Tanner Declaration Exhibit H
MODEL TRAIN CONTROL SYSTEM

Inventor: Matthew A. Katzer, 1416 NW Benfield Dr, Portland, OR (US) 97229

Assignee: Matthew A. Katzer, Hillsboro, OR (US)

Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

Appl. No.: 10/124,878
Filed: Apr. 17, 2002

Prior Publication Data
US 2002/0170458 A1 Nov. 21, 2002

Related U.S. Application Data
Continuation of application No. 09/858,222, filed on Apr. 17, 2002, now Pat. No. 6,660,467.

Int. Cl. ........................................... A63H 19/00
U.S. Cl. .................. 105/L5; 246/167 R; 246/197;
246/62
Field of Search ................. 105/1.5, 1.4, 29-2;
246/187 A, 167 R, 197, 62; 701/20

References Cited
U.S. PATENT DOCUMENTS
3,976,272 A 8/1976 Murray et al.
4,307,302 A 12/1981 Russell
5,072,900 A 12/1991 Malon

5,475,818 A * 12/1995 Molyneaux et al. ........... 701/20
5,493,642 A 2/1996 Dunsmuir et al.
5,638,522 A 6/1997 Dunsmuir et al.
5,696,689 A 12/1997 Okumura et al.
5,896,017 A 4/1999 Severson et al.
5,940,005 A 8/1999 Severson et al.
5,952,797 A 9/1999 Rossler
6,005,406 A * 5/2000 Katzer ..................... 105/L4
6,267,061 B1 7/2001 Katzer
6,270,040 B1 8/2001 Katzer

OTHER PUBLICATIONS
Chapell, David, Understanding ActiveX and OLE, 1996, Microsoft Press, Redmond.

* cited by examiner

Primary Examiner—William A. Cuchiniski, Jr.
Assistant Examiner—Olga Hernandez
Attorney, Agent, or Firm—Chernoff, Vihauer, McChung & Stenzel, LLP

ABSTRACT
A system which operates a digitally controlled model railroad transmitting a first command from a first client program to a resident external controlling interface through a first communications transport. The second command is transmitted from a second client program to the resident external controlling interface through a second communications transport. The first command and the second command are received by the resident external controlling interface which queues the first and second commands. The resident external controlling interface sends third and fourth commands representative of the first and second commands, respectively, to a digital command station for execution on the digitally controlled model railroad.

27 Claims, 3 Drawing Sheets
MODEL TRAIN CONTROL SYSTEM

This application is a continuation of U.S. patent application Ser. No. 09/858,222 filed on Apr. 17, 2002 U.S. Pat. No. 6,460,467.

BACKGROUND OF THE INVENTION

The present invention relates to a system for controlling a model railroad.

Model railroads have traditionally been constructed with a set of interconnected sections of train track, electric switches between different sections of the train track, and other electrically operated devices, such as train engines and drawbridges. Train engines receive their power to travel on the train track by electricity provided by a controller through the track itself. The speed and direction of the train engine is controlled by the level and polarity, respectively, of the electrical power supplied to the train track. The operator manually pushes buttons or pulls levers to cause the switches or other electrically operated devices to function, as desired. Such model railroad sets are suitable for a single operator, but unfortunately they lack the capability of adequately controlling multiple trains independently. In addition, such model railroad sets are not suitable for being controlled by multiple operators, especially if the operators are located at different locations distant from the model railroad, such as different cities.

A digital command control (DCC) system has been developed to provide additional controllability of individual train engines and other electrical devices. Each device the operator desires to control, such as a train engine, includes an individually addressable digital decoder. A digital command station (DCC) is electrically connected to the train track to provide a command in the form of a set of encoded digital bits to a particular device that includes a digital decoder. The digital command station is typically controlled by a person through a computer. A standard interface for the digital command control system in the NMRA DCC Standards, issued March 1997, and is incorporated herein by reference. While providing the ability to individually control different devices of the railroad set, the DCC system still fails to provide the capability for multiple operators to control the railroad devices, especially if the operators are remotely located from the railroad set and each other.

Digital Systems of Lawrenceville, Ga. has developed a software program for controlling a model railroad set from a remote location. The software includes an interface which allows the operator to select desired changes to devices of the railroad set that include a digital decoder, such as increasing the speed of a train or switching a switch. The software issues a command locally or through a network, such as the internet, to a digital command station at the railroad set which executes the command. The protocol used by the software is based on Cobra from Open Management Group where the software issues a command to a communication interface and awaits confirmation that the command was executed by the digital command station. When the software receives confirmation that the command executed, the software program sends the next command through the communication interface to the digital command station. In other words, the technique used by the software to control the model railroad is analogous to an inexpensive printer where commands are sequentially issued to the printer after the previous command has been executed. Unfortunately, it has been observed that the response of the model railroad to the operator appears slow, especially over a distributed network such as the internet. One technique to decrease the response time is to use high-speed network connections but unfortunately such connections are expensive.

What is desired, therefore, is a system for controlling a model railroad that effectively provides a high-speed connection without the additional expense associated therewith.

The foregoing and other objectives, features, and advantages of the invention will be more readily understood upon consideration of the following detailed description of the invention, taken in conjunction with the accompanying drawings.

SUMMARY OF THE PRESENT INVENTION

The present invention overcomes the aforementioned drawbacks of the prior art, in a first aspect, by providing a system for operating a digitally controlled model railroad, that includes transmitting a first command from a first client program to a resident external controlling interface through a first communications transport. A second command is transmitted from a second client program to the resident external controlling interface through a second communications transport. The first command and the second command are received by the resident external controlling interface which queues the first and second commands. The resident external controlling interface sends third and fourth commands representative of the first and second commands, respectively, to a digital command station for execution on the digitally controlled model railroad.

Incorporating a communications transport between the multiple client programs and the resident external controlling interface permits multiple operators of the model railroad at locations distant from the physical model railroad and each other. In the environment of a model railroad club where the members want to simultaneously control devices of the same model railroad layout, which preferably includes multiple trains operating thereon, the operators each provide commands to the resident external controlling interface, and hence the model railroad. In addition by queuing by commands at a single resident external controlling interface permits controlled execution of the commands by the digitally controlled model railroad, which may otherwise conflict with one another.

In another aspect of the present invention the first command is selectively processed and sent to one of a plurality of digital command stations for execution on the digitally controlled model railroad based upon information contained therein. Preferably, the second command is also selectively processed and sent to one of the plurality of digital command stations for execution on the digitally controlled model railroad based upon information contained therein. The resident external controlling interface also preferably includes a command queue to maintain the order of the commands.

The command queue also allows the sharing of multiple devices, multiple clients to communicate with the same device (locally or remote) in a controlled manner, and multiple clients to communicate with different devices. In other words, the command queue permits the proper execution in the cases of: (1) one client to many devices, (2) many clients to one device, and (3) many clients to many devices.

In yet another aspect of the present invention the first command is transmitted from a first client program to a first processor through a first communications transport. The first command is received at the first processor. The first processor provides an acknowledgment to the first client program through the first communications transport indicating that
the first command has properly executed prior to execution of commands related to the first command by the digitally controlled model railroad. The communications transport is preferably a COM or DCOM interface.

The model railroad application involves the use of extremely slow real-time interfaces between the digital command stations and the devices of the model railroad. In order to increase the apparent speed of execution to the client, other than using high-speed communication interfaces, the resident external controller interface receives the command and generates an acknowledgement to the client program in a timely manner before the execution of the command by the digital command stations. Accordingly, the execution of commands provided by the resident external controlling interface to the digital command stations occur in a synchronous manner, such as a first-in-first-out interface. The COM and DCOM communications transport between the client program and the resident external controlling interface is operated in an asynchronous manner, namely providing an acknowledgement thereby releasing the communications transport to accept further communications. The communication is the execution of the command. The operation of the synchronous and the asynchronous data communication for the commands provides the benefit that the operator considers the commands to occur nearly instantaneously while permitting the resident external controlling interface to verify that the command is proper and cause the commands to execute in a controlled manner by the digital command stations, all without additional high-speed communication networks. Moreover, for traditional distributed software execution there is no motivation to provide an acknowledgment prior to the execution of the command because the command executes quickly and most commands are sequential in nature. In other words, the execution of the next command is dependent upon proper execution of the prior command so there would be no motivation to provide an acknowledgment prior to its actual execution.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

FIG. 1 is a block diagram of an exemplary embodiment of a model train control system.

FIG. 2 is a more detailed block diagram of the model train control system of FIG. 1 including external device control logic.

FIG. 3 is a block diagram of the external device control logic of FIG. 2.

DETAILED DESCRIPTION OF THE PREPARED EMBODIMENT

Referring to FIG. 1, a model train control system 10 includes communications transport 12 interconnecting a client program 14 and a resident external controlling interface 16. The client program 14 executes on the model railroad operator's computer and may include any suitable system to permit the operator to provide desired commands to the resident external controlling interface 16. For example, the client program 14 may include a graphical interface representative of the model railroad layout where the operator issues commands to the model railroad by making changes to the graphical interface. The client program 14 also defines a set of Application Programming Interfaces (API's), described in detail later, which the operator accesses using the graphical interface or other programs such as Visual Basic, C++, Java, or browser based applications. There may be multiple client programs interconnected with the resident external controlling interface 16 so that multiple remote operators may simultaneously provide control commands to the model railroad.

The communications transport 12 provides an interface between the client program 14 and the resident external controlling interface 16. The communications transport 12 may be any suitable communications medium for the transmission of data, such as the Internet, local area network, satellite links, or multiple processes operating on a single computer. The preferred interface to the communications transport 12 is a COM or DCOM interface, as developed for the Windows operating system available from Microsoft Corporation. The communications transport 12 also determines if the resident external controlling interface 16 is system resident or remotely located in the resident external controller. The communications transport 12 may also use private or public communications protocol as a medium for communications. The client program 14 provides commands and the resident external controlling interface 16 responds to the communications transport 12 to execute information. A description of COM (common object model) and DCOM (distributed common object model) is provided by Chappel in a book entitled Understanding ActiveX and OLE, Microsoft Press, and is incorporated by reference herein.

Incorporating a communications transport 12 between the client program(s) 14 and the resident external controlling interface 16 permits multiple operators of the model railroad at locations distant from the physical model railroad and each other. In the environment of a model railroad club where the members want to simultaneously control devices of the same model railroad layout, which preferably includes multiple trains operating therein, the operators each provide commands to the resident external controlling interface, and hence the model railroad.

The manner in which commands are executed for the model railroad under COM and DCOM may be as follows. The client program 14 makes requests in a synchronous manner using COM/DCOM to the resident external controller 16. The synchronous manner of the request is the technique used by COM and DCOM to execute commands. The communications transport 12 packages the command for the transport mechanism to the resident external controlling interface 16. The resident external controlling interface 16 then passes the command to the digital command stations 18 which in turn executes the command. After the digital command station 18 executes the command an acknowledgement is passed back to the resident external controlling interface 16 which in turn passes an acknowledgement to the client program 14. Upon receipt of the acknowledgement, the client program 14, the communications transport 12 is again available to accept another command. The train control system 10, without more, permits execution of commands by the digital command stations 18 from multiple operators, but like the DigToys Systems' software the execution of commands is slow.

The present inventor came to the realization that unlike traditional distributed systems where the commands passed through a communications transport are executed nearly instantaneously by the server and then an acknowledgement is returned to the client, the model railroad application involves the use of extremely slow real-time interfaces between the digital command stations and the devices of the model railroad. The present inventor came to the further realization that in order to increase the apparent speed of execution to the client, other than using high-speed communication interfaces, the resident external controller interface 16 should receive the command and provide an
acknowledgement to the client program 12 in a timely manner before the execution of the command by the digital command stations 18. Accordingly, the execution of commands provided by the resident external controlling interface 16 to the digital command stations 18 occur in a synchronous manner, such as a first-in-first-out manner. The COM and DCOM communications transport 12 between the client program 14 and the resident external controlling interface 16 is operating in an asynchronous manner, namely providing an acknowledgement thereby releasing the communications transport 12 to accept further communications prior to the actual execution of the command. The combination of the synchronous and the asynchronous data communication for the commands provides the benefit that the operator can control the model railroad nearly while permitting the resident external controlling interface 16 to verify that the command is proper and cause the commands to execute in a controlled manner by the digital command stations 18, all without additional high-speed communication networks. Moreover, for traditional distributed software execution there is no motivation to provide an acknowledgement prior to the execution of the command because the command executes quickly and most commands are sequential in nature. In other words, the execution of the next command is dependent upon proper execution of the prior command so there would be no motivation to provide an acknowledgement prior to its actual execution. It is to be understood that other devices, such as digital devices, may be controlled in a manner as described for model railroads.

Referring to FIG. 2, the client program 14 sends a command over the communications transport 12 that is received by an asynchronous command processor 100. The asynchronous command processor 100 queries a local database storage 102 to determine if it is necessary to package a command to be transmitted to a command queue 104. The local database storage 102 primarily contains the state of the device of the model railroad, such as, for example, the speed of a train, the direction of a train, whether a drawbridge is up or down, whether a light is turned on or off, and the configuration of the model railroad layout. If the command received by the asynchronous command processor 100 is a query of the state of a device, then the asynchronous command processor 100 retrieves such information from the local database storage102 and provides the information to an asynchronous response processor 106. The asynchronous response processor 106 then provides a response to the client program 14 indicating the state of the device and releases the communications transport 12 for the next command.

The asynchronous command processor 100 also verifies, using the configuration information in the local database storage 102, that the command received is a potentially valid operation. If the command is invalid, the asynchronous command processor 100 provides such information to the asynchronous response processor 106, which in turn returns an error indication to the client program 14.

The asynchronous command processor 100 may determine that the necessary information is not contained in the local database storage 102 to provide a response to the client program 14 of the device state or that the command is a valid action. Actions may include, for example, an increase in the train's speed, or turning on/off of a device. In either case, the valid unknown state or action command is packaged and forwarded to the command queue 104. The packaging of the command may also include additional information from the local database storage 102 to complete the client program 14 request, if necessary. Together with packaging the command for the command queue 104, the asynchronous command processor 100 provides a command to the asynchronous request processor 106 to provide a response to the client program 14 indicating that the event has occurred, even though such an event has yet to occur on the physical railroad layout.

As such, it can be observed that whether or not the command is valid, whether or not the information requested by the command is available, or whether or not the command has been executed, the combination of the asynchronous command processor 100 and the asynchronous response processor 106 both verifies the validity of the command and provides a response to the client program 14 thereby freeing up the communications transport 12 for additional commands. Without the asynchronous nature of the resident external controlling interface 16, the response to the client program 14 would be, in many circumstances, delayed thereby resulting in frustration to the operator that the model railroad is performing in a slow and painstaking manner. In this manner, the railroad operator using the asynchronous interface appears to the operator as nearly instantaneously responsive.

Each command in the command queue 104 is fetched by a synchronous command processor 110 and processed. The synchronous command processor 110 queries a controller database storage 112 for additional information, as necessary, and determines if the command has already been executed based on the state of the devices in the controller database storage 112. In the event that the command has already been executed, as indicated by the controller database storage 112, then the synchronous command processor 110 passes information to the command queue 104 that the command has been executed or the state of the device. The asynchronous response processor 106 fetches the information from the command queue 104 and provides a suitable response to the client program 14, if necessary, and updates the local database storage 102 to reflect the updated status of the railroad layout devices.

If the command fetched by the synchronous command processor 110 from the command queue 104 requires execution by external devices, such as the train engine, then the command is passed to one of several external device control logic 114 blocks. The external device control logic 114 processes the command from the synchronous command processor 110 and issues appropriate control commands to the interface of the particular external device 116 to execute the command on the device and ensure that an appropriate response was received in response. The external device is preferably a digital command control device that transmits digital commands to decoders using the train track. There are several different manufacturers of digital command stations, each of which has a different set of input commands, so each external device is designed for a particular digital command station. In this manner, the system is compatible with different digital command stations. The digital command stations 18 of the external devices 116 provide a response to the external device control logic 114 which is checked for validity and identified as to which prior command it corresponds to so that the controller database storage 112 may be updated properly. The process of transmitting commands to and receiving responses from the external devices 116 is slow.

The synchronous command processor 110 is notified of the results from the external control logic 114 and, if appropriate, forwards the results to the command queue 104. The asynchronous response processor 106 clears the results
from the command queue 104 and updates the local database storage 102 and sends an asynchronous response to the client program 14, if needed. The response updates the client program 14 of the actual state of the railroad track devices, if changed, and provides an error message to the client program 14 if the devices actual state was previously improperly reported or a command did not execute properly.

The use of two separate database storage, each of which is substantially a mirror image of the other, provides a performance enhancement by a fast acknowledgement to the client program 14 using the local database storage 102 and thereby freeing up the communications transport 12 for additional commands. In addition, the number of commands forwarded to the external device control logic 114 and the external devices 116, which are relatively slow to respond, is minimized by maintaining information concerning the state and configuration of the model railroad. Also, the use of two separate database tables 102 and 112 allows more efficient multi-threading on multi-processor computers.

In order to achieve the separation of the asynchronous and synchronous portions of the system the command queue 104 is implemented as a named pipe, as developed by Microsoft for Windows. The queue 104 allows both portions to be separate from each other, where each considers the other to be the destination device. In addition, the command queue maintains the order of operation which is important to proper operation of the system.

The use of a single command queue 104 allows multiple instantiations of the asynchronous functionality, with one for each different client. The single command queue 104 also allows the sharing of multiple devices, multiple clients to communicate with the same device (locally or remote) in a controlled manner, and multiple clients to communicate with different devices. In other words, the command-queue 104 permits the proper execution in the cases of: (1) one client to many devices, (2) many clients to one device, and (3) many clients to many devices.

The present inventor came to the realization that the digital command stations provided by the different vendors have at least three different techniques for communicating with the digital decoders of the model railroad set. The first technique, generally referred to as a transaction (one or more operations), is a synchronous communication where a command is transmitted, executed, and a response is received therefrom prior to the transmission of the next sequentially received command. The DCS may execute multiple commands in this transaction. The second technique is a cache with out of order execution where a command is executed and a response received therefrom prior to the execution of the next command, but the order of execution is not necessarily the same as the order that the commands were provided to the command station. The third technique is a local-area-network model where the commands are transmitted and received simultaneously. In the LAN model there is no requirement to wait until a response is received for a particular command prior to sending the next command. Accordingly, the LAN model may result in many commands being transmitted by the command station that have yet to be executed. In addition, some digital command stations use two or more of these techniques.

With all these different techniques used to communicate with the model railroad set and the system 10 providing an interface for each different type of command station, there exists a need for the capability of matching up the responses from each of the different types of command stations with the particular command issued for record keeping purposes.

Without matching up the responses from the command stations, the databases can not be updated properly.

Validation functionality is included within the external device control logic 114 to accommodate all of the different types of command stations. Referring to FIG. 3, an external command processor 200 receives the validated command from the synchronous command processor 110. The external command processor 200 determines which device the command should be directed to, the particular type of command it is, and builds state information for the command. The state information includes, for example, the address, type, port, variables, and type of commands to be sent out. In other words, the state information includes a command set for a particular device on a particular port device. In addition, a copy of the original command is maintained for verification purposes. The constructed command is forwarded to the command sender 202 which is another queue, and preferably a circular queue. The command sender 202 receives the command and transmits commands within its queue in a repetitive nature until the command is removed from its queue. A command response processor 204 receives all the commands from the command stations and passes the commands to the validation function 206. The validation function 206 compares the received command against potential commands that are in the queue of the command sender 202 that could potentially provide such a result. The validation function 206 determines one of four potential results from the comparison. First, the results could be simply bad data that is discarded. Second, the results could be partially executed commands which are likewise normally discarded. Third, the results could be valid responses but not relevant to any command sent. Such a case could result from the operator manually changing the state of devices on the model railroad or from another external device, assuming a shared interface to the DCS. Accordingly, the results are validated and passed to the result processor 210. Fourth, the results could be valid responses relevant to a command sent. The corresponding command is removed from the command sender 202 and the results passed to the result processor 210.

The commands in the queue of the command sender 202, as a result of the validation process 206, are retransmitted a predetermined number of times, then if error still occurs the digital command station is reset, which if the error still persists then the command is removed and the operator is notified of the error.

APPLICATION PROGRAMMING INTERFACE

Train Tools™ Interface Description
Building your own visual interface to a model railroad
Questions concerning the product can be EMAILED to:
 unintools@kam-nav.com
You can also email questions to:
KAM Industries
2373 NW 185th Avenue Suite 416
Hillsboro, Oregon 97124
PAX: (503) 291-1221
Table of contents
1. OVERVIEW
2. TUTORIAL
2.1 Visual BASIC Throttle Example Application
2.2 Visual BASIC Throttle Example Source Code
3. EDL COMMAND REFERENCE
APPLICATION PROGRAMMING INTERFACE

3.1 Introduction
3.2 Data Types
3.3 Commands to access the server configuration variable database

- continued

KamCVCGetWord
KamCVPutWord
KamCVCGetEnable
KamCVPutEnable
KamCVCGetName
KamCVCMaxRegister
KamCVCGetMaxRegister

3.4 Commands to program configuration variables

- continued

KamProgramSend
KamProgramGetStatus
KamProgramReadCV
KamProgramReadBase
KamProgramReadDecoderToDataBase
KamProgramDecodeFromDatabase

3.5 Commands to control all decoder types

- continued

KamDecoderGetMaxModels
KamDecoderGetModelIndex
KamDecoderSetModelIndex
KamDecoderGetMaxAddresses
KamDecoderChangeOldNewAddr
KamDecoderMovePort
KamDecoderGetPort
KamDecoderCheckAddrInUse
KamDecoderGetModelFromObj
KamDecoderGetModelFromIndex
KamDecoderGetModelFromFacility
KamDecoderGetObjCount
KamDecoderGetObjAddr
KamDecoderPutAdd
KamDecoderPutDel
KamDecoderGetMsgName
KamCVCGetEnableMode
KamDecoderGetMaxSpeed

3.6 Commands to control locomotive decoders

- continued

KamLingPutSpeed
KamLingGetSpeedSteps
KamLingGetFunction
KamLingPutFunction
KamLingGetFunctionMax
KamLingSetFunction
KamLingPutFunctionMax
KamLingGetFunctionName
KamLingPutFunctionName
KamLingGetCircuitMax
KamLingPutCircuitParent
KamLingPutCircuitChild
KamLingPutCircuitRemoveObj

3.7 Commands to control accessory decoders

- continued

KamAccGetFunction
KamAccGetFunctionAll
KamAccPutFunction
KamAccPutFunctionAll
KamAccGetFunctionMax
KamAccGetFunctionName
KamAccPutName
KamAccPutFunctionName
KamAccRegFeedback
KamAccRegFeedbackAll
KamAccDefFeedback
KamAccDefFeedbackAll

3.8 Commands to control the command station

- continued

KamOptPutTrackOnStation
KamOptPutSwitchStation
KamOptPutSwitchPosition
KamOptPutPowerOn
KamOptPutPowerOff
KamOptPutHsReset
KamOptPutHealingSlope
KamOptPutEmergencyStop
KamOptPutSequenceStation

APPLICATION PROGRAMMING INTERFACE

5.9 Commands to configure the command station

- continued

communication port
KamPortPutConfig
KamPortGetConfig
KamPortPutName
KamPortPutMaxPort
KamPortGetMaxPort
KamPortGetMaxPhysical

5.10 Commands that control command flow to the command station

- continued

KamCmdConnect
KamCmdDisconnect
KamCmdSendMessage
KamCmdPutCAB
KamCmdPutAdd

5.12 Miscellaneous Commands

- continued

KamMiscGetErrorMsg
KamMiscGetClockTime
KamMiscGetClockTime
KamMiscGetInterfaceVersion
KamMiscSaveData
KamMiscGetController
KamMiscGetControllerName
KamMiscGetControllerNameAll
KamMiscGetControllerPara
KamMiscGetControllerStation
KamMiscGetControllerIndex
KamMiscSetController

1. OVERVIEW

This document is divided into two sections, the
Tutorial, and the IDL Command Reference. The tutorial
shows the complete code for a simple Visual BASIC
program that controls all the major functions of a locomotive.
This program makes use of many of the commands described
in the reference section. The IDL Command Reference
describes each command in detail.

I. TUTORIAL

A. Visual BASIC Throttle Example Application

The following application is created using the
Visual BASIC source code in the next section. It
controls all major locomotive functions such as speed,
direction, and auxiliary functions.

1. Visual BASIC Throttle Example Source Code

   Copyright 1998, KAM Industries. All rights reserved.

   This is a demonstration program showing the integration of
   VisualBasic and Train Server(tm) interface. You may use this
   application for non-commercial usage.

$Desc: $      $Author: $      $Revision: $
50        
Engine Commander, Computer Dispatcher, Train Server,
Train Tools, The Conductor and kamilind are registered
trademarks of KAM Industries. All rights reserved.

This first command adds the reference to the Train
Server Interface object TmEngCmd As New EngCmd

Engine Commander uses the term Ports, Devices and
Controllers

Ports -> These are logical ids where Decoders are
assigned to. Train Server/CT Interface supports a
limited number of logical ports. You can also think
of ports as mapping to a command station type. This
allows you to move decoders between command station
without losing any information about the decoder

Devices -> These are communications channels
configured in your computer.

You may have a single device (com1) or multiple
devices
<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW</th>
<th>Document 47</th>
<th>Filed 06/09/2006</th>
<th>Page 11 of 27</th>
</tr>
</thead>
</table>

### APPLICATION PROGRAMMING INTERFACE

- **COM 1 - COM6, LPT1, Others.** You are required to map a port to a device to access a command station.
- **Devices start from ID 0 to max ID (FYI: devices do not necessarily have to be serial channel).** Always check the same name of the device before you use it as well as the maximum number of devices supported.
- **The Command**
  - **EngCom.KamPortGetMaxPhysical(MaxPhysical, iSerial, iParallel) provides means that:** `iMaxPhysical = iSerial + iParallel + iOther`.
- **Controller - These are command the command station**
  - Like LENS, Digitrax, etc.
  - No hassle, easy DCC, Modem, etc. It is recommended that you check the command station ID before you use it.
- **Errors - All commands return an error status. If**
  - The error value is not zero, then the other return arguments are invalid.
  - In general, non-zero errors mean commands were not executed. To get the error message, you need to call KamMiscErrMessage and supply the error number.

**To operate your layout, you will need to perform a mapping between a Port (logical reference), Device (physical communications channel) and a Controller (command station) for the program to work. All references use the logical device as the reference device for access.**

- **Addresses used are an object reference. To use an address you must add the address to the command station using KamDecoderPutAddr.** One of the return values from this operation is an object reference.
- **We need certain variables as global objects; since**
  - the information is being used multiple times
  - **Dim iLogicalPort, iController, iComPort**
  - **iPortData, iPortProtocol, iPortStop, iPortSettings, iPortWatchdog, iPortFlow, PortData**
  - **Dim RawDevice As Long, iDecoderClass As Integer, iEncoderType As Integer, iMaxController As Long, iMaxLogical As Long, iMaxPhysical As Long, iMaxSerial As Long**

---

### Form load function

**Turn off the initial buttons**

**Set interface information**

### Private Sub Load()

**Dim strVer As String, strCom As String, strCold As String**

**Dim iError As Integer**

**'Get the interface version information**

**SetButtonState (False)

**If (Error) Then**

**MsgBox ("Train Server not loaded. Check DCMM-95")**

**iLogicalPort = 0**

**LogPort.Caption = iLogicalPort**

**ComPort.Caption = "??"**

**Controller.Caption = "Unknown"**

**Else**

**MsgBox ("(Simulation)(COM1) Train Server ... " & strVer)"

**Configuration information; Only need to change these values to use a different controller**

**' UNKOWN**

**' SIMULAT**

**5**

- **LLENZ_1x**
- **LLENZ_2x**
- **DIOCTR_D2700**
- **DIOCTR_DCS100**
- **MASTERSERIES**
- **SYSTEMONE**
- **RAMFX**
- **DYNATROL**
- **Nortel I Can 10**
- **SERIAL**
- **EASYDCC**
- **MRK550**
- **MRK623**
- **ZTC**
- **DIOCTR_FR1**
- **DIOCTR_FR2**
- **DIOCTR_FR3**

**LogicalPort = 1 Select Logical port 1 for communications**

**iController = 1 Select controller from the list above.**

**iComPort = 0 Use COM1; 0 means COM1 (Digitrax must use COM1 or COM2)**

**Digitrax Bas rate requires 16K**

- Most COM ports above COM2 do not support 16K. Check with the manufacturer of your smart com card for the baud rate. Keep in mind that COM2 is used for control.

**'Support COM4 - COM4 can only support'**

- **2 COM ports (like com1 & com2)**
- **'or com4 & com5'**

- **'If you change the controller, do not forget to change the baud rate to match the command station. See your user manual for details.**

**0: 'Baud rate is 300**

**1: 'Baud rate is 1200**

**2: 'Baud rate is 2400**

**3: 'Baud rate is 4800**

**4: 'Baud rate is 9600**

**5: 'Baud rate is 14.4**

**6: 'Baud rate is 16.4**

**7: 'Baud rate is 19.2**

**iPortRate = 4**

**' Parity values 0-4 -> no, odd, even, mark, space**

**iPortParity = 0**

**' Stop bits 0,1,2 -> 1,1.5,2**

**iPortStop = 0**

**' iPortReturns = 10**

**' iPortWatchdog = 2048**

**iPortFlow = 0**

**' Data bits 0 -> 8 Bits, 1-> 8 bits**

**iPortData = 1**

**Display the port and controller information**

**Error = EngCom.KamPortGetMaxPhysical(iMaxLogical, iMaxSerial, iMaxController)**

**' Get the port name and do some checking...**

**Error = EngCom.KamPortGetName(iComPort, strCom)**

**SetError (Errors) If**

**Error**

**Else**

**If (LogicalPort > iMaxLogical) Then MessageBox ("Com port out of range")**

**Error = EngCom.KamMiscGetControllerName(iController, strCold)**

**If (LogicalPort > iMaxLogical) Then MessageBox**
APPLICATION PROGRAMMING INTERFACE

("Logical port out of range")
SetErr (Error)
End If

If DisplayValuesInThrottle = True
    LogPort.Captn = LogPort.Captn + vbCrLf + PromptCaption + vbCrLf +
                  Control_caption + vbCrLf + vbCrLf
End If

***************
Send Command
Note:
- Please follow the command order. Order is important.
- For application to work...

Private Sub Command_Click()
' Send the command from the interface to the command
' station, use the engineObject
Dim Error As Integer
If Not ConnectEnabled Then
    ' This means that you need to set up the CV's or
    ' other operations first, then execute the command.
    IsSpeed = SpeedText
    Error = EngCmd.KamPortPutFunction(EngineObject, 0, 0, Value)
    If Error = EngCmd.KamPortPutFunction(EngineObject, 1, 1, Value)
        Error = EngCmd.KamPortPutFunction(EngineObject, 2, 2, Value)
    End If
    ' Debugging information
    If Error = EngCmd.KamComCommand(EngineObject)
        SetErr (Error)
    End If
End If

***************
Connect Controller

Private Sub Connect_Click()
Dim Error As Integer

'These are the index values for setting up the port
for use
' PORT_RETRANS 0 // Retrans index
' PORT_RATE 1 // Retrans index
' PORT_PARTY 2 // Retrans index
' PORT_STOP 3 // Retrans index
' PORT_WATCHDOG 4 // Retrans index
' PORT_FLOW 5 // Retrans index
' PORT_DATABITS 6 // Retrans index
' PORT_DEBUG 7 // Retrans index
' PORT_PAIRALLE 8 // Retrans index

'These are the index values for setting up the port
for use
' PORT_RETRANS 0 // Retrans index
' PORT_RATE 1 // Retrans index
' PORT_PARTY 2 // Retrans index
' PORT_STOP 3 // Retrans index
' PORT_WATCHDOG 4 // Retrans index
' PORT_FLOW 5 // Retrans index
' PORT_DATABITS 6 // Retrans index
' PORT_DEBUG 7 // Retrans index
' PORT_PAIRALLE 8 // Retrans index

*** Continue ***

APPLICATION PROGRAMMING INTERFACE

5
iError = EngCmd.KamPortPutConfig(LogicalPort, 4)
iPortWatchdog (0) setting PORT_WATCHDOG
iError = EngCmd.KamPortPutConfig(LogicalPort, 5)
iPortNow (0) setting PORT_FLOW
iError = EngCmd.KamPortPutConfig(LogicalPort, 6)
iPortData (0) setting PORT_DATABITS

10 ' We need to set the appropriate debug mode for display.
' this command can only be set if the following is true
' - Controller is not connected
' - port has not been mapped
' - Not serial version of application (Sharedor
' - always set to 130)

15 ' Display Write Log Debug
File Win Level Value
' 1 + 2 + 4 = 7
' 1 + 2 + 8 = 11
' 1 + 2 + 16 = 19
' 1 + 2 + 32 = 35
' 1 + 2 + 64 = 69
' 1 + 2 + 128 = 130

20 ' COMMONLY -- Read comm write
' comm ports
iError = EngCmd.KamComCommand(EngineObject)
SetErr (Error)
End If

30 ' Note: 1. This does effect the performance of your
' system; 130 is a save value for debug
' display. Always set the key to 1, a value
' of 0 will disable debug
2. The Digitrax control codes displayed are
' encrypted. The information that you

35 determine from the control codes is that
' information is sent (B) and a response is
' received (R)

40 ' DebugMode = 130
'iValue = ValueText

45 ' Now map the Logical Port, Physical device, Command
' station and Controller
'iError = EngCmd.KamPortPutMapController(LogicalPort, 0, IController, iComPort)

50 ' Error (Error) Displays the error message and error
number

55 Private Sub DCCAddr_Click()
Dim iAddr As Integer

All addresses must be match to a logical port to
operate
iDecoderType = 1 ' Set the decoder type to an NMRA

baseLineDecoder (1 - 8 mg)
iDecoderClass = 1 ' Set the decoder class to Engine
decoder (there are only two classes of decoders);
Engine and Accessory

Once we make a connection, we use the engineObject
as the reference object to send control information
If (Address.Text = 1) Then
iStatus = EngCmd.KamDecoderPutAdd (Address.Text, 0, iLogicalPort, iLogicalPort, 0, '

60 DecoderType, EngineObject)
US 6,530,329 B2

APPLICATION PROGRAMMING INTERFACE

Private Sub Disconnect_Click()
    Dim Error As Integer
    iError = engEng.KamCmdDisConnect(0)(LogicalPort)
    SetError (Error)
End Sub

Display error message

Private Sub SetError(Error As Integer)
    Dim sError As String
    Dim Status
    "This shows how to retrieve a sample error message from the interface for the status received.
    Status = engEng.KamMaxConErrorMsg(Error, sError)
    ErrorMsg.Caption = sError
    Result.Caption = Status(Status)
End Sub

APPLICATION PROGRAMMING INTERFACE

Private Sub engEng.CommEnd_Connect
    Dim Error As Integer
    iError = engEng.KamCmdCommEndConnect(0)(LogicalPort)
    SetError (Error)
End Sub

Private Sub engEng.CommEnd_Disconnect
    Dim Error As Integer
    iError = engEng.KamCmdDisConnect(0)(LogicalPort)
    SetError (Error)
End Sub

Private Sub engEng.KamCmdCommEndConnect
    Dim Error As Integer
    iError = engEng.KamCmdCommEndConnect(0)(LogicalPort)
    SetError (Error)
End Sub

Private Sub engEng.KamCmdDisConnect
    Dim Error As Integer
    iError = engEng.KamCmdDisConnect(0)(LogicalPort)
    SetError (Error)
End Sub

Private Sub engEng.KamCmdConn
    Dim Error As Integer
    iError = engEng.KamCmdConn(0)(LogicalPort)
    SetError (Error)
End Sub

Data is passed to and from the IDL interface using a series of primitive data types. Arrays of these simple types are also used. The exact type passed to and from

The following primitive data types are used:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Short integer</td>
</tr>
<tr>
<td>long</td>
<td>Long integer</td>
</tr>
<tr>
<td>BSTR</td>
<td>String</td>
</tr>
<tr>
<td>NameID</td>
<td>CV Range</td>
</tr>
<tr>
<td>Accessory</td>
<td>Speed Range</td>
</tr>
</tbody>
</table>

KamCVPutValue communicates only with the server, not the actual decoder. You then use the programming commands in the next section to transfer CVs to and from the decoder.
APPLICATION PROGRAMMING INTERFACE

-continued

Case 3:06-cv-01905-JSW     Document 47     Filed 06/09/2006     Page 14 of 27

17

-continued

APPLICATION PROGRAMMING INTERFACE

5  phCVWinString to the name of the CV as defined in NMRA
   Recommended Practice RP 9.2.2.
   @KamCVWinMinRegister
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   pMinRegister int * 2 Out Pointer to min CV
   register number

10 1 3:06-cv-01905-JSW     Document 47     Filed 06/09/2006     Page 14 of 27

   0x0001 - SET CV_MINIMIZE
   0x0002 - SET CV_READ
   0x0004 - SET CV_WRITE
   0x0008 - SET CV_ERROR
   Return Value Type Range Description
   iError short 1 Error flag
   1 iError = 0 for success.  Nonzero is an error number
      (see KamMiscGetErrorMsg).  KamCVPutValue takes the
decoder object ID and configuration variable (CV) number, and
a new CV value as parameters.  It sets the server copy of the
specified CV to the new CV value.
   @KamCVPutValue
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   CVValue int 0-255 In CV value
   1 Opaque object ID handle returned by
   KamDecoderPutAdd.

2 Maximum CV is 1024.  Maximum CV for this decoder is
given by KamCVGetMaxRegister.
   Return Value Type Range Description
   iError short 1 Error flag
   1 iError = 0 for success.  Nonzero is an error number
      (see KamMiscGetErrorMsg).  KamCVGetPutEnable takes the
decoder object ID, configuration variable (CV) number, and
a pointer to store the enable flag as parameters.  It sets the
location pointed to by pEnable to eEnable.
   @KamCVPutEnable
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   iEnable short 1 In CV enable
   1 Opake object ID handle returned by
   KamDecoderPutAdd.

2 Maximum CV is 1024.  Maximum CV for this decoder is
given by KamCVGetMaxRegister.
   Return Value Type Range Description
   iError short 1 Error flag
   1 iError = 0 for success.  Nonzero is an error number
      (see KamMiscGetErrorMsg).  KamCVPutEnable takes the
decoder object ID, configuration variable (CV) number, and
a new enable state as parameters.  It sets the server copy of the
CV's enable bit to eEnable.
   @KamCVGetEnable
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   iEnable short 1 In CV enable
   1 Opake object ID handle returned by
   KamDecoderPutAdd.

3 0x0001 - SET CV_MINIMIZE
   0x0002 - SET CV_READ
   0x0004 - SET CV_WRITE
   0x0008 - SET CV_ERROR
   0x0010 - SET CV_ERROR_WRITE
   Return Value Type Range Description
   iError short 1 Error flag
   1 iError = 0 for success.  Nonzero is an error number
      (see KamMiscGetErrorMsg).  KamCVSetPutEnable takes the
decoder object ID, configuration variable (CV) number, and
a new enable state as parameters.  It sets the server copy of the
CV's enable bit to eEnable.
   @KamCVGetEnable
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   iEnable short 1 In CV enable
   1 Opake object ID handle returned by
   KamDecoderPutAdd.

3 0x0001 - SET CV_MINIMIZE
   0x0002 - SET CV_READ
   0x0004 - SET CV_WRITE
   0x0008 - SET CV_ERROR
   0x0010 - SET CV_ERROR_WRITE
   Return Value Type Range Description
   iError short 1 Error flag
   1 iError = 0 for success.  Nonzero is an error number
      (see KamMiscGetErrorMsg).  KamCVGetPutEnable takes the
decoder object ID, configuration variable (CV) number, and
a new enable state as parameters.  It sets the server copy of the
CV's enable bit to eEnable.
   @KamCVSetEnable
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   iEnable short 1 In CV enable
   1 Opake object ID handle returned by
   KamDecoderPutAdd.

Maximum CV is 1024.  Maximum CV for this decoder is
given by KamCVGetMaxRegister.
   Return Value Type Range Description
   iError short 1 Error flag
   1 iError = 0 for success.  Nonzero is an error number
      (see KamMiscGetErrorMsg).  KamCVGetPutEnable takes the
decoder object ID, configuration variable (CV) number, and
a new enable state as parameters.  It sets the server copy of the
CV's enable bit to eEnable.
   @KamCVSetEnable
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   iEnable short 1 In CV enable
   1 Opake object ID handle returned by
   KamDecoderPutAdd.

Maximum value for this server given by
   KamPortGetMaxLogPort.
   0 PROGRAM_MODE_NONE
   1 PROGRAM_MODE_ADDRESS
   2 PROGRAM_MODE_REGIST
   3 PROGRAM_MODE_PAGE
   4 PROGRAM_MODE_DIRECT
   5 CODEL PROGRAM_MODE_OPS_SHORT
   6 PROGRAM_MODE_OPS_LONG
   Return Value Type Range Description
   iError short 1 Error flag
   1 iError = 0 for success.  Nonzero is an error number
      (see KamMiscGetErrorMsg).  KamProgramSetProgramMode takes the
decoder object ID, logical programming post ID, and programming mode as parameters.  It
changes the command station mode from normal operation
(PROGRAM_MODE_NONE) to the specified programming mode.
Once in programming modes, any number of programming
commands may be called.  When done, you must call
KamProgram with a parameter of PROGRAM_MODE_NONE to
return to normal operation.
APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>iProgLogPort</td>
<td>int</td>
<td>1-65535</td>
<td>2</td>
<td>In</td>
</tr>
</tbody>
</table>

pProbeMode | int | 3 | Out | Programming mode |

Opaque object ID handle returned by KamDecoderPutAdd.

MaxPort for this server given by KamFrontGetMaxLogPorts.

PROGRAM_MODE_NONE
PROGRAM_MODE_ADDRESS
PROGRAM_MODE_REGISTER
PROGRAM_MODE_PAGE
PROGRAM_MODE_DATA
PROGRAM_MODE_OPS_SHORT
PROGRAM_MODE_OPS_LONG

Return Value Type Range Description
iError short 1 Error flag Description

If iError = 0 for success. Noerrors is an error number (see KamMiscGetErrorMsg).

KamProgramGetMode takes the decoder object ID, logical programming port ID, and pointer to a place to store the programming mode as parameters. It sets the memory pointed to by pProbeMode to the present programming mode.

Opaque object ID handle returned by KamDecoderPutAdd.

0 returns OR'd value for all CVs. Other values return status for just that CV

0x0007 - SFT_CV_INUSE
0x0002 - SFT_CV_READ_DIRTY
0x0004 - SFT_CV_WRITE_DIRTY
0x0008 - SFT_CV_ERROR_READ
0x0010 - SFT_CV_ERROR_WRITE

Return Value Type Range Description
iError short 1 Error flag Description

If iError = 0 for success. Noerrors is an error number (see KamMiscGetErrorMsg).

KamProgramGetStatus takes the decoder object ID and pointer to a place to store the OR'd decoded programming status as parameters. It sets the memory pointed to by pProbeMode to the present programming mode.

Opaque object ID handle returned by KamDecoderPutAdd.

1 opague object ID handle returned by KamDecoderPutAdd.

Maximum CV is 1024. Maximum CV for this decoder is given by KamCVGetMaxRegister.

Return Value Type Range Description
iError short 1 Error flag Description

If iError = 0 for success. Noerrors is an error number (see KamMiscGetErrorMsg).

KamProgramCV takes the decoder object ID, configuration variable (CV) number, and a new CV value as parameters. It programs (writes) a single decoder CV using the specified value as source data.

KamProgramReadDecoderDB

Parameter | Type | Range | Direction | Description |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IDecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
</tbody>
</table>

Opaque object ID handle returned by KamDecoderPutAdd.

Return Value Type Range Description
iError short 1 Error flag Description

If iError = 0 for success. Noerrors is an error number (see KamMiscGetErrorMsg).

KamProgramReadDecoderDB takes the decoder object ID as a parameter. It reads all enabled CV values from the decoder and stores them in the server database.

Opaque object ID handle returned by KamDecoderPutAdd.

Return Value Type Range Description
iError short 1 Error flag Description

If iError = 0 for success. Noerrors is an error number (see KamMiscGetErrorMsg).

KamProgramReadDecoderDB takes the decoder object ID as a parameter. It programs (writes) all enabled decoder CV values using the server copy of the CVs as source data.

A. Commands to control all decoder types.

This section describes the commands that all decoder types. These commands do things such as getting the maximum address a given type of decoder supports, editing decoders to the database, etc.

Opaque object ID handle returned by KamDecoderPutAdd.

Returns maxModels int 1 Out Pointer to Max model ID

MaxModels 1 Normally 1-65535, 0 on error

Return Value Type Range Description
iError short 1 Error flag Description

If iError = 0 for success. Noerrors is an error number (see KamMiscGetErrorMsg).

KamDecoderGetMaxModels takes no parameters. It sets the memory pointed to by pMaxModels to the maximum decoder type ID.

Opaque object ID handle returned by KamDecoderPutAdd.

Returns maxModels int 1 Out Pointer to Max model ID

Model int 1-65535 1 In Decoder type id

Opaque object ID handle returned by KamDecoderPutAdd.

Returns maxModels int 1 Out Pointer to Max model ID

Model int 1-65535 1 In Decoder type id

KamDecoderGetMaxModels.

String

Opaque object ID handle returned by KamDecoderPutAdd.

Return Value Type Range Description
iError short 1 Error flag Description

If iError = 0 for success. Noerrors is an error number (see KamMiscGetErrorMsg).

KamDecoderGetMaxModels.

Opaque object ID handle returned by KamDecoderPutAdd.

Return Value Type Range Description
iError short 1 Error flag Description

If iError = 0 for success. Noerrors is an error number (see KamMiscGetErrorMsg).

KamDecoderGetMaxModels.

Opaque object ID handle returned by KamDecoderPutAdd.

Return Value Type Range Description
iError short 1 Error flag Description

If iError = 0 for success. Noerrors is an error number (see KamMiscGetErrorMsg).

KamDecoderSetModelToCVObj takes a decoder ID and takes a decoder object ID as parameters. It sets the decoder model type of the decoder address iDecoderObjectID to the type.
<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>pbsMgName</td>
<td>BSTR</td>
<td>2</td>
<td>Out</td>
<td>Pointer to manufacturer name</td>
</tr>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>isError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
</tbody>
</table>

### Application Programming Interface

**Case 3:06-cv-01905-JSW**

**Document 47**

**Filed 06/09/2006**

**Page 17 of 27**

---

**US 6,530,329 B2**

---

**Case 3:06-cv-01905-JSW**

**Document 47**

**Filed 06/09/2006**

**Page 17 of 27**

---

**US 6,530,329 B2**

---

**Case 3:06-cv-01905-JSW**

**Document 47**

**Filed 06/09/2006**

**Page 17 of 27**

---

**US 6,530,329 B2**

---

**Case 3:06-cv-01905-JSW**

**Document 47**

**Filed 06/09/2006**

**Page 17 of 27**

---

**US 6,530,329 B2**

---

**Case 3:06-cv-01905-JSW**

**Document 47**

**Filed 06/09/2006**

**Page 17 of 27**

---

**US 6,530,329 B2**

---

**Case 3:06-cv-01905-JSW**

**Document 47**

**Filed 06/09/2006**

**Page 17 of 27**

---

**US 6,530,329 B2**

---
APPLICATION PROGRAMMING INTERFACE (continued)

iError short 3 Error flag
1 iError = 0 for success. Nonzero is an error number
(see KmMacGetErrorMsg).

KmEngGetSpeed takes the decoder object ID and parameters to
locate a decoder object in the locomotive database and
directs the speed to be returned.

KmEngGetSpeed
Parameter List
<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
</tr>
<tr>
<td>iSpeed</td>
<td>int</td>
<td>* 2</td>
<td>Out</td>
</tr>
<tr>
<td>iDirection</td>
<td>int</td>
<td>* 3</td>
<td>Out</td>
</tr>
</tbody>
</table>

1 iOpague object ID handle returned by
KamDecoderPutAddr.

iSpeed range is dependent on whether the decoder is set to 14, 18, or 24 steps speed steps and matches the values defined by NMRA S0.2 and RP 9.2.1. 0 is stop and 1 is emergency stop for all modes.

2 Forward is boolean TRUE and reverse is boolean FALSE.

Return Value
<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError short</td>
<td>1</td>
<td>Error flag</td>
</tr>
</tbody>
</table>

1 iError = 0 for success. Nonzero is an error number
(see KmMacGetErrorMsg).

KmEngGetSpeed takes the decoder object ID, new
locomotive speed, and new locomotive direction as
parameters. It sets the locomotive database speed to
iSpeed and the locomotive database direction to
iDirection. Note: This command only changes the
locomotive database. The data is not sent to the decoder
until execution of the KmCmdCommand command. Speed is
set to the maximum possible for the decoder if iSpeed exceeds the decoder range.

KmEngGetSpeedSteps
Parameter List
<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
</tr>
<tr>
<td>iSpeedSteps</td>
<td>int</td>
<td>* 14,28,128</td>
<td>Out</td>
</tr>
</tbody>
</table>

1 iOpague object ID handle returned by
KamDecoderPutAddr.

Return Value
<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError short</td>
<td>1</td>
<td>Error flag</td>
</tr>
</tbody>
</table>

1 iError = 0 for success. Nonzero is an error number
(see KmMacGetErrorMsg).

KmEngGetSpeedSteps takes the decoder object ID and a
pointer to a location to store the number of speed steps
as a parameter. It sets the memory pointed to by
iSpeedSteps to the number of speed steps.

KmEngGetSpeedSteps
Parameter List
<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
</tr>
<tr>
<td>iSpeedSteps</td>
<td>int</td>
<td>* 14,28,128</td>
<td>Out</td>
</tr>
</tbody>
</table>

1 iOpague object ID handle returned by
KamDecoderPutAddr.

iError short 1 Error flag
1 iError = 0 for success. Nonzero is an error number
(see KmMacGetErrorMsg).

KmEngGetSpeedSteps takes the decoder object ID and a
new number of speed steps as a parameter. It sets the
number of speed steps in the locomotive database to
iSpeedSteps. Note: This command only changes the
locomotive database. The data is not sent to the decoder
even if execution of the KmCmdCommand command.
KmEngGetMaxSpeed returns the
maximum possible speed for the decoder. An error is
generated if an attempt is made to set the speed steps
beyond this value.

KmEngGetFunction.
Parameter List
<table>
<thead>
<tr>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iFunctionID</td>
<td>long</td>
<td>1</td>
<td>In</td>
</tr>
<tr>
<td>iFunction</td>
<td>int</td>
<td>* 0-8-2</td>
<td>Out</td>
</tr>
</tbody>
</table>

1 iOpague object ID handle returned by
KamDecoderPutAddr.
APPLICATION PROGRAMMING INTERFACE

```
KamEngPutName takes a decoder object ID and a BSTR as parameters. It sets the symbolic locomotive name to bsEngName.

0KamEngGetFunctionName

Parameter List
Type   Range   Description
IDecoderObjectID   long    1  In  Decoder object ID
iFunctionID       int      0-8 2  In  Function ID number
pbFunctionName    BSTR*    3  Out  Pointer to function name

Opaque object ID handle returned by KamDecoderPutAdd.
```

2. FL is 0. FL-Fe are 1-8 respectively. Maximum for this decoder is given by KamEngGetFunctionMax. Exact return type depends on language. It is CurTag* for C++. Empty string on error.

```
Return Value  Type    Range   Description
iError short  1  Error flag
1 iError = 0 for success. Nonzero is an error number
(see KmsMacGetErrorMessage).
KamEngGetFunctionName takes a decoder object ID, function ID, and a pointer to the function name as parameters. It sets the memory pointed to by pbFunctionString to the symbolic name of the specified function.
```

```
0KamEngGetFunctionName

Parameter List
Type   Range   Description
IDecoderObjectID   long    1  In  Decoder object ID
iFunctionID       int      0-8 2  In  Function ID number
pbFunctionString  BSTR    3  Out  Function name

Opaque object ID handle returned by KamDecoderPutAdd.
```

2. FL is 0. FL-Fe are 1-8 respectively. Maximum for this decoder is given by KamEngGetFunctionMax.

```
3. Exact parameter type depends on language. It is LPCSTR for C++. Return Value  Type    Range   Description
iError short  1  Error flag
1 iError = 0 for success. Nonzero is an error number
(see KmsMacGetErrorMessage).
KamEngGetFunctionMax takes a decoder object ID, function ID, and a BSTR as parameters. It sets the specified symbolic function name to pbFunctionString.
```

```
0KamEngGetFunctionMax

Parameter List
Type   Range   Description
IDecoderObjectID   long    1  In  Decoder object ID
pbFunctionMax      int*     2  Out  Pointer to max consist number

Opaque object ID handle returned by KamDecoderPutAdd.
```

2. Command stations dependent.

```
Return Value  Type    Range   Description
iError short  1  Error flag
1 iError = 0 for success. Nonzero is an error number
(see KmsMacGetErrorMessage).
KamEngGetConsistMax takes the decoder object ID and a pointer to a location to store the maximum consist as parameters. It sets the location pointed to by pbMaxConsist to the maximum number of locomotives that can be placed in a command station controlled consist.
```

Note that this command is designed for command station consisting. CV consisting is handled using the CV commands.

```
KamEngGetConsistParent

Parameter List
Type   Range   Description
IDCCParentObjectID   long  1  In  Parent decoder object ID

IDCAliasAddr    int    2  In  Alias decoder address

Opaque object ID handle returned by KamDecoderPutAdd.
```

2. 1-127 for short locomotive addresses. 1-10239 for long locomotive decoders.

```
Return Value  Type    Range   Description
iError short  1  Error flag
1 iError = 0 for success. Nonzero is an error number
(see KmsMacGetErrorMessage).
KamEngGetConsistParent takes the parent object ID and an alias address as parameters. It makes the decoder
```
### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>29</strong> <strong>US 6,530,329 B2</strong></td>
</tr>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

#### DecoderPutData

1. Opague object ID handle returned by `KamDecoderPutData`
2. Each bit represents a single function state. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |

#### DecoderPutFunction

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectId long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>iFunctionID int</td>
<td>0-31</td>
<td>Function ID number</td>
<td></td>
</tr>
<tr>
<td>iValue int</td>
<td>3</td>
<td>Function value</td>
<td></td>
</tr>
</tbody>
</table>
1. Opague object ID handle returned by `KamDecoderPutData`
2. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Function action is boolean `TRUE` and inactive is boolean `FALSE`.
4. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |

#### DecoderPutFunctionAll

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectId long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>iValue int</td>
<td>2</td>
<td>Pointer to function state array</td>
<td></td>
</tr>
</tbody>
</table>
1. Opague object ID handle returned by `KamDecoderPutData`
2. Each bit represents a single function state. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |

#### DecoderPutFunctionMax

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectId long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pMaxMaxFunction int</td>
<td>0-31</td>
<td>2 Out</td>
<td>Pointer to maximum function number</td>
</tr>
</tbody>
</table>
1. Opague object ID handle returned by `KamDecoderPutData`
2. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |

#### DecoderPutFunctionName

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectId long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>iFunctionID int</td>
<td>0-31</td>
<td>Function ID number</td>
<td></td>
</tr>
<tr>
<td>pFuncNameString BSTR *</td>
<td>2 Out</td>
<td>Pointer to function name</td>
<td></td>
</tr>
</tbody>
</table>
1. Opague object ID handle returned by `KamDecoderPutData`
2. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |

#### DecoderPutFunction

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectId long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>iFunctionID int</td>
<td>0-31</td>
<td>Function ID number</td>
<td></td>
</tr>
<tr>
<td>pFuncNameString BSTR *</td>
<td>3 Out</td>
<td>Pointer to function name</td>
<td></td>
</tr>
</tbody>
</table>
1. Opague object ID handle returned by `KamDecoderPutData`
2. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |

#### DecoderPutFunctionName

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectId long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>iFunctionID int</td>
<td>0-31</td>
<td>Function ID number</td>
<td></td>
</tr>
<tr>
<td>pFuncNameString BSTR</td>
<td>3 Out</td>
<td>Pointer to function name</td>
<td></td>
</tr>
</tbody>
</table>
1. Opague object ID handle returned by `KamDecoderPutData`
2. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |

#### DecoderPutFunction

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectId long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>iFunctionID int</td>
<td>0-31</td>
<td>Function ID number</td>
<td></td>
</tr>
<tr>
<td>pFuncNameString BSTR</td>
<td>3 Out</td>
<td>Pointer to function name</td>
<td></td>
</tr>
</tbody>
</table>
1. Opague object ID handle returned by `KamDecoderPutData`
2. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |

#### DecoderPutFunctionName

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectId long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>iFunctionID int</td>
<td>0-31</td>
<td>Function ID number</td>
<td></td>
</tr>
<tr>
<td>pFuncNameString BSTR</td>
<td>3 Out</td>
<td>Pointer to function name</td>
<td></td>
</tr>
</tbody>
</table>
1. Opague object ID handle returned by `KamDecoderPutData`
2. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |

#### DecoderPutFunction

<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW Document 47 Filed 06/09/2006 Page 20 of 27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-continued</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectId long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>iFunctionID int</td>
<td>0-31</td>
<td>Function ID number</td>
<td></td>
</tr>
<tr>
<td>pFuncNameString BSTR</td>
<td>3 Out</td>
<td>Pointer to function name</td>
<td></td>
</tr>
</tbody>
</table>
1. Opague object ID handle returned by `KamDecoderPutData`
2. Maximum for this decoder is given by `KamallocGetFunctionMax`
3. Return Value: Type Range | Description
   | iError short | iError flag |
   | iError = 0 for success, Nonzero is an error number (see `KamallocGetErrorMsg`) |
APPLICATION PROGRAMMING INTERFACE

(see KmsMiscGetLastError).

KmsAccGetFeedbackAll takes a decoder object ID, node name string, and function ID, as parameters. It returns interest in the function given by the function ID given by the method given by the node name string bstrAccNode.

bstrAccNode identifies the server application and method to call if the function changes state. In format is "%([Server])([App])([Method])" where [Server] is the server name, [App] is the application name, and [Method] is the method name.

KmsAccGetFeedBackAll
Parameter List Type Range Direction Description
DecoderObject ID long 1 In Decoder object ID
bstrAccNode BSTR 2 In Server node name
1 opaque object ID handle returned by KmsDecodePutAdd.
2 exact parameter type depends on language. It is LPCSTR for C++.

Return Value Type Range Description
Error short 1 Error flag
1 Error = 0 for success. Nonezero is an error number (see KmsMiscGetLastError).

KmsAccGetFeedBackAll takes a decoder object ID, node name string, and function ID, as parameters. It returns interest in the function given by the function ID given by the method given by the node name string bstrAccNode.

bstrAccNode identifies the server application and method to call if the function changes state. In format is "%([Server])([App])([Method])" where [Server] is the server name, [App] is the application name, and [Method] is the method name.

KmsAccGetFeedBackAll
Parameter List Type Range Direction Description
DecoderObject ID long 1 In Decoder object ID
bstrAccNode BSTR 2 In Server node name
1 opaque object ID handle returned by KmsDecodePutAdd.
2 exact parameter type depends on language. It is LPCSTR for C++.

Return Value Type Range Description
Error short 1 Error flag
1 Error = 0 for success. Nonezero is an error number (see KmsMiscGetLastError).

KmsAccGetFeedBackAll takes a decoder object ID, node name string, and function ID, as parameters. It returns interest in the function given by the function ID given by the method given by the node name string bstrAccNode.

bstrAccNode identifies the server application and method to call if the function changes state. In format is "%([Server])([App])([Method])" where [Server] is the server name, [App] is the application name, and [Method] is the method name.

A. Commands to control the command station

This section describes the commands that control the command station. These commands do things such as controlling command station power. The steps to control a given command station vary depending on the type of command station.

KmsOpnPutTurnOnStation
Parameter List Type Range Direction Description
LogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by KmsPortGetMaxLogPort.
Return Value Type Range Description
Error short 1 Error flag
1 Error = 0 for success. Nonezero is an error number (see KmsMiscGetLastError).

KmsOpnPutTurnOffStation takes a logical port ID as a parameter. It performs the steps necessary to turn on the command station. This command performs a combination of other commands such as KmsOpnPutStartStation, KmsOpnPortClearStation, and KmsOpnPutPowerOn.

KmsOpnPutStartStation
Parameter List Type Range Direction Description
LogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by KmsPortGetMaxLogPort.
Return Value Type Range Description
Error short 1 Error flag
1 Error = 0 for success. Nonezero is an error number (see KmsMiscGetLastError).

KmsOpnPutClearStation takes a logical port ID as a parameter. It performs the steps necessary to start the command station.

KmsOpnPutClearStation
Parameter List Type Range Direction Description
LogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by KmsPortGetMaxLogPort.
Return Value Type Range Description
Error short 1 Error flag
1 Error = 0 for success. Nonezero is an error number (see KmsMiscGetLastError).

KmsOpnPutClearStation takes a logical port ID as a parameter. It performs the steps necessary to clear the command station queue.

KmsOpnPutStopStation
Parameter List Type Range Direction Description
LogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by KmsPortGetMaxLogPort.
Return Value Type Range Description
Error short 1 Error flag
1 Error = 0 for success. Nonezero is an error number (see KmsMiscGetLastError).

KmsOpnPutStopStation takes a logical port ID as a parameter. It performs the steps necessary to stop the command station.

KmsOpnPutPowerOn
Parameter List Type Range Direction Description
LogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by KmsPortGetMaxLogPort.
Return Value Type Range Description
Error short 1 Error flag
1 Error = 0 for success. Nonezero is an error number (see KmsMiscGetLastError).

KmsOpnPutPowerOn takes a logical port ID as a parameter. It performs the steps necessary to apply power to the track.

KmsOpnPutPowerOff
Parameter List Type Range Direction Description
LogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by KmsPortGetMaxLogPort.
Return Value Type Range Description
Error short 1 Error flag
1 Error = 0 for success. Nonezero is an error number (see KmsMiscGetLastError).

KmsOpnPutPowerOff takes a logical port ID as a parameter. It performs the steps necessary to remove power from the track.
APPLICATION PROGRAMMING INTERFACE

Parameter List: Range Direction Description
0 CamOpPutMaxLogPort
1 iLogicsPortID int 1-65535 In Logical port ID
2 Maximum value for this server given by CamOpGetMaxLogPort.

Return Value: Type Range Description
0 Error short 1 Error flag
1 Error = 0 for success. nonzero is an error number

Case 3:06-cv-01905-JSW     Document 47     Filed 06/09/2006     Page 22 of 27

US 6,530,329 B2

-continued

APPLICATION PROGRAMMING INTERFACE

Parameter List: Range Direction Description
0 CamOpPutMaxLogPort
1 iLogicsPortID int 1-65535 In Logical port ID
2 Maximum value for this server given by CamOpGetMaxLogPort.

Return Value: Type Range Description
0 Error short 1 Error flag
1 Error = 0 for success. nonzero is an error number

Case 3:06-cv-01905-JSW     Document 47     Filed 06/09/2006     Page 22 of 27

US 6,530,329 B2

-continued

Parameter List: Range Direction Description
0 CamOpPutMaxLogPort
1 iLogicsPortID int 1-65535 In Logical port ID
2 Maximum value for this server given by CamOpGetMaxLogPort.

Return Value: Type Range Description
0 Error short 1 Error flag
1 Error = 0 for success. nonzero is an error number

Case 3:06-cv-01905-JSW     Document 47     Filed 06/09/2006     Page 22 of 27

US 6,530,329 B2

-continued
APPLICATION PROGRAMMING INTERFACE

-continued

iCommPortID int 1-65535 3 In Physical comm port ID

1. Maximum value for this server given by iCommPortGetMaxLogPorts.
2. See FIG. 6: Controller ID to controller name
3. Maximum value for this server given by iCommPortGetMaxPhysical.

Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

iCommPortGetMaxLogicalPorts takes a logical port ID, a command station type ID, and a physical communications port ID as arguments. It maps iLogicalPortID to iCommPortID for the type of command station specified by iControllerID.

0KCommPortGetMaxLogicalPorts Parameter List Type Range Direction Description
pMaxLogicalPorts int * 1 Out Maximum logical port ID

1 Normally 1-65535. 0 returned on error.

Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

iCommPortGetMaxPhysical takes a pointer to a logical port ID as a parameter. It sets the memory pointed to by pMaxLogicalPorts to the maximum logical port ID.

0KCommPortGetMaxPhysical Parameter List Type Range Direction Description
pMaxPhysical int * 1 Out Maximum physical port ID

1 Normally 1-65535. 0 returned on error.

Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

This section describes the commands that control the command flow to the command station.

0KCommConnect Parameter List Type Range Direction Description
LogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by iCommPortGetMaxLogicalPorts.

Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

iCommPortConnect takes a logical port ID as a parameter. It connects to the server to the specified command station.

0KCommConnect Parameter List Type Range Direction Description
LogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by iCommPortGetMaxLogicalPorts.

Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

This section describes miscellaneous commands that do not fit into the other categories.

0KCommMiscGetErrorMsg Parameter List Type Range Direction Description

5

APPLICATION PROGRAMMING INTERFACE

-continued

Parameter List Type Range Direction Description
DecoderObjectID long 1 In Decoder object ID
1 Opaque object ID handler returned by iCommPortAdd.

Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

KCommCommand takes the decoder object ID as a parameter. It sends all state changes from the server database to the specified locomotive or accessory decoder.

A. Cab Control Commands

This section describes commands that control the cabs attached to a command station.

0KCommCabGetMessage Parameter List Type Range Direction Description
iCabAddress int 1-65535 1 In Cab address

1 Maximum value is command station dependent.
2 Exact return type depends on language. It is Cxing * for C++. Empty string on error.

Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

KCommCabGetMessage takes a cab address and a BSTR as parameters. It sets the memory pointed to by pCmdObj in the present cab message.

0KCommCabPutMessage Parameter List Type Range Direction Description
iCabAddress int 1 In Cab address

1 Maximum value is command station dependent.
2 Exact parameter type depends on language. It is LPCWSTR for C++.

Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

KCommCabPutMessage takes a cab address and a BSTR as parameters. It sets the cab message to bCmdMsg.

0KCommSetCabAddr Parameter List Type Range Direction Description
DecoderObjectID long 1 In Decoder object ID

iCabAddress int 1-65535 2 Out Pointer to Cab address
1 Opaque object ID handler returned by iCommPortAdd.

2 Maximum value is command station dependent.
45 Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

KCommSetCabAddr takes a decoder object ID and a pointer to a cab address as parameters. It sets the memory pointed to by pCmdObj to the address of the cab attached to the specified decoder.

0KCommSetCabAddrToCab Parameter List Type Range Direction Description
DecoderObjectID long 1 In Decoder object ID
iCabAddress int 1-65535 2 In Cab address
1 Opaque object ID handler returned by iCommPortAdd.

2 Maximum value is command station dependent.
45 Return Value Type Range Description
errno short 1 Error flag
1 errno = 0 for success. Nonzero is an error number (see iCommPortGetErrorMsg).

KCommSetCabAddrToCab takes a decoder object ID and cab address as parameters. It attaches the decoder specified by iDecoderID to the cab specified by iCabAddress.
US 6,530,329 B2

-continued

APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Error</th>
<th>int</th>
<th>0-65535</th>
<th>1</th>
<th>Error flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError</td>
<td>0</td>
<td>for success. Nonzero indicates an error.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value</td>
<td>Type</td>
<td>Range</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter List</td>
<td>Type</td>
<td>Range</td>
<td>Direction</td>
<td>Description</td>
</tr>
<tr>
<td>KamMacGetConfigOnline</td>
<td>INT</td>
<td>1-65535</td>
<td>1</td>
<td>Logical port ID</td>
</tr>
<tr>
<td>iSocketPortID</td>
<td>int</td>
<td>2-3</td>
<td>Clock source</td>
<td></td>
</tr>
<tr>
<td>SelectTimeMode</td>
<td>int</td>
<td>0-6</td>
<td>Day of week</td>
<td></td>
</tr>
<tr>
<td>pIDay</td>
<td>int</td>
<td>0-23</td>
<td>Hours</td>
<td></td>
</tr>
<tr>
<td>pHours</td>
<td>int</td>
<td>0-59</td>
<td>Minutes</td>
<td></td>
</tr>
<tr>
<td>pHMinutes</td>
<td>int</td>
<td>3</td>
<td>Out</td>
<td>Fast clock ratio</td>
</tr>
</tbody>
</table>

Max value for this server given by KamPortGetMaxLogPorts.

0 Load from command station and sync server.
1 Load direct from server.
2 Load from cached command copy of station data.
3 Real time clock.

-continued

APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Error</th>
<th>int</th>
<th>0-65535</th>
<th>1</th>
<th>Error flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError</td>
<td>0</td>
<td>for success. Nonzero is an error number.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value</td>
<td>Type</td>
<td>Range</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter List</td>
<td>Type</td>
<td>Range</td>
<td>Direction</td>
<td>Description</td>
</tr>
<tr>
<td>KamMacGetConfigTime</td>
<td>BSTR *</td>
<td>1</td>
<td>Out</td>
<td>Pointer to interface version string</td>
</tr>
</tbody>
</table>

1 Exact return type depends on language. It is C
ting for C++. Empty string on error.

KamMacGetConfigVersion takes an error flag as a parameter.
It returns a BSTR containing the descriptive error
message associated with the specified error flag.

-continued
### Application Programming Interface (Cont'd)

#### Error short 1 Error flag
1. **if** Error = 0 for success. Nonzero is an error number.

- **Parameter List**
  - **Parameter** Type Range Direction Description
  - **iError** int 1-65535 1 In Command station type ID
  - **LogicalPortID** int 0-65535 2 In Logical port ID
  - **pBuffer** int 0-65535 Out Pointer to extended index

2. **See** FIG. 6: Controller ID to controller name

#### Return Value Type Range Description
1. **if** Error = 0 for success. Nonzero is an error number.
2. **See** FIG. 6: Controller ID to controller name

### The terms and expressions which have been employed in the foregoing specification are used therein as terms of description and not of limitation, and there is no intention, in the use of such terms and expressions, of excluding equivalents of the features shown and described or portions thereof, it being recognized that the scope of the invention is defined and limited only by the claims which follow.

- **What is claimed is:**
  1. A method of operating a digitally controlled model railroad comprising the steps of:
     - (a) transmitting a first command from a first program to an interface;
     - (b) transmitting a second command from a second program to said interface;
     - (c) sending third and fourth commands from said interface representative of said first and second commands, respectively, to a digital command station.

2. The method of claim 1, further comprising the steps of:
   - (a) providing an acknowledgment to said first program in response to receiving said first command by said interface prior to sending said third command to said digital command station; and
   - (b) providing an acknowledgment to said second program in response to receiving said second command by said interface prior to sending said fourth command to said digital command station.

3. The method of claim 2, further comprising the steps of:
   - (a) selectively sending said third command to one of a plurality of digital command stations; and
   - (b) selectively sending said fourth command to one of said plurality of digital command stations.

4. The method of claim 3, further comprising the step of receiving command station responses representative of the state of said digitally controlled model railroad from said plurality of digital command stations.

5. The method of claim 4, further comprising the step of comparing said command station responses to previous commands sent to at least one of said plurality of digital command stations to determine which of said previous commands it corresponds with.

6. The method of claim 5, further comprising the steps of:
   - (a) maintaining a sending queue of commands to be transmitted to said plurality of digital command stations; and
   - (b) transmitting at least one of said commands in said sending queue periodically until removed from said sending queue as a result of the comparison of said command station responses to previous commands.

7. The method of claim 6, further comprising the step of updating a database of the state of said digitally controlled model railroad based upon said receiving command station responses representative of said state of said digitally controlled model railroad.

8. The method of claim 7, further comprising the step of providing said acknowledgment to said first program in...
response to receiving said first command by said interface together with state information from said database related to said first command.
9. The method of claim 8 wherein said first command and said third command are the same command, and said second command and said fourth command are the same command.
10. A method of operating a digitally controlled model railroad comprising the steps of:
   (a) transmitting a first command from a first program to an interface; and
   (b) said interface selectively sending a second command representative of said first command to one of a plurality of digital command stations based upon information contained within at least one of said first and second commands.
11. The method of claim 10, further comprising the steps of:
   (a) transmitting a third command from a second program to said interface; and
   (b) said interface selectively sending a fourth command representative of said third command to one of said plurality of digital command stations based upon information contained within at least one of said third and fourth commands.
12. The method of claim 10 wherein said first program and said interface are operating on the same computer.
13. The method of claim 11 wherein said first program, said second program, and said interface are all operating on different computers.
14. The method of claim 10, further comprising the step of providing an acknowledgment to said first program in response to receiving said first command by said interface prior to sending said second command to one of said plurality of said digital command stations.
15. The method of claim 10 wherein said interface communicates in an asynchronous manner with said first program while communicating in a synchronous manner with said plurality of digital command stations.
16. A method of operating a digitally controlled model railroad comprising the steps of:
   (a) transmitting a first command from a first program to an interface;
   (b) transmitting a second command from a second program to said interface; and
   (c) said interface selectively sending a third and fourth command representative of said first command and said second command, respectively, to the same digital command station.
17. The method of claim 16 wherein said interface communicates in an asynchronous manner with said first and second programs while communicating in a synchronous manner with said digital command station.
18. The method of claim 16, further comprising the step of providing an acknowledgment to said first program in response to receiving said first command by said interface prior to sending said third command to said digital command station.
19. A method of operating a digitally controlled model railroad comprising the steps of:
   (a) transmitting a first command from a first program to a first processor; and
   (b) said first processor providing an acknowledgment to said first program indicating that said first command has properly executed prior to execution of commands related to said first command by said digitally controlled model railroad.
20. The method of claim 19, further comprising the step of sending said first command to a second processor which processes said first command into a state suitable for a digital command station.
21. The method of claim 19, further comprising the steps of:
   (a) transmitting a second command from a second program to said first processor; and
   (b) said first processor selectively providing an acknowledgment to said second program indicating that said second command has properly executed prior to execution of commands related to said second command by said digitally controlled model railroad.
22. The method of claim 21, further comprising the steps of:
   (a) sending a third command representative of said first command to one of a plurality of digital command stations based upon information contained within at least one of said first and third commands; and
   (b) sending a fourth command representative of said second command to one of said plurality of digital command stations based upon information contained within at least one of said second and fourth commands.
23. A method of operating a digitally controlled model railroad comprising the steps of:
   (a) transmitting a first command from a first program to an asynchronous command processor;
   (b) said asynchronous command processor providing an acknowledgment to said first program indicating that said first command has properly executed prior to execution of said first command by said digitally controlled model railroad;
   (c) sending said first command to a command queue where said asynchronous command processor considers the intended destination device of said first command; and
   (d) processing said first command by said asynchronous command processor into a suitable format for execution by a digital command station for said digitally controlled model railroad.
24. The method of claim 23 further comprising the steps of:
   (a) receiving responses from said digital command station; and
   (b) updating a first database of the state of said digitally controlled model railroad based upon said responses from said digital command station.
25. The method of claim 24, further comprising the steps of:
   (a) sending a first response to said command queue from said asynchronous command processor where said asynchronous command processor considers said command queue the intended destination device of said first response; and
   (b) processing said first response by said asynchronous command processor into a suitable format for said first program.
26. The method of claim 25, further comprising the step of updating a second database of the state of said digitally controlled model railroad by said asynchronous command processor based upon said first response from said asynchronous command processor.
27. The method of claim 26, further comprising the step of querying said second database by said asynchronous command processor providing said acknowledgment to said first program providing the information requested and not sending said first command to said command queue.
It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 2,
Line 38, change "model railroad In" to -- model railroad. In --