Exhibit B
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. 6,460,467.

Int. Cl. A63H 19/00

U.S. Cl. 105/1.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

5,493,642 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.
5,946,005 A 8/1999 Severson et al.
5,952,797 A 9/1999 Rossler
6,005,406 A * 5/2000 Katzer .................. 105/1.4
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. Cuchinaki, Jr.
Assistant Examiner—Olga Hernandez

Attorney, Agent, or Firm—Cusenoff, Vilhauer, McCaugh & 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. 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.

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. The 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 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 (DCS) 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 personal computer. A standard for the digital command control system is 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.

Digitoy 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 program 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 resistant 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, would 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 provides an acknowledgment 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 occurs in a synchronous manner, such as a first-in-first-out process. The model railroad application on a DCOM communications transport between the client program and the resident external controlling interface is operated in an asynchronous manner, namely providing an acknowledgment thereby releasing the communications transport 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 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 PREFERRED EMBODIMENT

Referring to FIG. 1, a model train control system 10 includes a 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 with the resident external controlling interface 16 responds to the communications transport 12 to exchange information. A description of COM (common object model) and DCOM (distributed common object model) is provided by Chappell 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 resistant 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 controlling interface 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 by 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 observe the state of the 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 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. 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 devices of the model railroad, such as for example, the speed of a train, the direction of a train, whether a draw bridge 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 storage 102 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, and whether or not the command has 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, 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 cue 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 storages, 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 coordinated 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 result 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.
3.1 Introduction
3.2 Data Types
3.3 Commands to access the server configuration variable database
   - KmcVGetValue
   - KmcVPutValue
   - KmcCVGetEnable
   - KmcVPutEnable
   - KmcCVGetName
   - KmcCVGetMinRegister
   - KmcCVGetMaxRegister
3.4 Commands to program configuration variables
   - KmcProgram
   - KmcProgramSetMode
   - KmcProgramGetStatus
   - KmcProgramReadCV
   - KmcProgramCV
   - KmcProgramReadDecoderToDatabase
   - KmcProgramDecoderFromDatabase
3.5 Commands to control all decoder types
   - KmcDecoderGetMaxModels
   - KmcDecoderGetModelName
   - KmcDecoderSetModelToObj
   - KmcDecoderGetMaxAddress
   - KmcDecoderChangeULnNewAddr
   - KmcDecoderMovePort
   - KmcDecoderGetPort
   - KmcDecoderCheckAddrInUse
   - KmcDecoderGetModelFromObj
   - KmcDecoderGetModelFacility
   - KmcDecoderGetObjCount
   - KmcDecoderGetObjectArrayIndex
   - KmcDecoderPutAdd
   - KmcDecoderPutDel
   - KmcDecoderGetMgrName
   - KmcCVGetEnableMode
   - KmcDecoderGetMaxSp
3.6 Commands to control locomotive decoders
   - KmcLingPutSpeed
   - KmcLingGetSp
   - KmcLingGetSpeedSteps
   - KmcLingPutFunction
   - KmcLingGetFunction
   - KmcLingGetFunctionMax
   - KmcLingGetName
   - KmcLingPutName
   - KmcLingGetFunctionName
   - KmcLingGetFunctionName
   - KmcLingGetCV
   - KmcLingGetCVn
   - KmcLingPutDB
   - KmcLingPutDBn
   - KmcLingGetDB
   - KmcLingGetDBn
3.7 Commands to control accessory decoders
   - KmcAaccessGetFunction
   - KmcAaccessGetFunctionAll
   - KmcAaccessPutFunction
   - KmcAaccessPutFunctionAll
   - KmcAaccessFunctionMax
   - KmcAaccessName
   - KmcAaccessPutName
   - KmcAaccessFunctionName
   - KmcAaccessFunctionName
   - KmcAaccessRegFeedback
   - KmcAaccessFeedbackAll
   - KmcAaccessFeedback
   - KmcAaccessRegFeedback
3.8 Commands to control the command station
   - KmcOptPutTurnOnStation
   - KmcOptPutTurnOffStation
   - KmcOptPutStopStation
   - KmcOptPutPowerOn
   - KmcOptPutPowerOff
   - KmcOptPutReset
   - KmcOptPutEmergencyStop
   - KmcOptPut///EmergencyStop

5.9 Commands to configure the command station
   - KmcOptPutConfig
   - KmcOptGetConfig
   - KmcOptPutName
   - KmcOptPutMapController
   - KmcOptPutMaxLogPort
   - KmcOptPutMaxPhysical
5.10 Commands that control command flow to the command station
   - KmcOptConnect
   - KmcOptDisconnect
5.11 Cbb Control Commands
   - KmcOptPutMessage
   - KmcOptPutCbbAdd
   - KmcOptPutCbbAddrToCbb
5.12 Miscellaneous Commands
   - KmcMiscGetErrorMsg
   - KmcMiscGetClockTime
   - KmcMiscGetClockTime
   - KmcMiscGetInterfaceVersion
   - KmcMiscGetVersion
   - KmcMiscGetControllerName
   - KmcMiscGetControllerNameAllPort
   - KmcMiscGetCommandStationValue
   - KmcMiscGetCommandStationValue
   - KmcMiscGetCommandStationIndex
   - KmcMiscGetControllerID
   - KmcMiscGetControllerFacility

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.
   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.

$Devs: $Author: $Revision: $
**APPLICATION PROGRAMMING INTERFACE**

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

*Continued*

**APPLICATION PROGRAMMING INTERFACE**

1. (COM 1 - COM6, LPT1, Others), You are required to map a port to a device to access a command station.
2. Devices start from ID 0 to max ID (FYI, devices do not necessarily have to be serial channel. Always check the name of the device before you use it as well as the maximum number of devices supported.
3. The Command
   - ENG_Port_get MaxPhysical(MaxPhysical, iSerial, iLPEN)
   - Interface + IPanel + Other: It is recommended that you check the command station ID before you use it.
4. Errors - All commands return an error status. If the error value is non-zero, then the error return arguments are invalid. In general, non-zero errors mean command was not executed. To get the error message, you need to call KAM_MiscErrorMessage and supply the error number
5. 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.
6. Addresses are an object reference. To use an address you must add the address to the command station using KamDecoderPortAdd. One of the return values from this operation is an object reference.
7. We need certain variables as global objects, since the information is being used multiple times.
8. Dim ILogicalPort, iController, iComPort
9. Dim ipSerial, iSerialPort, iPortStop, iPortRemains, iPortWatchdog, iPortFlow, iPortData
10. Dim IfLineObject As Long, iDecoderClass As Integer, iDecoderType As Integer
11. Dim iMaxController As Long
12. Dim iMaxLogical As Long, iMaxPhysical As Long, iMaxSerial As Long, iMaxParallel As Long

---

**Form load function**

- Turn of the initial buttons
- Set the interface information

---

**Private Sub Form_Load**

- Dim sTR Vers As String, stCOM As String, strCold As String
- Dim Error As Integer
- Get the interface version information
- SetButtonState (False)
- If (Error) Then
  - MsgBox ("Train Server not loaded. Check DCOM-95")
- iLogicalPort = 0
- LogPort.Caption = iLogicalPort
- ComPort.Caption = "COM + IPanel"
- Controller.Caption = "Unknown"  
Else
  - MsgBox ("(Simulation) (COM1) Train Server -- " & sTR Vers)

---

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

- UNKNOWN
- SIMULATE

---

**LogicalPort = 1 Select Logical port 1 for communications**
- **Controller = 1 Select controller from the list above.**
- **iComPort = 0 Use COM1; 0 means COM1 (Digitalis must use COM1 or COM2)**

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

If (Error) Then
  - MsgBox ("(Simulation) (COM1) Train Server -- " & sTR Vers)

---

**Display the port and controller information**

- Set (Port iPort) = GetMaxPhysical()
- Controller.Caption = "Unknown"
- Set (Port iPort) = GetMaxSerial()
- Dim iPort (0) = "COM port out of range"
- If (Error) Then
  - MsgBox ("(Simulation) (COM1) Train Server -- " & sTR Vers)
APPLICATION PROGRAMMING INTERFACE

("Logical port out of range")
SetError (Error)
End If
  Display values is Throttle.
  LogPort.Caption = LogicalPort
  ComPort.Caption = strCom
  Controller.Caption = strCtrl
End Sub

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

Private Sub Command1_Click()
  'Send the command from the interface to the command
  'station, see the engineObject
  Dim Error, IsSpeed As Integer
  If Not Connect.Enabled Then
    Timetools Interface is a caching interface.
    'This means that you need to set up the CV's or
    'other operations first, then execute the
    'command.
    IsSpeed = Speed.Text
  End If
  Error = EngCmd.KamEngPuFunction(EngineObject, 0, F0.Value)
  If Error = EngCmd.KamEngPuFunction(EngineObject, 1, F1.Value)
  Error = EngCmd.KamEngPuFunction(EngineObject, 2, F2.Value)
  Error = EngCmd.KamEngPuFunction(EngineObject, 3, F3.Value)
  If Error = EngCmd.KamEngPuFunction(EngineObject, 4)
  Error = EngCmd.KamEngPuFunction(EngineObject, 5, F5.Value)
  Error = EngCmd.KamEngPuFunction(EngineObject, 6)
  Error = EngCmd.KamEngPuFunction(EngineObject, 7)
  If Error = 0 Then Error = EngCmd.KamEngCmdCommand(EngineObject)
  SetError (Error)
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_PARITY 2 // Retrans index
  PORT_STOP 3 // Retrans index
  PORT_WATCHDOO 4 // Retrans index
  PORT_FLOW 5 // Retrans index
  PORT_DATABITS 6 // Retrans index
  PORT_DEBUG 7 // Retrans index
  PORT_PARALLEL 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_PARITY 2 // Retrans index
  PORT_STOP 3 // Retrans index
  PORT_WATCHDOO 4 // Retrans index
  PORT_FLOW 5 // Retrans index
  PORT_DATABITS 6 // Retrans index
  PORT_DEBUG 7 // Retrans index
  PORT_PARALLEL 8 // Retrans index
  iError = EngCmd.KamPortPuConfig(LogicalPort, 0,
    iPortRate, 0) // setting PORT_RETRANS
  iError = EngCmd.KamPortPuConfig(LogicalPort, 1,
    iPortRate, 0) // setting PORT_RATE
  iError = EngCmd.KamPortPuConfig(LogicalPort, 2,
    iPortRate, 0) // setting PORT_PARITY
  iError = EngCmd.KamPortPuConfig(LogicalPort, 3,
    iPortRate, 0) // setting PORT_STOP

  iError = EngCmd.KamPortPuConfig(LogicalPort, 4,
    iPortWatchdog, 0) // setting PORT_WATCHDOO
  iError = EngCmd.KamPortPuConfig(LogicalPort, 5,
    iPortWatchdog, 0) // setting PORT_Animado
  iError = EngCmd.KamPortPuConfig(LogicalPort, 6,
    iPortWatchdog, 0) // setting PORT_DATABITS

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

  'Write Display Log Debug
  'File Win Level Value
  ' 1 2 4 8 = 1 7 -> LEVEL1 -- put packets into
  ' 1 2 4 8 = 1 7 -> LEVEL2 -- Status messages
  ' 1 2 4 8 = 1 7 -> LEVEL3 --
  'send to window
  ' 1 2 4 8 = 1 7 -> LEVEL4 -- All system
  ' 1 2 4 8 = 1 7 -> LEVEL5 -- detailed
  ' 1 2 4 8 = 1 7 -> LEVEL6 -- debugging information
  ' 1 2 4 8 = 1 7 -> COMMONLY -- Read comm write
  'comm ports
  If Error = 130 Then
    iDebugMode = 130
    iValue = iValue.Text
    Display value for reference
  Else
    iError = EngCmd.KamPortPuConfig(LogicalPort, 7, iDebug, 0)
  End If
  'Now map the Logical Port, Physical device, Command
  'station and Controller
  iError = EngCmd.KamPortPuConfigure(EngineObject, 0, 0, 0)
  iError = EngCmd.KamPortPuConfigure(EngineObject, 1, 0, 0)
  If (Error) Then
    SetButtonText = False
  Else
    SetButtonText = True
  End If

  'Set the address button
  Private Sub DCCAddr_Click()
    Dim iAddress As Integer
    'All addresses must be match to a logical port to work
    'iDecoderType = 1
    'Set the decoder type to an NMR
    'base_address decoder (1 - 8 msg)
    'iDecoderClass = 1
    'Set the decoder class to Engine
    '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 iAddress.Text = 1 Then
      iStatus = EngCmd.KamDecoderPutAdd(Address.Text,
        LogicalPort, 0, 0, iDecoderType, EngineObject)
    Else
      'End If
  End Sub

***************
APPLICATION PROGRAMMING INTERFACE
***************
US 6,530,329 B2

APPLICATION PROGRAMMING INTERFACE

Private Sub Disconnect_Click()
    Dim Error As Integer
    iError = EngCmd.DisConnect(LogicalPort)
    SetError(iError)
    CloseButtonState = False
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 = EngCmd.GetMinMaxConfError(Msg(iError), "cError")
    ErrorMsg.Caption = sError
    Result.Caption = Status(0)
End Sub

Private Sub SetButtonState(State As Boolean)
    'We set the state of the buttons; either connected
    'or disconnected.
    If (State) Then
        Connect.Enabled = True
        Disconnect.Enabled = False
        OCNCmd.Enabled = True
        OREMCmd.Enabled = True
        DCCAddr.Enabled = True
        UpDownAddress.Enabled = True
    Else
        Connect.Enabled = False
        Disconnect.Enabled = False
        OCNCmd.Enabled = False
        OREMCmd.Enabled = False
        DCCAddr.Enabled = False
        UpDownAddress.Enabled = False
        Throttle.Enabled = False
    End If
End Sub

Private Sub PowerOff_Click()
    Dim iError As Integer
    iError = EngCmd.KamOptPutPowerOff(LogicalPort)
    SetError(iError)
End Sub

Power On function

### APPLICATION PROGRAMMING INTERFACE

1. **fileXGetProcInfo**
   - **Parameter List**
     - `ProcEntry`
     - `ModuleName` string
     - `ProcExit` function
   - **Description**
     - This function returns information about a module's entry and exit points.

2. **GetModuleEntryAddress**
   - **Parameter List**
     - `ModuleName` string
     - `ProcEntry` function
     - `Address` pointer
   - **Description**
     - Obtains the address of a module's entry point.

3. **GetModuleExitAddress**
   - **Parameter List**
     - `ModuleName` string
     - `ProcExit` function
     - `Address` pointer
   - **Description**
     - This function obtains the address of a module's exit point.

4. **fileXGetProcInfo**
   - **Parameter List**
     - `ModuleName` string
     - `ProcEntry` function
     - `ProcExit` function
   - **Description**
     - This function returns additional details about a module's entry and exit points.

5. **fileXGetProcInfo**
   - **Parameter List**
     - `ModuleName` string
     - `ProcEntry` function
     - `ProcExit` function
   - **Description**
     - This function provides further information about module entry and exit points.

6. **GetModuleInfo**
   - **Parameter List**
     - `ModuleName` string
     - `Info` structure
   - **Description**
     - Retrieves detailed information about a module.

7. **fileXGetProcInfo**
   - **Parameter List**
     - `ModuleName` string
     - `ProcEntry` function
     - `ProcExit` function
   - **Description**
     - This function returns information about a module's entry and exit points.

8. **fileXGetProcInfo**
   - **Parameter List**
     - `ModuleName` string
     - `ProcEntry` function
     - `ProcExit` function
   - **Description**
     - This function provides additional details about module entry and exit points.

9. **fileXGetProcInfo**
   - **Parameter List**
     - `ModuleName` string
     - `ProcEntry` function
     - `ProcExit` function
   - **Description**
     - This function returns information about a module's entry and exit points.

10. **fileXGetProcInfo**
    - **Parameter List**
      - `ModuleName` string
      - `ProcEntry` function
      - `ProcExit` function
    - **Description**
      - This function provides additional details about module entry and exit points.

11. **fileXGetProcInfo**
    - **Parameter List**
      - `ModuleName` string
      - `ProcEntry` function
      - `ProcExit` function
    - **Description**
      - This function returns information about a module's entry and exit points.

12. **fileXGetProcInfo**
    - **Parameter List**
      - `ModuleName` string
      - `ProcEntry` function
      - `ProcExit` function
    - **Description**
      - This function provides additional details about module entry and exit points.

13. **fileXGetProcInfo**
    - **Parameter List**
      - `ModuleName` string
      - `ProcEntry` function
      - `ProcExit` function
    - **Description**
      - This function returns information about a module's entry and exit points.

14. **fileXGetProcInfo**
    - **Parameter List**
      - `ModuleName` string
      - `ProcEntry` function
      - `ProcExit` function
    - **Description**
      - This function provides additional details about module entry and exit points.

15. **fileXGetProcInfo**
    - **Parameter List**
      - `ModuleName` string
      - `ProcEntry` function
      - `ProcExit` function
    - **Description**
      - This function returns information about a module's entry and exit points.

16. **fileXGetProcInfo**
    - **Parameter List**
      - `ModuleName` string
      - `ProcEntry` function
      - `ProcExit` function
    - **Description**
      - This function provides additional details about module entry and exit points.

17. **fileXGetProcInfo**
    - **Parameter List**
      - `ModuleName` string
      - `ProcEntry` function
      - `ProcExit` function
    - **Description**
      - This function returns information about a module's entry and exit points.

18. **fileXGetProcInfo**
    - **Parameter List**
      - `ModuleName` string
      - `ProcEntry` function
      - `ProcExit` function
    - **Description**
      - This function provides additional details about module entry and exit points.
**APPLICATION PROGRAMMING INTERFACE**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>decoder_object_ID</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>program_port_ID</td>
<td>int</td>
<td>1-65535</td>
<td>2 In</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Value</th>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pProgMode</td>
<td>int</td>
<td>3 Out</td>
<td>Programming mode</td>
</tr>
</tbody>
</table>

1. opaque object ID handle returned by `KamDecoderPutAdd()`.  
2. Maximum CV is 1024. Maximum CV for this decoder depends on the specified variable (CV) number and the specified variable (CV) number as parameters.

**continued**
### APPLICATION PROGRAMMING INTERFACE (continued)

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iModel</td>
<td>int</td>
<td>1</td>
<td>In</td>
<td>Decoder type ID</td>
</tr>
<tr>
<td>pMaxAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Maximum decoder address</td>
</tr>
</tbody>
</table>

1. Maximum value for this server given by KamDecoderGetMaxModels.
2. Model dependent; 0 returned on error.

### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pModelId</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pBaseAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Pointer to decoder object ID</td>
</tr>
</tbody>
</table>

5. KamPortGetMaxLogPorts.

### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pModelId</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pBaseAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Pointer to decoder object ID</td>
</tr>
</tbody>
</table>

20. KamDecoderGetModelFromObj.

### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pModelId</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pBaseAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Pointer to decoder object ID</td>
</tr>
</tbody>
</table>

25. KamDecoderGetModelFacility.

### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pModelId</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pBaseAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Pointer to decoder object ID</td>
</tr>
</tbody>
</table>


### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pModelId</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pBaseAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Pointer to decoder object ID</td>
</tr>
</tbody>
</table>

35. KamDecoderGetFacility.

### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pModelId</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pBaseAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Pointer to decoder object ID</td>
</tr>
</tbody>
</table>

40. KamDecoderGetFacility.

### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pModelId</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pBaseAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Pointer to decoder object ID</td>
</tr>
</tbody>
</table>

45. KamDecoderGetFacility.

### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pModelId</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pBaseAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Pointer to decoder object ID</td>
</tr>
</tbody>
</table>

50. KamDecoderGetFacility.

### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pModelId</td>
<td>long</td>
<td>1 In</td>
<td>Decoder object ID</td>
<td></td>
</tr>
<tr>
<td>pBaseAddress</td>
<td>int</td>
<td>*</td>
<td>2 Out</td>
<td>Pointer to decoder object ID</td>
</tr>
</tbody>
</table>

55. KamDecoderGetFacility.
Case 3:06-cv-01905-JSW     Document 96     Filed 09/11/2006     Page 17 of 27

US 6,530,329 B2

-continued

APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index int</td>
<td>2</td>
<td>In</td>
<td>Decoder array index</td>
</tr>
<tr>
<td>iDecoderClass int</td>
<td>2</td>
<td>In</td>
<td>object class of decoder</td>
</tr>
<tr>
<td>pDecoderObjectID long * 3</td>
<td>Out</td>
<td>Pointer to decoder object ID</td>
<td></td>
</tr>
</tbody>
</table>

1. 0 to (KamDecoderGetAddressCount – 1).
2. 1 - DECODER_ENGINE_TYPE, 2 - DECODER_SWITCH_TYPE, 3 - DECODER_SENSOR_TYPE.
3. Opaque object ID handle returned by KamDecoderPutAdd.

Return Value Type Range Description
iError short 1 Error flag
iError 0 = success. Nonzero is an error number (see KamMaccGetErrorMag).
KamDecoderGetObjCount takes a decoder class and a pointer to an address count as parameters. It sets the memory pointed to by pAddressCount to the count of active decoders of the type given by iDecoderClass.
KamDecoderGetObjAddIndex

Parameter List | Type     | Range    | Description                  |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iAddress int</td>
<td>1</td>
<td>In</td>
<td>Decoder address</td>
</tr>
<tr>
<td>iLogicalCmdPortID int</td>
<td>1</td>
<td>65535 2</td>
<td>Logical command port ID</td>
</tr>
<tr>
<td>iLogicalProgPortID int</td>
<td>1</td>
<td>65535 2</td>
<td>Logical programming port ID</td>
</tr>
<tr>
<td>iClearState int</td>
<td>3</td>
<td>In</td>
<td>Clear state flag</td>
</tr>
<tr>
<td>iModel int</td>
<td>4</td>
<td>In</td>
<td>Decoder model type ID</td>
</tr>
<tr>
<td>pDecoderObjectID long * 5</td>
<td>Out</td>
<td>Decoder</td>
<td></td>
</tr>
</tbody>
</table>

1. 1-127 for short locomotive addresses. 1-1023 for long locomotive decoder, 0-511 for accessory decoders. Maximum value for this server given by KamPortGetModuleNamePort.
3. 0 - retain state, 1 - clear state.
4. Maximum value for this server given by KamDecoderGetMaxModels.
5. Opaque object ID handle. The object ID is used to reference the decoder.

Return Value Type Range Description
iError short 1 Error flag
iError 0 = success. Nonzero is an error number (see KamMaccGetErrorMag).
KamDecoderPutAdd takes a decoder object ID, command logical port, programming logical port, clear flag, decoder model ID, and a pointer to a decoder object ID as parameters. It creates a new locomotive object in the locomotive database and sets the memory pointed to by pDecoderObjectID to the decoder object ID used by the server as a key.
KamDecoderPutAdd

Parameter List | Type     | Range    | Description                  |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iAddress int</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>iClearState int</td>
<td>2</td>
<td>In</td>
<td>Clear state flag</td>
</tr>
</tbody>
</table>

1. Opaque object ID handle returned by KamDecoderPutAdd.
2. 0 - retain state, 1 - clear state.

Return Value Type Range Description
iError short 1 Error flag
iError 0 = success. Nonzero is an error number (see KamMaccGetErrorMag).
KamDecoderPutDel takes a decoder object ID and clear flag as parameters. It deletes the locomotive object specified by DecoderObjectID from the locomotive database.
KamDecoderPutDel

-continued

APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectID long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>pBarMagName BSTR * 2</td>
<td>Out</td>
<td>Pointer to manufacturer name</td>
<td></td>
</tr>
</tbody>
</table>

1. Opaque object ID handle returned by KamDecoderPutAdd.
2. Exact return type depends on language. It is Casting * for C++. Empty string on error.

Return Value Type Range Description
iError short 1 Error flag
iError 0 = success. Nonzero is an error number (see KamMaccGetErrorMag).
KamDecoderGetBarMagName takes a decoder object ID and a pointer to a manufacturer name string as parameters. It sets the memory pointed to by pBarMagName to the name of the decoder manufacturer.
KamDecoderGetBarPowerMode

Parameter List | Type     | Range    | Description                  |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectID long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>pBarPowerMode BSTR * 2</td>
<td>Out</td>
<td>Pointer to decoder power mode</td>
<td></td>
</tr>
</tbody>
</table>

1. Opaque object ID handle returned by KamDecoderPutAdd.
2. Exact return type depends on language. It is Casting * for C++. Empty string on error.

Return Value Type Range Description
iError short 1 Error flag
iError 0 = success. Nonzero is an error number (see KamMaccGetErrorMag).
KamDecoderGetBarPowerMode takes a decoder object ID and a pointer to the power mode string as parameters. It sets the memory pointed to by pBarPowerMode to the decoder power mode.
KamDecoderGetBarSpeed

Parameter List | Type     | Range    | Description                  |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectID long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>pBarSpeedStep int * 2</td>
<td>Out</td>
<td>Pointer to max speed step</td>
<td></td>
</tr>
</tbody>
</table>

1. Opaque object ID handle returned by KamDecoderPutAdd.
2. Exact return type depends on language. It is Casting * for C++. Empty string on error.

Return Value Type Range Description
iError short 1 Error flag
iError 0 = success. Nonzero is an error number (see KamMaccGetErrorMag).
KamDecoderGetBarSpeed takes a decoder object ID and a pointer to the maximum supported speed as parameters. It sets the memory pointed to by pBarSpeedStep to the maximum speed supported by the decoder.
KamDecoderGetSpeed

Parameter List | Type     | Range    | Description                  |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderObjectID long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>pBarSpeedStep int * 2</td>
<td>Out</td>
<td>Pointer to locomotive speed</td>
<td></td>
</tr>
</tbody>
</table>

1. Opaque object ID handle returned by KamDecoderPutAdd.
2. Exact return type depends on whether the decoder is set to 14, 18, or 128 speed steps and matches the values defined by NMRA 89.2 and RP 9.2.1, 0 is stop and 1 is emergency stop for all modes.
3. Forward is boolean TRUE and reverse is boolean FALSE.

Return Value Type Range Description
iError short 1 Error flag
iError 0 = success. Nonzero is an error number (see KamMaccGetErrorMag).
KamDecoderGetSpeed takes a decoder object ID and clear flag as parameters. It deletes the locomotive object specified by DecoderObjectID from the locomotive database.
KamDecoderGetSpeedDel
**APPLICATION PROGRAMMING INTERFACE**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Parameters</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError short</td>
<td>1 Error flag</td>
<td>1</td>
<td>1</td>
<td>in</td>
<td>error flag</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetSpeed takes the decoder object ID and pointers to locations to store the locomotive speed and direction as parameters. It sets the memory pointed to by iSpeed to the locomotive speed and the memory pointed to by iDirection to the locomotive direction.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps and matches the values defined by NMRA S02.2 and RP 9.2.1. 0 is stop and 1 is emergency stop for all modes. 0.0 is false, TRUE is reverse and value is FALSE.</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetSpeed 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 KamCmdCommand command. Speed is set to the maximum possible for the decoder if iSpeed exceeds the decoder range.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetSpeedSteps 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.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetSpeedSteps 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 until execution of the KamCmdCommand command. KamEngGetMaxSpeed returns the maximum possible speed for the decoder. An error is generated if an attempt is made to set the speed beyond this value.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetFunction takes the decoder object ID, and optional function ID as parameters. It sets the memory pointed to by iFunction to the specified function state.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetFunction takes the decoder object ID, a function ID, and a pointer to the location to store the specified function state as parameters. It sets the memory pointed to by iFunction to the specified function state.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetFunction takes the decoder object ID, a function ID, and a new function state as parameters. It sets the specified locomotive database function state to iFunction. Note: This command only changes the locomotive database. The data is not sent to the decoder until execution of the KamCmdCommand command.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetFunction takes the decoder object ID, a function ID, and a pointer to the new function state as parameters. It sets the memory pointed to by iFunction to the specified function state for the specified decoder.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetFunction takes a decoder object ID and a pointer to the maximum function ID as parameters. It sets the memory pointed to by iMaxFunction to the maximum possible function number for the specified decoder.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetFunction takes a decoder object ID and a new function ID as parameters. It sets the memory pointed to by iMaxFunction to the specified function state for the specified decoder.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
<tr>
<td>1 0</td>
<td>0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamEngGetFunction takes a decoder object ID and a pointer to the new function state as parameters. It sets the memory pointed to by iMaxFunction to the specified function state for the specified decoder.</td>
<td>Decoder object ID</td>
<td>in</td>
<td>14, 18, 128 speed steps</td>
<td>in</td>
</tr>
</tbody>
</table>
APPLICATION PROGRAMMING INTERFACE

KanEngPutName takes a decoder object ID and a BSTR as parameters, it sets the symbolic loco name to bsEngName.

KanEngGetFunctionName
Parameter List
Function ID
bsFnNameString
BSTR
Surname to function name

Opaque object ID handle returned by KanDecodePutAdd.

If FL is 0, FlsFIs are 1-8 respectively. Maximum for this decoder is given by KanEngGetFunctionMax. Exact return type depends on language, it is casting for C++. Empty string on error.

Return Value
Type
Range
Description

Error short
1
Error flag
1
If Error = 0 for success. Nonzero is an error number
Area.SetMaxErrorMsg.
KanEngGetFunctionName takes a decoder object ID, function ID, and a pointer to the function name as parameters. It sets the memory pointed to by bsFnNameString to the symbolic name of the specified function.

KanEngGetFunctionName
Parameter List
Type
Range
Direction
Description
DecoderObjectID
long
1
In
Decoder object ID
FunctionID
int
0-8
2
In
Function ID number
bsFnNameString
BSTR
3
Out
Pointer to function name

Opaque object ID handle returned by KanDecodePutAdd.

If FL is 0, FlsFIs are 1-8 respectively. Maximum for this decoder is given by KanEngGetFunctionMax.

Exact parameter type depends on language. It is LPCSTR for C++.

Return Value
Type
Range
Description

Error short
1
Error flag
1
If Error = 0 for success. Nonzero is an error number
Area.SetMaxErrorMsg.
KanEngGetFunctionName takes a decoder object ID, function ID, and a BSTR as parameters. It sets the specified symbolic function name to bsFnNameString.

KanEngGetFunctionMax
Parameter List
Type
Range
Direction
Description
DecoderObjectID
long
1
In
Decoder object ID
pmMaxCnsist
int
2
Out
Pointer to max consist number

Opaque object ID handle returned by KanDecodePutAdd.

Command station dependent.

Return Value
Type
Range
Description

Error short
1
Error flag
1
If Error = 0 for success. Nonzero is an error number
Area.SetMaxErrorMsg.
KanEngGetFunctionMax takes the decoder object ID and a pointer to a location to store the maximum consist as parameter. It sets the location pointed to by pmMaxCnsist 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.

KanEngGetConsistParent
Parameter List
Type
Range
Direction
Description
IDCCParseObject
long
1
In
Parent decoder object ID

Opaque object ID handle returned by KanDecodePutAdd.

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

Return Value
Type
Range
Description

Error short
1
Error flag
1
If Error = 0 for success. Nonzero is an error number
Area.SetMaxErrorMsg.
KanEngGetConsistParent takes the parent object ID and an alias address as parameters. It makes the decoder specified by IDCCParseObject the consist parent referred to by IDCCAliasAddr. Note that this command is designed for command station consisting. CV consisting is handled using the CV commands. If a new parent is defined for a consist, the old parent becomes a child in the consist.

To delete a parent in a consist without deleting the consist, you must add a new parent then delete the old parent using KanEngPutConsistRemoveObj.

KanEngPutConsistChild
Parameter List
Type
Range
Direction
Description
IDCCParseObject
long
1
In
Parent decoder object ID

Opaque object ID handle returned by KanDecodePutAdd.

Return Value
Type
Range
Description

Error short
1
Error flag
1
If Error = 0 for success. Nonzero is an error number
Area.SetMaxErrorMsg.
KanEngPutConsistChild takes the decoder parent object ID and decoder object ID as parameters. It assigns the decoder specified by IDCCParseObject to the consist identified by IDCCParseObject. Note that this command is designed for command station consisting. CV consisting is handled using the CV commands. Note: This command is invalid if the parent has not been set previously using KanEngPutConsistParent.

KanEngPutConsistRemoveObj
Parameter List
Type
Range
Direction
Description
IDCCParseObject
long
1
In
Decoder object ID

Opaque object ID handle returned by KanDecodePutAdd.

Return Value
Type
Range
Description

Error short
1
Error flag
1
If Error = 0 for success. Nonzero is an error number
Area.SetMaxErrorMsg.
KanEngPutConsistRemoveObj takes the decoder object ID as a parameter. It removes the decoder specified by IDCCParseObject from the consist. Note that this command is designed for command station consisting. CV consisting is handled using the CV commands. Note: If the parent is removed, all children are removed also.

A. Commands to control accessory decoders

This section describes the commands that control accessory decoders. These commands control things such as accessory decoder activation state. For efficiency, a copy of all the engine variables such speed is stored in the server. Commands such as KanAccGetFunction communicate only with the server, not the actual decoder. You should first make any changes to the server copy of the engine variables. You can send all changes to the engine using the KanCmdCommand command.

KanAccGetFunction
Parameter List
Type
Range
Direction
Description
DecoderObjectID
long
1
In
Decoder object ID
FunctionID
int
-31
3
In
Function ID number
IpFunction
int
3
Out
Pointer to function value

Opaque object ID handle returned by KanDecodePutAdd.

2. Maximum for this decoder is given by KanAccGetFunctionMax.

Function active is boolean TRUE and inactive is boolean FALSE.

Return Value
Type
Range
Description

Error short
1
Error flag
1
If Error = 0 for success. Nonzero is an error number
Area.SetMaxErrorMsg.
KanAccGetFunction takes the decoder object ID, a function ID, and a pointer to the location to store the specified function state as parameters. It sets the memory pointed to by IpFunction to the specified function state.

KanAccGetFunctionAll
Parameter List
Type
Range
Direction
Description
DecoderObjectID
long
1
In
Decoder object ID
IpValue
int
2
Out
Function bit mask
29

APPLICATION PROGRAMMING INTERFACE

Opaque object ID handle returned by KamDecoderPutAdd.

Each bit represents a single function state.

Maximum for this decoder is given by KamAccGetFunctionMax.

Return Value Type Range Description

iError short 1 Error flag

1 iError = 0 for success. Nonzero is an error number

0 KamAccPutFunction

Parameter List Type Range Direction Description

DecoderObjectID long 1 In Decoder object ID

FunctionID int 0-31 2 In Function ID number

iFunction int 3 In Function value

1 Opaque object ID handle returned by KamDecoderPutAdd.

Maximum for this decoder is given by KamAccGetFunctionMax.

Function active is boolean TRUE and inactive is boolean FALSE.

Return Value Type Range Description

iError short 1 Error flag

1 iError = 0 for success. Nonzero is an error number

0 KamAccPutFunctionAll

Parameter List Type Range Direction Description

DecoderObjectID long 1 In Decoder object ID

iValue int 2 In Pointer to function state array

1 Opaque object ID handle returned by KamDecoderPutAdd.

Each bit represents a single function state.

Maximum for this decoder is given by KamAccGetFunctionMax.

Return Value Type Range Description

iError short 1 Error flag

1 iError = 0 for success. Nonzero is an error number

0 KamAccGetFunctionMax

Parameter List Type Range Direction Description

DecoderObjectID long 1 In Decoder object ID

iMaxFunction int 0-31 2 Out Pointer to maximum function number

1 Opaque object ID handle returned by KamDecoderPutAdd.

Maximum for this decoder is given by KamAccGetFunctionMax.

Return Value Type Range Description

iError short 1 Error flag

1 iError = 0 for success. Nonzero is an error number

0 KamAccGetMax

Parameter List Type Range Direction Description

DecoderObjectID long 1 In Decoder object ID

phMaxFunctionString BSTR * 2 Out Access name

1 Opaque object ID handle returned by KamDecoderPutAdd.

-continued
APPLICATION PROGRAMMING INTERFACE

(see KamMiscGetErrorMsg).
KamAccelRegFeedbackAll takes a decoder object ID, node name string, and function ID, as parameters. It returns interest in the function given by (FunctionID) by the method given by the node name string (NodeName).

1. When KbAccNode identifies the server application and method to call if the function changes state. Its format is:

   "[(Server)] ([App]) ([Method])" where (Server) is the server name, (App) is the application name, and (Method) is the method name.

2. Exact parameter type depends on language. It is LPCSTR for C++.

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

KamAccelRegFeedbackAll takes a decoder object ID, node name string, and function ID, as parameters. It returns interest in all functions by the method given by the node name string (NodeName). KbAccNode identifies the server application and method to call if the function changes state. Its format is:

   "[(Server)] ([App]) ([Method])" where (Server) is the server name, (App) is the application name, and (Method) is the method name.

2. Exact parameter type depends on language. It is LPCSTR for C++.

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

KamAccelRegFeedbackAll takes a decoder object ID, node name string, and function ID, as parameters. It returns interest in all functions by the method given by the node name string (NodeName). KbAccNode identifies the server application and method to call if the function changes state. Its format is:

   "[(Server)] ([App]) ([Method])" where (Server) is the server name, (App) is the application name, and (Method) is the method name.

2. Exact parameter type depends on language. It is LPCSTR for C++.

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

KamAccelRegFeedbackAll takes a decoder object ID, node name string, and function ID, as parameters. It returns interest in all functions by the method given by the node name string (NodeName). KbAccNode identifies the server application and method to call if the function changes state. Its format is:

   "[(Server)] ([App]) ([Method])" where (Server) is the server name, (App) is the application name, and (Method) is the method name.

2. Exact parameter type depends on language. It is LPCSTR for C++.

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

KamAccelRegFeedbackAll takes a decoder object ID, node name string, and function ID, as parameters. It returns interest in all functions by the method given by the node name string (NodeName). KbAccNode identifies the server application and method to call if the function changes state. Its format is:

   "[(Server)] ([App]) ([Method])" where (Server) is the server name, (App) is the application name, and (Method) is the method name.

2. Exact parameter type depends on language. It is LPCSTR for C++.

   Return Value Type Range Description
   Error short 1 Error flag
   1. If error = 0 for success. Nonzero is an error number.
<table>
<thead>
<tr>
<th>Port Name</th>
<th>Configuration Type Index</th>
<th>Description</th>
</tr>
</thead>
</table>
|...(continued)

35

APPLICATION PROGRAMMING INTERFACE

-continued

iCommPortID int 1-65535 In Physical comm port ID

1 Maximum value for this server given by
KamPortGetMaxLogPorts.
2 See FIG. 6: Controller ID to controller name
mapping for values. Maximum value for this server is
given by KamMacMaxControllerID.
3 Maximum value for this server given by
KamPortGetMaxPhysical.

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

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

0KamPortGetMaxLogPots
Parameter List Type Range Direction Description
pMaxLogPots int * 1 Out Maximum logical port ID

1 Normally 1-65535. 0 returned on error.
Return Value Type Range Description
iError short 1 Error flag
1 iError = 0 for success. Nonzero is an error number
(see KamMacGetErrorMsg).

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

0KamPortGetMaxPysical
Parameter List Type Range Direction Description
pMaxPhysical int * 1 Out Maximum physical port ID
pMaxSerial int * 1 Out Maximum serial port ID
pMaxParallel int * 1 Out Maximum parallel port ID

1 Normally 1-65535. 0 returned on error.
Return Value Type Range Description
iError short 1 Error flag
1 iError = 0 for success. Nonzero is an error number
(see KamMacGetErrorMsg).

KamPortGetMaxPhysical takes a pointer to the number of
physical ports, the number of serial ports, and the
number of parallel ports as parameters. It sets the
memory pointed to by the parameters to the associated
values.

A. Commands that control command flow to the command station.

1 This section describes the commands that
control the command flow to the command station. These
commands do things such as connecting and disconnecting
from the command station.

0KamCommConnect
Parameter List Type Range Direction Description
iLogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by
KamPortGetMaxLogPots.

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

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

0KamCommDisconnect
Parameter List Type Range Direction Description
iLogicalPortID int 1-65535 1 In Logical port ID
1 Maximum value for this server given by
KamPortGetMaxLogPots.

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

KamCommDisconnect takes a logical port ID as a parameter. It
disconnects the server to the specified command station.

36

APPLICATION PROGRAMMING INTERFACE

-continued

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

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

KamDecoderPutAddToCab takes a decoder object ID and a
point in to the address of the specified decoder.

Parameter List Type Range Direction Description
iDecoderObjectID long 1 In Decoder object ID
iCabAddress int 1-65535 2 In Pointer to Cab address
1 Opaque object ID handler returned by
KamDecoderPutAdd.

2 Maximum value is command station dependent.

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

KamCabPutAddToCab takes a decoder object ID and a
cab address as parameters. It attaches the decoder
specified by iDecoderObjectID to the cab specified by iCabAddress.

1 Miscellaneous Commands

A. Miscellaneous Commands

This section describes miscellaneous commands that
do not fit into the other categories.
### APPLICATION PROGRAMMING INTERFACE

```plaintext
|-----------------------------|-------------------------------------------------|
```

#### 37

- **Error**: int 0-65535 1 In Error flag
  - **iError** = 0 for success. Nonzero indicates an error.
  - **Return Value** Type Range Description
  - **Parameters**: List
    - **BSTR** Parameter List
    - **Parameter List** Type Range Direction Description
      - **LogicalPortID** int 1-65535 1 In Logical port ID
      - **SelectTimeMode** int 2 In Clock source
      - **pIDay** int * 0-6 Out Day of week
      - **pIMinutes** int * 0-59 Out Minutes
      - **pIS_ratio** int * 3 Out Fast clock ratio
    - **Maximum value for this server given by KamPortGetMaxLogPort**: 2
    - **0**: Load from command station and sync server.
    - **1**: Load from direct server. 2: Load from cached server copy of command station time.
    - **Real time clock mto**.
    - **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.

- **KamMacGetClockTime**
  - **Parameter List** Type Range Direction Description
    - **LogicalPortID** int 1-65535 1 In Logical port ID
    - **bDay** int 0-6 In Day of week
    - **bIMinutes** int 0-59 In Minutes
    - **bIS_ratio** int 3 In Fast clock ratio
    - **Maximum value for this server given by KamPortGetMaxLogPort**: 1
    - **Real time clock mto**.
    - **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.

- **KamMacGetInterfaceVersion**
  - **Parameter List** Type Range Direction Description
    - **LogicalPortID** int 1-65535 1 In Logical port ID
    - **bValue** int 0-65535 Out Command station value
    - **0**: KamMacGetCommandStationIndex. **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.

- **KamMacSaveData**
  - **Parameter List** Type Range Direction Description
    - **LogicalPortID** int 1-65535 1 In Logical port ID
    - **bValue** int 0-65535 Out Command station value
    - **0**: KamMacGetCommandStationIndex. **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.
```

### APPLICATION PROGRAMMING INTERFACE

```plaintext
|-----------------------------|-------------------------------------------------|
```

#### 38

- **Error**: int 0-65535 1 In Error flag
  - **iError** = 0 for success. Nonzero indicates an error.
  - **Return Value** Type Range Description
  - **Parameters**: List
    - **BSTR** Parameter List
    - **Parameter List** Type Range Direction Description
      - **ControllerID** int 1-65535 1 In Command station type ID
      - **pName** BSTR * 2 Out Command station type name
    - **Maximum value for this server given by KamPortGetMaxLogPort**: 1
    - **0**: Load from command station and sync server.
    - **1**: Load from direct server. 2: Load from cached server copy of command station time.
    - **Real time clock mto**.
    - **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.

- **KamMacGetControllerNameAtPort**
  - **Parameter List** Type Range Direction Description
    - **LogicalPortID** int 1-65535 1 In Logical port ID
    - **pName** BSTR * 2 Out Command station type name
    - **Maximum value for this server given by KamPortGetMaxLogPort**: 1
    - **0**: Load from command station and sync server.
    - **1**: Load from direct server. 2: Load from cached server copy of command station time.
    - **Real time clock mto**.
    - **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.

- **KamMacGetControllerName**
  - **Parameter List** Type Range Direction Description
    - **LogicalPortID** int 1-65535 1 In Logical port ID
    - **pName** BSTR * 2 Out Command station type name
    - **Maximum value for this server given by KamPortGetMaxLogPort**: 1
    - **0**: Load from command station and sync server.
    - **1**: Load from direct server. 2: Load from cached server copy of command station time.
    - **Real time clock mto**.
    - **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.

- **KamMacGetCommandStationValue**
  - **Parameter List** Type Range Direction Description
    - **ControllerID** int 1-65535 1 In Command station type ID
    - **LogicalPortID** int 1-65535 2 In Logical port ID
    - **bValue** int * 0-65535 Out Command station value
    - **0**: KamMacGetCommandStationIndex. **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.

- **KamMacGetCommandStationValue**
  - **Parameter List** Type Range Direction Description
    - **ControllerID** int 1-65535 1 In Command station type ID
    - **LogicalPortID** int 1-65535 2 In Logical port ID
    - **bValue** int * 0-65535 Out Command station value
    - **0**: KamMacGetCommandStationIndex. **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.

- **KamMacGetCommandStationValue**
  - **Parameter List** Type Range Direction Description
    - **ControllerID** int 1-65535 1 In Command station type ID
    - **LogicalPortID** int 1-65535 2 In Logical port ID
    - **bValue** int * 0-65535 Out Command station value
    - **0**: KamMacGetCommandStationIndex. **Return Value** Type Range Description
      - **bError** short 1 Error flag
        - **iError** = 0 for success. Nonzero is an error number.
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; and
   (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 so as 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) retransmitting 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 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 synchronous 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 synchronous command processor where said synchronous 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 synchronous 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.

* * * * *
UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,530,329 B2
DATED : March 11, 2003
INVENTOR(S) : Katzer

Page 1 of 1

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 --

Signed and Sealed this

Fifteenth Day of March, 2005

[Signature]

JON W. DUDAS
Director of the United States Patent and Trademark Office