file is full, messages are written to the second log file. When the second log file is full, messages in the first log file are overwritten. These log files are present inside the simulation folder (co-existing with the .sim and pcode files).
4.5 *Action Capture and Replay*

The virtual controller can capture specific commands that alter the state of the running application, and replay them at a later time. Capture of certain EGD commands from the CIMPLICITY screen and SDI commands from the ToolboxST application is currently supported.

Before the start of capture, the simulation is saved and the StartCapture() command is issued. From this point, commands from the CIMPLICITY project and the ToolboxST application are captured and sent back to the SimulationExecutive through a function pointer registered by the executive. The StopCapture() command is issued whenever the capture is to be stopped. The SimulationExecutive is responsible for storing commands and time stamps. When the simulation is to be replayed, the Backtrack() command is issued to go back to the saved state. The Replay() is then called at the appropriate instant in simulation whenever a command is to be replayed until all are complete.

The *cmdserver.dll* must be loaded for this feature, and the *CmdServer.h* file must be referenced.

Include file *...X:\Program Files\GE Energy\<product>\Vxx.xx.xx\Include\CmdServer.h* in the user code.

Create an object of class cCmdServer in the application. The Init(FunctionPointer) or Init(FunctionPointerWithDeviceName, DeviceName) is now called. Both of them take function pointers with different arguments. One takes only a character buffer whereas the other takes both the character buffer and the device name as arguments. The latter lets the user know which device published the message. Since storing captured command messages is in the scope of external simulation applications, the function pointer must point to a user-written function that stores all captured data.

The character buffer sent out is a structure as follows:

```c
struct ACRCaptureBuffer
{
    int mLength;
    int mType;
    unsigned char mRecvBuffer[MAX_MSG_SIZE];
};
```

The mLength contains the total length of the message including the size of mLength field, the size of type mType field, and the actual length of the mRecvBuffer. The mType is a field used by the virtual controller internally to distinguish between EGD and SDI capture data. The mRecvBuffer, which contains the actual data, can be a maximum of 1500 bytes.
Sample code for capture is as follows:

```c
pCmdServer = new cCmdServer;
pSimulator = new Simulator;

int CaptureFunction(char *Log)
{
    FILE *fp=NULL;
    int ReturnValue =0;
    unsigned long FrameNumber;
    int ret;
    unsigned long wrt;
    HANDLE fd;

    fd = CreateFile("Cmdlog",FILE_APPEND_DATA,FILE_SHARE_WRITE,NULL,OPEN_ALWAYS,
    FILE_ATTRIBUTE_NORMAL,NUL);
    if(fd == INVALID_HANDLE_VALUE)
    {
        printf("File creation failed\n");
        return -1;
    }
    pSimulator->GetFrameNumber(FrameNumber);
    ret = WriteFile(fd,&FrameNumber, 4,&wrt,NULL);
    if( (ret == 0) || (wrt != 4) )
    {
        printf("Write Failed");
        return -1;
    }
    int Length = *(int*)(Log);
    ret = WriteFile(fd,Log,Length,&wrt,NULL);
    if( (ret == 0) || (wrt != Length) )
    {
        printf("Write Failed");
        return -1;
    }
    CloseHandle(fd);
    return 0;
}
```
pCmdServer->Init(CaptureFunction);
pSimulator->StartSimulation("Path.sim", NULL, NULL);
pSimulator->SaveVardef();
pCmdServer->StartCapture();
pSimulator->Step();
pSimulator->Step();
pCmdServer->StopCapture();
pSimulator->EndSimulation();
pCmdServer->Close();

..  

pCmdServer->Init(CaptureFunction);
pSimulator->StartSimulation("Path.sim", NULL, NULL);
pSimulator->SaveVardef();
pCmdServer->StartCapture();
pSimulator->Step();
pSimulator->Step();
pCmdServer->StopCapture();
pSimulator->EndSimulation();
pCmdServer->Close();

..  

If the Init() method has the device name, the function pointer also should contain the device name as one of the arguments. If the same function pointer is used for multiple virtual controllers, it can be used in the function to differentiate between all capture messages.

Sample code for replay is as follows:

```c
pCmdServer = new cCmdServer;
pSimulator = new Simulator;
struct ACRCaptureBuffer CMDBuffer;
pSimulator->PauseSimulation();
pSimulator->BackTrackSimulation();
fd = CreateFile("Cmdlog",FILE_READ_DATA,FILE_SHARE_READ,NULL,OPEN_ALWAYS,
FILE_ATTRIBUTE_NORMAL,NULL);
if(fd)
{
```
while(ReadFile(fd,&FrameNumber,4,&rd,NULL) && rd != 0)
{
    while(CurrentFrameNumber != FrameNumber)
    {
        pSimulator->Step(true);
        pSimulator->GetFrameNumber(CurrentFrameNumber);
        Sleep(pSimulator->GetConfiguredBaseFrameRate());
    }
    ret = ReadFile(fd1,&CMDBuffer,2*sizeof(int),&rd,NULL);
    int LengthToRead = *(int*)&CMDBuffer - (2* sizeof(int));
    int CaptureMsgType = *((int*)&CMDBuffer+1);
    ret = ReadFile(fd1,(char*) &CMDBuffer+(2*sizeof(int)),LengthToRead,
                   &rd,NULL);
    Status = pSimulator->Replay((char*) &CMDBuffer,CMDBuffer.mLength);
}
CloseHandle(fd1);
5 Mark VIe Multi-core Support

The Mark VIe Virtual Controller (32+64) product supports Mark VIe applications that include multi-core operation. Multi-core operation allows users to distribute their application program to run across multiple CPU cores. The Mark VIe Controller Multi-core Capability User Guide (GEH-6827) defines the concepts and implementation of multi-core in the Mark VIe Controller, this section defines additional information about multi-core operation in the virtual controller.

Program Scheduling
The virtual controller schedules secondary execution groups to run on a different CPU core than the primary execution group, but unlike the real controller it does not bind each secondary execution group to a particular core. This allows multiple secondary execution groups that preempt each other in the real controller to run in parallel in the virtual controller, thereby optimizing faster-than-real-time operation. It also prevents introducing a limitation on the number of virtual controllers due to the number of available CPU cores.

Single Stepping
Single stepping the virtual controller executes a single primary execution group frame. Any secondary execution group that is due to start in that frame is also started and runs concurrently and independent of the primary execution group. The single step is complete when the primary execution group has finished, the secondary execution groups continue if they span multiple primary execution group frames. Single stepping again runs another primary execution group frame. This continues until it gets to the primary execution group frame that matches the final frame of the secondary execution group. At that time if the secondary execution group has not finished, the primary execution group waits for it to complete before considering the frame complete. All outputs from the secondary execution group are transferred back to the primary execution group before signaling the frame as complete. This means the simulation executive has up-to-date outputs from the secondary execution groups that have completed that frame with no special coordination required.

Breakpoints
Breakpoints are only supported from the primary execution group. Breakpoints can be set on programs running in a secondary execution group (it does not generate an error) but execution is not halted when the program is executed. This allows programs to be moved between execution groups without generating breakpoint errors, but the virtual controller only supports activating breakpoints for programs running in the primary execution group.

Save and Restore
Save operations (both binary and named) wait for all execution groups to complete their sequencing before saving variable values. This prevents the unstable condition of saving values while secondary execution groups are changing them. Secondary execution groups that have finished sequencing have updated their variable values, but they are not transferred to other execution groups until its final frame. This allows the Save and Restore operations to capture at any frame boundary and not have to wait until the final frame of the secondary execution groups. Restore operations always start at the beginning of the next frame, but it does not need to be the first frame of a secondary execution group because it knows that the sequencing has completed and it is awaiting the final frame to transfer the results to other execution groups.

Simulation Executive Inputs and Outputs
The Simulation Executive should only transfer I/O variables and not internal variables. This becomes significant in a multi-core environment as the virtual controller handles the synchronized transfer of variable values at each execution group's start and final frame. If the simulation executive reads and writes variables that are not protected by that transfer procedure it can alter variables in the middle of secondary execution group's sequencing. All I/O variables are automatically protected by the transfer mechanism and are safe to be transferred in any frame. Variables defined in secondary execution groups are not protected and are not defined as I/O variables.
Symbol Table Information

Due to the frame-coordinated transfer of variables between execution groups the virtual controller maintains multiple copies (multiple tokens) of a single named variable. Only one execution group can write to a variable, and for all I/O variables that writing execution group is the primary execution group. When the virtual controller is asked for symbol information for a variable, it supplies the token for the writing execution group instance of the variable. This token should be used for sending commands, for I/O (if it is an I/O variable), and this is the token reported in alarm, event, or SOE reports. This token should also be used to force, unforce, or simulate a variable, as the writing execution group value is propagated to all other execution groups at their start frame.

Command Timing

Commands are sent to the writing execution group token for a variable, which is then propagated to all other execution groups during the normal frame transfer of data. For all I/O variables, that is always in the primary execution group and the command is processed each frame. When the writing execution group token is in a secondary execution group, the command is queued in the virtual controller to be delivered at the next start frame for that secondary execution group. This prevents commands from changing the value of a variable in the middle of its sequencing.

ChangeBaseFramePeriod Function is Not Supported

The ChangeBaseFramePeriod function is not supported in Virtual Controller products that include multi-core operation. The function remains defined in the API for upwards compatibility, but calling it does not impact operation. To change the base frame rate, use the same procedure used with the real Mark VIe controllers - change the base frame rate in ToolboxST and download the controller.

If the Virtual Controller's ChangeBaseFramePeriod function was used to execute simulations faster than real time while using the Run function, use the new SetClockMultiplier function as a replacement. It allows you to run faster than real time without needing to switch from the clock-based Run function to the Simulation Executive based Step function.

New SetClockMultiplier Function

A new SetClockMultiplier function has been added to the Virtual Controller products that include multi-core operation. This new function adjusts the relative start time from one frame to the next so that frames run earlier they normally would. The time base increments at the base frame rate, so the Virtual Controller time base increments faster than real time. This means that time versus value reports or trends look the same whether it was run in real time or faster than real time.

The operating system uses 1 millisecond scheduling granularity, so the fastest the Virtual Controller can be run is with an effective 1 millisecond frame rate. That means a 10 millisecond base frame rate can run no faster than 10x normal speed, a 20 millisecond base frame rate can run no faster than 20x normal speed, and so on. The 1 millisecond granularity also means that small changes in the clock multiplier may not impact the (integer) milliseconds between frames. The actual integer millisecond frame rate is computed as the base frame rate divided by the clock multiplier, with a minimum value of one.

If the computer cannot complete the frame processing during the effective frame rate, then the frames run as fast as they can - back to back with no delay. The clock multiplier value can also be less than one, causing it to run slower than real time. If this causes timeouts in the EGD and SDI outputs (such as when the Virtual Controller is paused or stepped slowly) then extra outputs are generated to prevent receivers from declaring the Virtual Controller as unhealthy. A clock multiplier of zero will reset back to the base frame rate (equivalent to a clock multiplier of 1.0).
Consolidating 32-bit and 64-bit Products

The Mark VIe Virtual Controller (32+64) product consolidates using both 32-bit and 64-bit Simulation Executive programs with 32-bit virtual Mark VIe application programs. This contrasts with the previous Mark VIe Virtual Controller X64 product which supplied 64-bit virtual controller applications. Consolidating to a single set of 32-bit application programs mimics the real Mark VIe controller, but it does have a few implications for the Simulation Executive and its environment:

- Both the 32-bit and 64-bit Simulation Executives use the same header files (*.h) when compiling. This requires that the 64-bit compilation use the Microsoft default setting of an int being defined as 32-bits.
- When linking a 64-bit Simulation Executive use the 64-bit libraries (with the -x64 suffix) instead of the 32-bit libraries. See the section Header and Library Files for additional details.
- When downloading additional block libraries (such as ARES, or ones built using the Block Library Creator product) always download or copy the 32-bit version of the libraries. The 64-bit versions of the libraries are no longer required, even when using a 64-bit Simulation Executive. This means that blocks that do not supply a 64-bit library can now be used with a 64-bit Simulation Executive.
Notes
**Glossary of Terms**

**API**  Application Programming Interface. A set of functions that provide application functions to other components.

**.dll**  Dynamic Link Library (file extension)

**EGD**  Ethernet Global Data. Control network and protocol for the controller. Devices share data through EGD exchanges (pages)

**Fidelity**  The ability of the simulation and the controls to match the operation of the running power plant.

**IC**  Initial Condition (simulation state)

**NVRAM**  Non-Volatile Random-Access Memory, used to store saved variables, totalizers, and forced variables. The Virtual Controller uses disk files to emulate NVRAM.

**Online**  Mode that provides full CPU communications, allowing data to be both read and written. It is the state of the application when it is communicating with the system for which it holds the configuration. Online is also a download mode where the device is not stopped and then restarted.

**pcode**  A binary set of records created by the application, which contain the controller application configuration code for a device. Pcode is stored in RAM and Flash memory.

**pin**  Block, macro, or module parameter that creates a variable used to make interconnections.

**Plant Data Highway (PDH)**  Ethernet communication network between the HMI Servers, HMI Viewers and workstations.

**product code (runtime)**  Software stored in the controller’s Flash memory that converts application code (pcode) to executable code.

**runtime**  Refer to pcode.

**Sequence of Events (SOE)**  High-speed record of contact transitions taken during a plant event for data analysis.

**server**  A computer that gathers data over the Ethernet from plant devices, and makes the data available to computer-based operator interfaces known as viewers.

**simplex**  Operation mode that requires only one set of control and I/O, and generally uses only one channel. The entire Mark VIe control system can operate in simplex mode.

**simulation**  Running a system without all configured I/O devices by modeling the behavior of the machine and the devices in software.

**TCP/IP**  Communication protocols developed to inter-network dissimilar systems. It is a de facto UNIX standard, but is supported on almost all systems. TCP controls data transfer and IP provides the routing for functions, such as file transfer and email.

**trend**  A time-based plot to show the history of values, similar to a recorder, available in the Turbine Historian.

**Unit Data Highway (UDH)**  Connects the Mark VI controller, static starter control system, excitation control system, PLCs, and other GE-provided equipment to the HMI Servers.

**WorkstationST**  A ControlST application that provides a user interface for configuring various features such as alarm management, control system health, Historians, and data interfaces.
Index

A
Action capture and replay 39
Alarms 11–12, 33
Application code 27
Architecture 11

B
binary save and restore 33

C
Components
  power plant simulator 9
  system 11
Configuration 15
Configure
  IP addresses 25
  loopback adapter 24
  ToolboxST system 16

D
diagnostic
  messages 38
Download
  application code 27

F
features 8
files
  library 35
  log 38
Files
  .xml 29
functions
  binary save and restore 33
  named save and restore 33

H
Hardware 7
header files 35

I
initial conditions 32
I/O
  identify variables 31
  registration 30
  simulator 30
  Installation 15

L
library files 37
Limitations
  system 12
  log files 38
LoopbackAdapter
  configure 24

M
Mark VIe Multi-core Support 43
  breakpoints 43
  ChangeBaseFramePeriod function 44
  command timing 44
  consolidating 32-bit and 64-bit products 45
  program scheduling 43
  save and restore 43
  SetClockMultiplier function 44
  simulation executive inputs/outputs 43
  single stepping 43
  symbol table 44
  messages
    diagnostic 38

N
NVRAM 34

O
Operating systems 7
override
  variable 33

P
Process/Plant model 30
Product and application code 27

R
Requirements
  simulation executive 13

S
save and restore
  binary 33
  named 33
Sequence of Events 34
Simulation
  run 32
Simulation platforms integration 29
Simulator
components 9
System
layout 12
limitations 12
requirements 15
simulation components 11
ToolboxST 16
System architecture 11

T
ToolboxST system configuration 16

V
variable
override 33
Virtual controller
create 20
Virtual Controller
features 8

W
WorkstationST configuration 23