Appendix B

US006530329B2

(12) **United States Patent**
Katzer

(10) **Patent No.: US 6,530,329 B2**
(45) **Date of Patent: *Mar. 11, 2003**

(54) **MODEL TRAIN CONTROL SYSTEM**

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

(73) 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.

(21) Appl. No.: **10/124,878**

(22) Filed: **Apr. 17, 2002**

(65) **Prior Publication Data**

US 2002/0170458 A1 Nov. 21, 2002

Related U.S. Application Data

(63) Continuation of application No. 09/858,222, filed on Apr. 17, 2002, now Pat. No. 6,460,467.

(51) **Int. Cl.**[7] **A63H 19/00**

(52) **U.S. Cl.** .... **105/1.5**; 246/167 R; 246/197; 246/62

(58) **Field of Search** .... 105/1.5, 1.4, 29.2; 246/187 A, 167 R, 197, 62; 701/20

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,944,986 A 3/1976 Staples
3,976,272 A 8/1976 Murray et al.
4,307,302 A 12/1981 Russell
4,853,883 A * 8/1989 Nickles et al. 348/121
5,072,900 A 12/1991 Malon
5,475,818 A * 12/1995 Molyneaux et al. 701/20
5,493,642 A 2/1996 Dunsmuir et al.
5,638,522 A 6/1997 Dunsmuir et al.
5,681,015 A * 10/1997 Kull .... 246/167 R
5,696,689 A 12/1997 Okumura et al.
5,787,371 A * 7/1998 Balukin et al. .... 246/187 A
5,828,979 A 10/1998 Ploivka et al.
5,896,017 A 4/1999 Severson et al.
5,940,005 A 8/1999 Severson et al.
5,952,797 A 9/1999 Rossler
6,065,406 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.

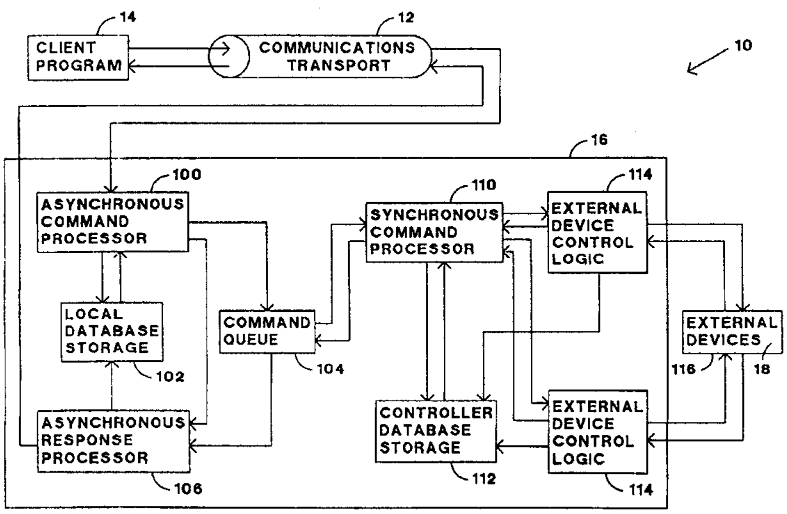
* cited by examiner

Primary Examiner—William A. Cuchlinski, Jr.
Assistant Examiner—Olga Hernandez
(74) *Attorney, Agent, or Firm*—Chernoff, Vilhauer, McClung & Stenzel, LLP

(57) **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



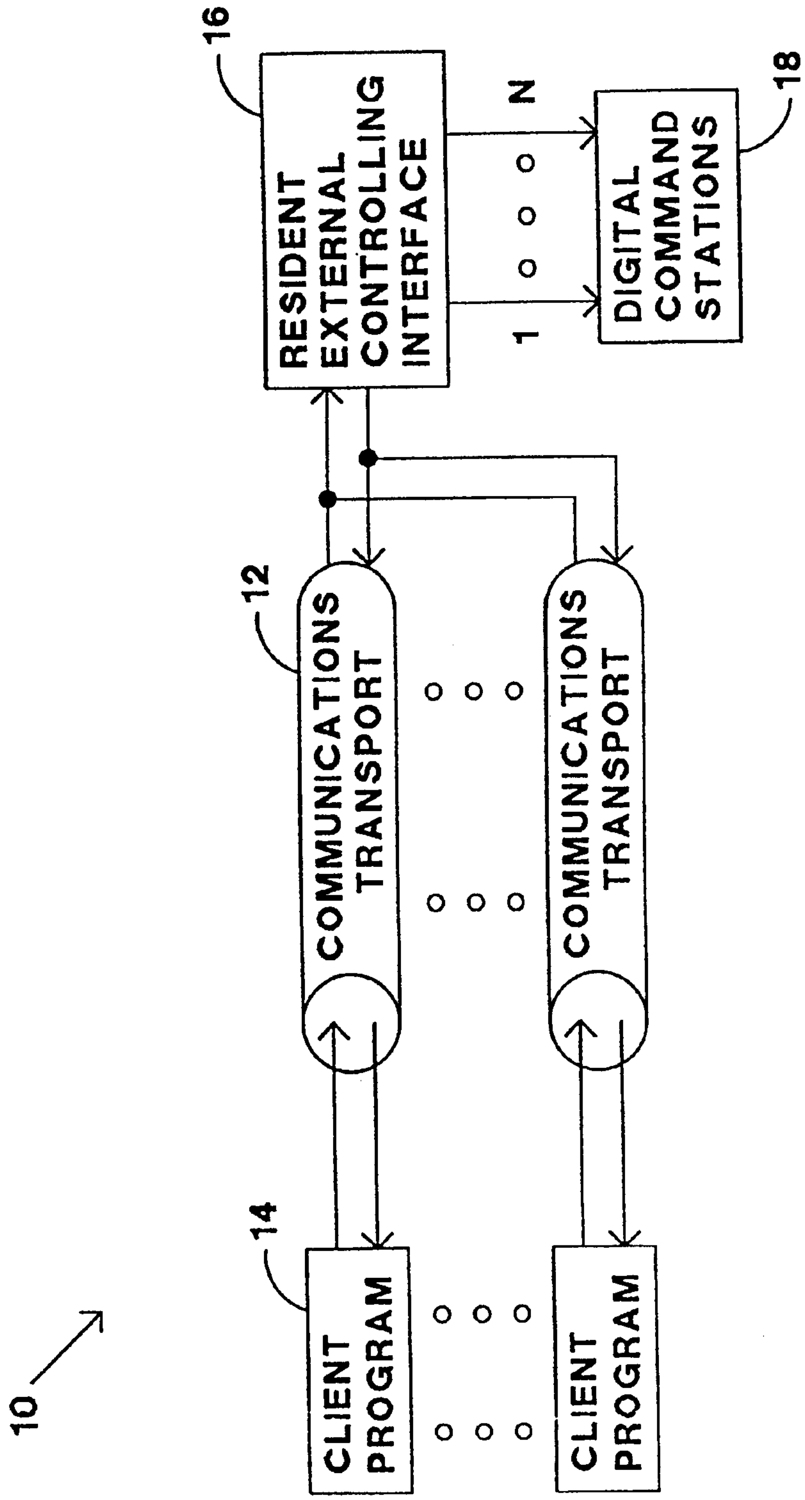

16
RESIDENT EXTERNAL CONTROLLING INTERFACE
N
1
DIGITAL COMMAND STATIONS
18
12
COMMUNICATIONS TRANSPORT
COMMUNICATIONS TRANSPORT
14
CLIENT PROGRAM
CLIENT PROGRAM
10

FIG. 1

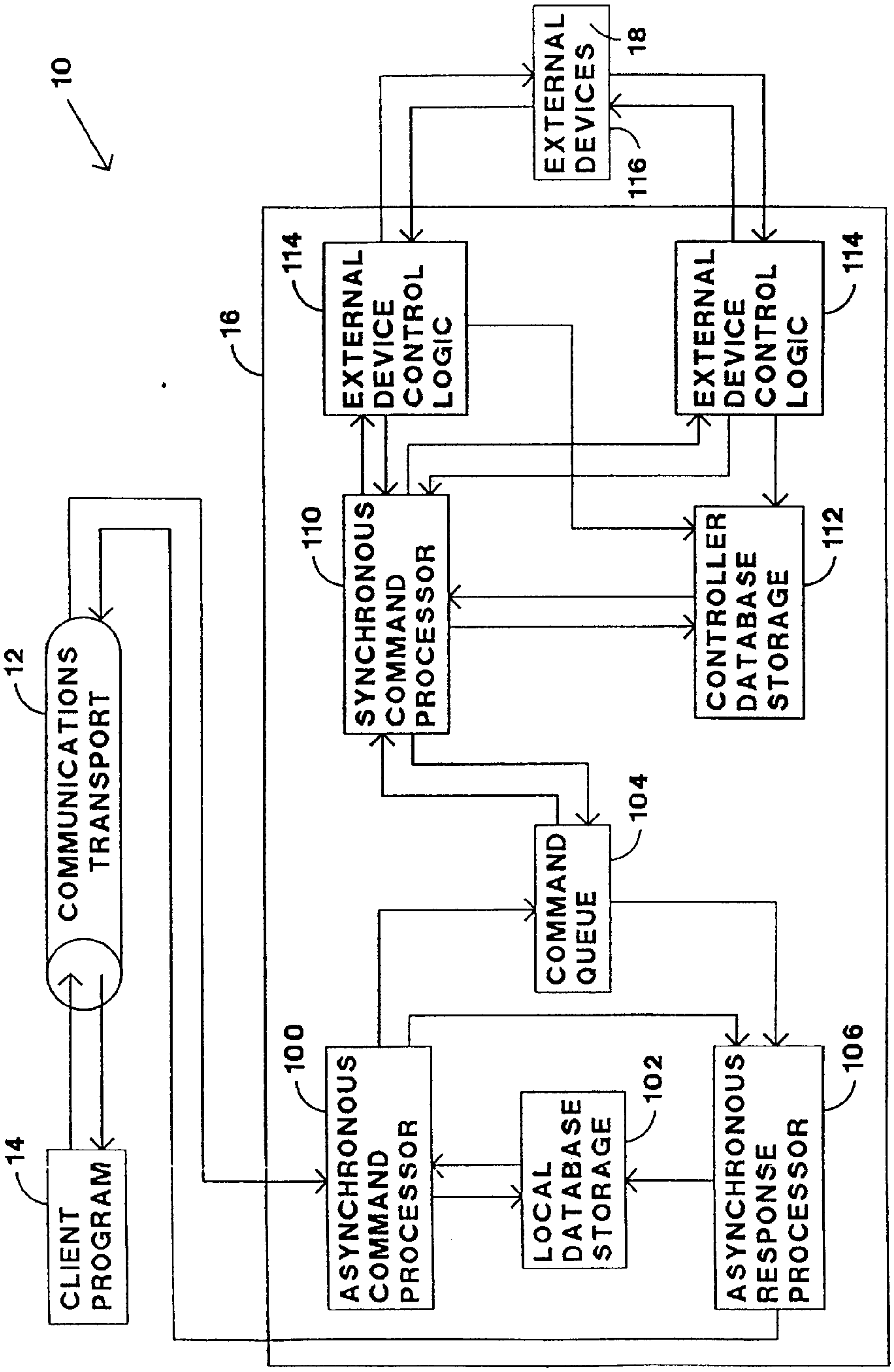

10
CLIENT PROGRAM
14
COMMUNICATIONS TRANSPORT
12
16
ASYNCHRONOUS COMMAND PROCESSOR
100
LOCAL DATABASE STORAGE
102
ASYNCHRONOUS RESPONSE PROCESSOR
106
COMMAND QUEUE
104
SYNCHRONOUS COMMAND PROCESSOR
110
CONTROLLER DATABASE STORAGE
112
EXTERNAL DEVICE CONTROL LOGIC
114
EXTERNAL DEVICE CONTROL LOGIC
114
EXTERNAL DEVICES
18
116

FIG. 2

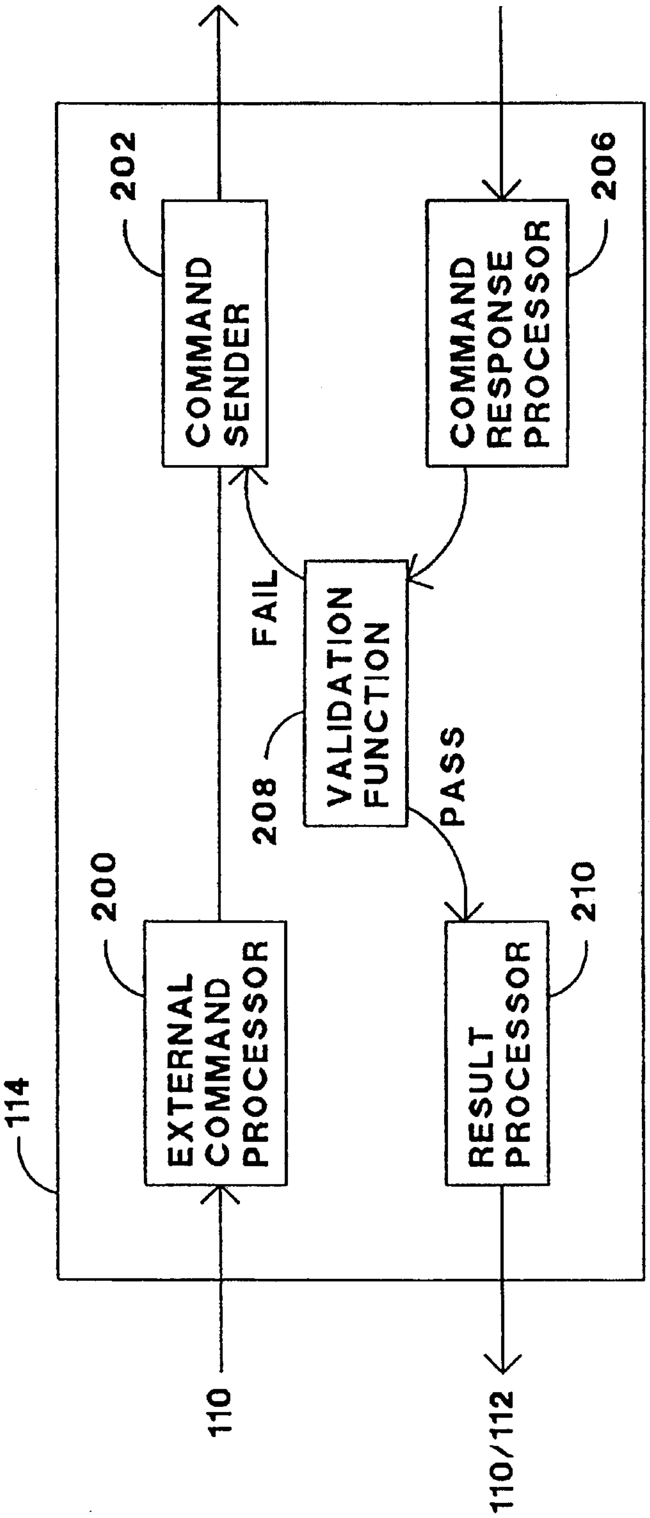

114
200
EXTERNAL COMMAND PROCESSOR
202
COMMAND SENDER
206
COMMAND RESPONSE PROCESSOR
208
VALIDATION FUNCTION
FAIL
PASS
210
RESULT PROCESSOR
110
110/112

FIG. 3

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

A digital command control (DDC) 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 suitable 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.

DigiToys 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 controller 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 acknowledgement 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 acknowledgement to the client program in a timely manner before the execution of the command by the digital command stations. Accordingly, the execution of commands provided by the resident external controlling interface to the digital command stations occur in a synchronous manner, such as a first-in-first-out manner. The COM and DCOM communications transport between the client program and the resident external controlling interface is operated in an asynchronous manner, namely providing an acknowledgement thereby releasing the communications transport to accept further communications 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 on an external system. The communications transport **12** may also use private or public communications protocol as a medium for communications. The client program **14** provides commands and the resident external controlling interface **16** responds to the communications transport **12** to exchange information. A description of COM (common object model) and DCOM (distributed common object model) is provided by Chappel in a book entitled Understanding ActiveX and OLE, Microsoft Press, and is incorporated by reference herein.

Incorporating a communications transport **12** between the client program(s) **14** and the resident external controlling interface **16** permits multiple operators of the model railroad at locations distant from the physical model railroad and each other. In the environment of a model railroad club where the members want to simultaneously control devices of the same model railroad layout, which preferably includes multiple trains operating thereon, 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 interface controller **16**. The synchronous manner of the request is the technique used by COM and DCOM to execute commands. The communications transport **12** packages the command for the transport mechanism to the resident external controlling interface **16**. The resident external controlling interface **16** then passes the command to the digital command stations **18** which in turn executes the command. After the digital command station **18** executes the command an acknowledgement is passed back to the resident external controlling interface **16** which in turn passes an acknowledgement to the client program **14**. Upon receipt of the acknowledgement 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 DigiToys 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 operated 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 considers the commands to occur nearly instantaneously 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 to the asynchronous command processor **100**, 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 operation using the asynchronous interface appears to the operator as nearly instantaneously responsive.

Each command in the command queue **104** is fetched by a synchronous command processor **110** and processed. The synchronous command processor **110** queries a controller database storage **112** for additional information, as necessary, and determines if the command has already been executed based on the state of the devices in the controller database storage **112**. In the event that the command has already been executed, as indicated by the controller database storage **112**, then the synchronous command processor **110** passes information to the command queue **104** that the command has been executed or the state of the device. The asynchronous response processor **106** fetches the information from the command 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 posted 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 **100** 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 instantrations of the asynchronous functionality, with one for each different client. The single command queue **104** also allows the sharing of multiple devices, multiple clients to communicate with the same device (locally or remote) in a controlled manner, and multiple clients to communicate with different devices. In other words, the command-queue **104** permits the proper execution in the cases of: (1) one client to many devices, (2) many clients to one device, and (3) many clients to many devices.

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

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

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

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

APPLICATION PROGRAMMING INTERFACE

Train ToolsTM Interface Description
Building your own visual interface to a model railroad
Copyright 1992–1998 KAM Industries.
Computer Dispatcher, Engine Commander, The Conductor, Train Server, and Train Tools are Trademarks of KAM Industries, all Rights Reserved.
Questions concerning the product can be EMAILED to:
traintools@kam.rain.com
You can also mail questions to:
KAM Industries
2373 NW 185th Avenue Suite 416
Hillsboro, Oregon 97124
FAX - (503) 291-1221

Table of contents

-continued

APPLICATION PROGRAMMING INTERFACE

3.1 Introduction
3.2 Data Types
3.3 Commands to access the server configuration variable database
- KamCVGetValue
- KamCVPutValue
- KamCVGetEnable
- KamCVPutEnable
- KamCVGetName
- KamCVGetMinRegister
- KamCVGetMaxRegister

3.4 Commands to program configuration variables
- KamProgram
- KamProgramGetMode
- KamProgramGetStatus
- KamProgramReadCV
- KamProgramCV
- KamProgramReadDecoderToDataBase
- KamProgramDecoderFromDataBase

3.5 Commands to control all decoder types
- KamDecoderGetMaxModels
- KamDecoderGetModelName
- KamDecoderSetModelToObj
- KamDecoderGetMaxAddress
- KamDecoderChangeOldNewAddr
- KamDecoderMovePort
- KamDecoderGetPort
- KamDecoderCheckAddrInUse
- KamDecoderGetModelFromObj
- KamDecoderGetModelFacility
- KamDecoderGetObjCount
- KamDecoderGetObjAtIndex
- KamDecoderPutAdd
- KamDecoderPutDel
- KamDecoderGetMfgName
- KamDecoderGetPowerMode
- KamDecoderGetMaxSpeed

3.6 Commands to control locomotive decoders
- KamEngGetSpeed
- KamEngPutSpeed
- KamEngGetSpeedSteps
- KamEngPutSpeedSteps
- KamEngGetFunction
- KamEngPutFunction
- KamEngGetFunctionMax
- KamEngGetName
- KamEngPutName
- KamEngGetFunctionName
- KamEngPutFunctionName
- KamEngGetConsistMax
- KamEngPutConsistParent
- KamEngPutConsistChild
- KamEngPutConsistRemoveObj

3.7 Commands to control accessory decoders
- KamAccGetFunction
- KamAccGetFunctionAll
- KamAccPutFunction
- KamAccPutFunctionAll
- KamAccGetFunctionMax
- KamAccGetName
- KamAccPutName
- KamAccGetFunctionName
- KamAccPutFunctionName
- KamAccRegFeedback
- KamAccRegFeedbackAll
- KamAccDelFeedback
- KamAccDelFeedbackAll

3.8 Commands to control the command station
- KamOprPutTurnOnStation
- KamOprPutStartStation
- KamOprPutClearStation
- KamOprPutStopStation
- KamOprPutPowerOn
- KamOprPutPowerOff
- KamOprPutHardReset
- KamOprPutEmergencyStop
- KamOprGetStationStatus

-continued

APPLICATION PROGRAMMING INTERFACE

3.9 Commands to configure the command station communication port
- KamPortPutConfig
- KamPortGetConfig
- KamPortGetName
- KamPortPutMapController
- KamPortGetMaxLogPorts
- KamPortGetMaxPhysical

3.10 Commands that control command flow to the command station
- KamCmdConnect
- KamCmdDisConnect
- KamCmdCommand

3.11 Cab Control Commands
- KamCabGetMessage
- KamCabPutMessage
- KamCabGetCabAddr
- KamCabPutAddrToCab

3.12 Miscellaneous Commands
- KamMiscGetErrorMsg
- KamMiscGetClockTime
- KamMiscPutClockTime
- KamMiscGetInterfaceVersion
- KamMiscSaveData
- KamMiscGetControllerName
- KamMiscGetControllerNameAtPort
- KamMiscGetCommandStationValue
- KamMiscSetCommandStationValue
- KamMiscGetCommandStationIndex
- KamMiscMaxControllerID
- KamMiscGetControllerFacility

I. 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.

A. Visual BASIC Throttle Example Source Code

```
' Copyright 1998, KAM Industries. All rights reserved.
'
'   This is a demonstration program showing the
'   integration of VisualBasic and Train Server(tm)
'   interface. You may use this application for non
'   commercial usage.
'
'$Date: $
'$Author: $
'$Revision: $
'$Log: $
'   Engine Commander, Computer Dispatcher, Train Server,
'   Train Tools, The Conductor and kamind are registered
'   Trademarks of KAM Industries. All rights reserved.
'
'   This first command adds the reference to the Train
'   ServerT Interface object Dim EngCmd As New EngComIfc
'
'   Engine Commander uses the term Ports, Devices and
'   Controllers
'   Ports ->  These are logical ids where Decoders are
'   assigned to. Train ServerT Interface supports a
'   limited number of logical ports. You can also think
'   of ports as mapping to a command station type. This
'   allows you to move decoders between command station
'   without losing any information about the decoder
'
'   Devices ->  These are communications channels
'   configured in your computer.
'   You may have a single device (com1) or multiple
'   devices
```

-continued

APPLICATION PROGRAMMING INTERFACE

```
'       (COM 1 - COM8, LPT1, Other). You are required to
'       map a port to a device to access a command station.
'       Devices start from ID 0 -> 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.
'       The Command
'       EngCmd.KamPortGetMaxPhysical(lMaxPhysical, lSerial,
'       lParallel) provides means that... lMaxPhysical =
'       lSerial + lParallel + lOther
'
'       Controller - These are command the command station
'       like LENZ, Digitrax
'       Northcoast, EasyDCC, Marklin... It is recommend
'       that you check the command station ID before you
'       use it.
'
'       Errors    - All commands return an error status. If
'                 the error value is non zero, then the
'                 other return arguments are invalid. In
'                 general, non zero errors means command was
'                 not executed. To get the error message,
'                 you need to call KamMiscErrorMessage and
'                 supply the error number
'
'       To Operate your layout you will need to perform a
'       mapping between a Port (logical reference), Device
'       (physical communications channel) and a Controller
'       (command station) for the program to work. All
'       references uses the logical device as the reference
'       device for access.
'
'       Addresses used are an object reference. To use an
'       address you must add the address to the command
'       station using KamDecoderPutAdd ... One of the return
'       values from this operation is an object reference
'       that is used for control.
'
'       We need certain variables as global objects; since
'       the information is being used multiple times
Dim iLogicalPort, iController, iComPort
Dim iPortRate, iPortParity, iPortStop, iPortRetrans,
        iPortWatchdog, iPortFlow, iPortData
Dim lEngineObject As Long, iDecoderClass As Integer,
        iDecoderType    As Integer
Dim lMaxController As Long
Dim lMaxLogical As Long, lMaxPhysical As Long, lMaxSerial
        As Long, lMaxParallel As Long
'*******************************
'Form load function
'- Turn of the initial buttons
'- Set he interface information
'*******************************
Private Sub Form_load()
        Dim strVer As String, strCom As String, strCntrl As
            String
        Dim iError As Integer
        'Get the interface version information
        SetButtonState (False)
        iError = EngCmd.KamMiscGetInterfaceVersion(strVer)
        If (iError) Then
                MsgBox (("Train Server not loaded. Check
                    DCOM-95"))
                iLogicalPort = 0
                LogPort.Caption = iLogicalPort
                ComPort.Caption = "???"
                Controller.Caption = "Unknown"
        Else
                MsgBox (("Simulation(COM1) Train Server -- " &
                    strVer))
                '*************************************
                'Configuration information; Only need to
                    change these values to use a different
                    controller...
                '*************************************
                ' UNKNOWN          0 // Unknown control type
                ' SIMULAT          1 // Interface simulator
```

-continued

APPLICATION PROGRAMMING INTERFACE

```
                ' LENZ_1x          2 // Lenz serial support module
                ' LENZ_2x          3 // Lenz serial support module
                ' DIGIT_DT200      4 // Digitrax direct drive
                                      support using DT200
                ' DIGIT_DCS100     5 // Digitrax direct drive
                                      support using DCS100
                ' MASTERSERIES     6 // North Coast engineering
                                      master Series
                ' SYSTEMONE        7 // System One
                ' RAMFIX           8 // RAMFIxx system
                ' DYNATROL         9 // Dynatrol system
                ' Northcoast binary 10 // North Coast binary
                ' SERIAL          11 // NMRA Serial
                                      interface
                ' EASYDCC         12 // NMRA Serial interface
                ' MRK6050         13 // 6050 Marklin interface
                                      (AC and DC)
                ' MRK6023         14 // 6023 Marklin hybrid
                                      interface (AC)
                ' ZTC             15 // ZTC Systems ltd
                ' DIGIT_PR1       16 // Digitrax direct drive
                                      support using PR1
                ' DIRECT          17 // Direct drive interface
                                      routine
'***********************************************************
        iLogicalPort = 1 'Select Logical port 1 for
                            communications
        iController = 1 'Select controller from the list
                            above.
        iComPort = 0 ' use COM1; 0 means com1 (Digitrax must
                            use Com1 or Com2)
                'Digitrax Baud rate requires 16.4K!
                'Most COM ports above Com2 do not
                'support 16.4K. Check with the
                'manufacture of your smart com card
                'for the baud rate. Keep in mind that
                'Dumb com cards with serial port
                'support Com1 - Com4 can only support
                '2 com ports (like com1/com2
                'or com3/com4)
                'If you change the controller, do not
                'forget to change the baud rate to
                'match the command station. See your
                'user manual for details
'***********************************************************
                ' 0: // Baud rate is 300
                ' 1: // Baud rate is 1200
                ' 2: // Baud rate is 2400
                ' 3: // Baud rate is 4800
                ' 4: // Baud rate is 9600
                ' 5: // Baud rate is 14.4
                ' 6: // Baud rate is 16.4
                ' 7: // Baud rate is 19.2
                iPortRate = 4
                '     Parity values 0-4 -> no, odd, even, mark,
                      space
                iPortParity = 0
                '     Stop bits 0,1,2 -> 1, 1.5, 2
                iPortStop = 0
                iPortRetrans = 10
                iPortWatchdog = 2048
                iPortFlow = 0
                '     Data bits 0 - > 7 Bits, 1-> 8 bits
                iPortData = 1
        'Display the port and controller information
        iError = EngCmd.KamPortGetMaxLogPorts(lMaxLogical)
        iError = EngCmd.KamPortGetMaxPhysical(lMaxPhysical,
                lMaxSerial, lMaxParallel)
        ' Get the port name and do some checking...
        iError = EngCmd.KamPortGetName(iComPort, strCom)
        SetError (iError)
        If (iComPort > lMaxSerial) Then MsgBox ("Com port
                our of range")
        iError =
                EngCmd.KamMiscGetControllerName(iController,
                strCntrl)
        If (iLogicalPort > lMaxLogical) Then MsgBox
```

-continued

APPLICATION PROGRAMMING INTERFACE

```
("Logical port out of range")
            SetError (iError)
    End If
      'Display values in Throttle..
      LogPort.Caption = iLogicalPort
      ComPort.Caption = strCom
      Controller.Caption = strCntrl
End Sub
'******************************
'Send Command
'Note:
'     Please follow the command order. Order is important
'     for the application to work!
'******************************
Private Sub Command_Click()
      'Send the command from the interface to the command
      station, use the engineObject
      Dim iError, iSpeed As Integer
      If Not Connect.Enabled Then
            TrainTools interface is a caching interface.
            'This means that you need to set up the CV's or
            'other operations first; then execute the
            'command.
            iSpeed = Speed.Text
            iError =
      EngCmd.KamEngPutFunction(lEngineObject, 0, F0.Value)
            iError =
            EngCmd.KamEngPutFunction(lEngineObject, 1,
            F1.Value)
            iError =
            EngCmd.KamEngPutFunction(lEngineObject, 2,
            F2.Value)
            iError =
            EngCmd.KamEngPutFunction(lEngineObject, 3,
            F3.Value)
            iError = EngCmd.KamEngPutSpeed(lEngineObject,
            iSpeed, Direction.Value)
            If iError = 0 Then iError =
            EngCmd.KamCmdCommand(lEngineObject)
            SetError (iError)
         End If
End Sub
'******************************
'Connect Controller
'******************************
Private Sub Connect_Click()
      Dim iError 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_WATCHDOG         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_WATCHDOG         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.KamPortPutConfig(iLogicalPort, 0,
            iPortRetrans, 0) ' setting PORT_RETRANS
      iError = EngCmd.KamPortPutConfig(iLogicalPort, 1
            iPortRate, 0) ' setting PORT_RATE
      iError = EnqCmd.KamPortPutConfig(iLogicalPort, 2,
            iPortParity, 0) ' setting PORT_PARITY
      iError = EngCmd.KamPortPutConfig(iLogicalPort, 3
            iPortStop, 0) ' setting PORT_STOP
```

-continued

APPLICATION PROGRAMMING INTERFACE

```
      iError = EngCmd.KamPortPutConfig(iLogicalPort, 4
            iPortWatchdog, 0) ' setting PORT_WATCHDOG
      iError = EngCmd.KamPortPutConfig(iLogicalPort, 5,
            iPortFlow, 0) ' setting PORT_FLOW
      iError = EngCmd.KamPortPutConfig(iLogicalPort, 6,
            iPortData, 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 share ware version of application (Shareware
'     always set to 130)
' Write Display   Log   Debug
' File   Win   Level   Value
' 1 + 2 + 4 = 7            -> LEVEL1 -- put packets into
'     queues
' 1 + 2 + 8 = 11           -> LEVEL2 -- Status messages
'     send to window
' 1 + 2 + 16 = 19          -> LEVEL3 --
' 1 + 2 + 32 = 35          -> LEVEL4 -- All system
'     semaphores/critical sections
' 1 + 2 + 64 = 67          -> LEVEL5 -- detailed
'     debugging information
' 1 + 2 + 128 =0 131       -> COMMONLY -- Read comm write
'     comm ports
'
'You probably only want to use values of 130. This will
'give you a display what is read or written to the
'controller. If you want to write the information to
'disk, use 131. The other information is not valid for
'end users.
' Note:   1.      This does effect the performance of you
'                 system; 130 is a save value for debug
'                 display. Always set the key to 1, a value
'                 of 0 will disable debug
'         2.      The Digitrax control codes displayed are
'                 encrypted. The information that you
'                 determine from the control codes is that
'                 information is sent (S) and a response is
'                 received (R)
'
iDebugMode = 130
iValue = Value.Text' Display value for reference
iError = EngCmd.KamPortPutConfig(iLogicalPort, 7, iDebug,
          iValue)' setting PORT_DEBUG
'Now map the Logical Port, Physical device, Command
      station and Controller
iError = EngCmd.KamPortPutMapController(iLogicalPort,
          iController, iComPort)
iError = EngCmd.KamCmdConnect(iLogicalPort)
iError = EngCmd.KamOprPutTurnOnStation(iLogicalPort)
If (iError) Then
      SetButtonState (False)
    Else
      SetButtonState (True)
    End If
SetError (iError) 'Displays the error message and error
      number
End Sub
'******************************
'Set the address button
'******************************
Private Sub DCCAddr_Click()
      Dim iAddr, iStatus As Integer
      ' All addresses must be match to a logical port to
      operate
      iDecoderType = 1          ' Set the decoder type to an NMRA
          baseline decoder ( 1 - 8 reg)
      iDecoderClass = 1         ' Set the decoder class to Engine
      decoder (there are only two classes of decoders;
      Engine and Accessory
      'Once we make a connection, we use the lEngineObject
      'as the reference object to send control information
      If (Address.Text > 1) Then
          iStatus = EngCmd.KamDecoderPutAdd(Address.Text,
              iLogicalPort, iLogicalPort, 0,
              iDecoderType, lEngineObject)
```

-continued

APPLICATION PROGRAMMING INTERFACE

```
        SetError (iStatus)
        If(lEngineObject) Then
              Command.Enabled = True 'turn on the control
              (send) button
              Throttle.Enabled = True 'Turn on the throttle
           Else
              MsgBox ("Address not set, check error message")
              End If
        Else
              MsgBox ("Address must be greater then 0 and
                    less then 128")
              End If
End Sub
'*******************
'Disconenct button
'*******************
Private Sub Disconnect_Click()
        Dim iError As Integer
        iError = EngCmd.KamCmdDisConnect(iLogicalPort)
        SetError (iError)
        SetButtonState (False)
End Sub
'**********************
'Display error message
'**********************
Private Sub SetError(iError As Integer)
        Dim szError As String
        Dim iStatus
        ' This shows how to retrieve a sample error message
        from the interface for the status received.
        iStatus = EngCmd.KamMiscGetErrorMsg(iError, szError)
        ErrorMsg.Caption = szError
        Result.Caption = Str(iStatus)
End Sub
'**************************
'Set the Form button state
'**************************
Private Sub SetButtonState(iState As Boolean)
        'We set the state of the buttons; either connected
        or disconnected
        If (iState) Then
              Connect.Enabled = False
              Disconnect.Enabled = True
              ONCmd.Enabled = True
              OffCmd.Enabled = True
              DCCAddr.Enabled = True
              UpDownAddress.Enabled = True
        'Now we check to see if the Engine Address has been
        'set; if it has we enable the send button
        If (lEngineObject > 0) Then
              Command.Enabled = True
              Throttle.Enabled = True
           Else
              Command.Enabled = False
              Throttle.Enabled = False
           End If
        Else
              Connect.Enabled = True
              Disconnect.Enabled = False
              Command.Enabled = False
              ONCmd.Enabled = False
              OffCmd.Enabled = False
              DCCAddr.Enabled = False
              UpDownAddress.Enabled = False
              Throttle.Enabled = False
              End If
End Sub
'*******************
'Power Off function
'*******************
Private Sub OffCmd_Click()
        Dim iError As Integer
        iError = EngCmd.KamOprPutPowerOff(iLogicalPort)
        SetError (iError)
End Sub
'*******************
'Power On function
```

-continued

APPLICATION PROGRAMMING INTERFACE

```
'*****************
Private Sub ONCmd_Click()
        Dim iError As Integer
        iError = EngCmd.KamOprPutPowerOn(iLogicalPort)
        SetError (iError)
End Sub
'***********************
'Throttle slider control
'***********************
Private Sub Throttle_Click()
        If (lEngineObject) Then
              If (Throttle.Value > 0) Then
                 Speed.Text = Throttle.Value
                 End If
              End If
End Sub
```

I. IDL COMMAND REFERENCE

A. Introduction

This document describes the IDL interface to the KAM Industries Engine Commander Train Server. The Train Server DCOM server may reside locally or on a network node This server handles all the background details of controlling your railroad. You write simple, front end programs in a variety of languages such as BASIC, Java, or C++ to provide the visual interface to the user while the server handles the details of communicating with the command station, etc.

A. Data Types

Data is passed to and from the IDL interface using a several primitive data types. Arrays of these simple types are also used. The exact type passed to and from your program depends on the programming language your are using.

The following primitive data types are used:

IDL Type	BASIC Type	C++ Type	Java Type	Description
short	short	short	short	Short signed integer
int	int	int	int	Signed integer
BSTR	BSTR	BSTR	BSTR	Text string
long	long	long	long	Unsigned 32 bit value

Name ID CV Range Valid CV's Functions Address Range Speed Steps
NMRA Compatible 0 None None 2 1–99 14
Baseline 1 1–8 1–8 9 1–127 14
Extended 2 1–106 1–9, 17, 18, 19, 23, 24, 29, 30, 49, 66–95 9 1–10239 14,28,128
All Mobile 3 1–106 1–106 9 1–10239 14,28,128

Name ID	CV Range	Valid CV's	Functions	Address Range
Accessory	4	513–593	513–593	8	0–511
All Stationary	5	513–1024	513–1024	8	0–511

A long /DecoderObject/D value is returned by the KamDecoderPutAdd call if the decoder is successfully registered with the server. This unique opaque ID should be used for all subsequent calls to reference this decoder.

A. Commands to access the server configuration variable database

This section describes the commands that access the server configuration variables (CV) database. These CVs are stored in the decoder and control many of its characteristics such as its address. For efficiency, a copy of each CV value is also stored in the server database. Commands such as KamCVGetValue and KamCVPutValue communicate only with the server, not the actual decoder. You then use the programming commands in the next section to transfer CVs to and from the decoder.

0KamCVGetValue

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iCVRegint		1–1024 2	In	CV register
pCVValue	int *	3	Out	Pointer to CV value

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Range is 1–1024. Maximum CV for this decoder is given by KamCVGetMaxRegister.

3 CV Value pointed to has a range of 0 to 255.

Return Value	Type	Range	Description
iError short	1	Error flag	

-continued

APPLICATION PROGRAMMING INTERFACE

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamCVGetValue takes the decoder object ID and configuration variable (CV) number as parameters. It sets the memory pointed to by pCVValue to the value of the server copy of the configuration variable.

0KamCVPutValue

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iCVRegint	1–1024	2	In	CV register
iCVValue	int	0–255	In	CV value

1 Opaque object ID handle returned by KamDecoderPutAdd.

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

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamCVPutValue takes the decoder object ID, configuration variable (CV) number, and a new CV value as parameters. It sets the server copy of the specified decoder CV to iCVValue.

0KamCVGetEnable

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iCVRegint	1–1024	2	In	CV number
pEnable	int *	3	Out	Pointer to CV bit mask

1 Opaque object ID handle returned by KamDecoderPutAdd.

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

3 0x0001 - SET_CV_INUSE 0x0002 - SET_CV_READ_DIRTY
0x0004 - SET_CV_WRITE_DIRTY 0x0008 - SET_CV_ERROR_READ
0x0010 - SET_CV_ERROR_WRITE

Return Value	Type	Range	Description
iError short	1	Error flag	

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamCVGetEnable takes the decoder object ID, configuration variable (CV) number, and a pointer to store the enable flag as parameters. It sets the location pointed to by pEnable.

0KamCVPutEnable

Parameter List	Type	Range	Direction	Description
iDecoderObjectID	long	1	In	Decoder object ID
iCVRegint	1–1024	2	In	CV number
iEnableint		3	In	CV bit mask

1 Opaque object ID handle returned by KamDecoderPutAdd.

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

3 0x0001 - SET_CV_INUSE 0x0002 - SET_CV_READ_DIRTY
0x0004 - SET_CV_WRITE_DIRTY 0x0008 - SET_CV_ERROR_READ
0x0010 - SET_CV_ERROR_WRITE

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamCVPutEnable takes the decoder object ID, configuration variable (CV) number, and a new enable state as parameters. It sets the server copy of the CV bit mask to iEnable.

0KamCVGetName

Parameter List	Type	Range	Direction	Description
iCV	int	1–1024	In	CV number
pbsCVNameString	BSTR *	1	Out	Pointer to CV name string

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

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamCVGetName takes a configuration variable (CV) number as a parameter. It sets the memory pointed to by pbsCVNameString to the name of the CV as defined in NMRA Recommended Practice RP 9.2.2.

0KamCVGetMinRegister

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
pMinRegister	int *	2	Out	Pointer to min CV register number

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Normally 1–1024. 0 on error or if decoder does not support CVs.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamCVGetMinRegister takes a decoder object ID as a parameter. It sets the memory pointed to by pMinRegister to the minimum possible CV register number for the specified decoder.

0KamCVGetMaxRegister

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
pMaxRegister	int *	2	Out	Pointer to max CV register number

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Normally 1–1024. 0 on error or if decoder does not support CVs.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamCVGetMaxRegister takes a decoder object ID as a parameter. It sets the memory pointed to by pMaxRegister to the maximum possible CV register number for the specified decoder.

A. Commands to program configuration variables

This section describes the commands read and write decoder configuration variables (CVs). You should initially transfer a copy of the decoder CVs to the server using the KamProgramReadDecoderToDataBase command. You can then read and modify this server copy of the CVs. Finally, you can program one or more CVs into the decoder using the KamProgramCV or KamProgramDecoderFromDataBase command. Not that you must first enter programming mode by issuing the KamProgram command before any programming can be done.

0KamProgram

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iProgLogPort	int	1–65535 2	In	Logical programming port ID
iProgMode	int	3	In	Programming mode

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Maximum value for this server given by KamPortGetMaxLogPorts.

3 0 - PROGRAM_MODE_NONE
1 - PROGRAM_MODE_ADDRESS
2 - PROGRAM_MODE_REGISTER
3 - PROGRAM_MODE_PAGE
4 - PROGRAM_MODE_DIRECT
5 - DCODE_PRGMODE_OPS_SHORT
6 - PROGRAM_MODE_OPS_LONG

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamProgram take the decoder object ID, logical programming port ID, and programming mode as parameters. It changes the command station mode from normal operation (PROGRAM_MODE_NONE) to the specified programming mode. Once in programming modes, any number of programming commands may be called. When done, you must call KamProgram with a parameter of PROGRAM_MODE_NONE to return to normal operation.

-continued

APPLICATION PROGRAMMING INTERFACE

0KamProgramGetMode

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iProgLogPort	int	1–65535 2	In	Logical programming port ID
piProgMode	int *	3	Out	Programming mode

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Maximum value for this server given by KamPortGetMaxLogPorts.

3 0 - PROGRAM_MODE_NONE
1 - PROGRAM_MODE_ADDRESS
2 - PROGRAM_MODE_REGISTER
3 - PROGRAM_MODE_PAGE
4 - PROGRAM_MODE_DIRECT
5 - DCODE_PRGMODE_OPS_SHORT
6 - PROGRAM_MODE_OPS_LONG

Return Value	Type	Range	Description
iError	short	1	Error flag Description

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

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

0KamProgramGetStatus

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iCVRegint		0–1024 2	In	CV number
piCVAllStatus	int *	3	Out	Or'd decoder programming status

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 0 returns OR'd value for all CVs. Other values return status tor just that CV.

3 0x0001 - SET_CV_INUSE
0x0002 - SET_CV_READ_DIRTY
0x0004 - SET_CV_WRITE_DIRTY
0x0008 - SET_CV_ERROR_READ
0x0010 - SET_CV_ERROR_WRITE

Return Value	Type	Range	Description
iError	short	1	Error flag

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

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

0KamProgramReadCV

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iCVRegint		2	In	CV number

1 Opaque object ID handle returned by KamDecoderPutAdd.

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

Return Value	Type	Range	Description
iError	short	1	Error flag

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

KamProgramCV takes the decoder object ID, configuration variable (CV) number as parameters. It reads the specified CV variable value to the server database.

0KamProgramCV

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iCVRegint		2	In	CV number
iCVValue	int	0–255	In	CV value

1 Opaque object ID handle returned by KamDecoderPutAdd.

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

Return Value	Type	Range	Description
iError	short	1	Error flag

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

-continued

APPLICATION PROGRAMMING INTERFACE

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

0KamProgramReadDecoderToDataBase

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

1 Opaque object ID handle returned by KamDecoderPutAdd.

Return Value	Type	Range	Description
iError	short	1	Error flag

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

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

0KamProgramDecoderFromDataBase

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

1 Opaque object ID handle returned by KamDecoderPutAdd.

Return Value	Type	Range	Description
iError	short	1	Error flag

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

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

A. Commands to control all decoder types

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

0KamDecoderGetMaxModels

Parameter List	Type	Range	Direction	Description
piMaxModels	int *	1	Out	Pointer to Max model ID

1 Normally 1–65535. 0 on error.

Return Value	Type	Range	Description
iError	short	1	Error flag

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

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

0KamDecoderGetModelName

Parameter List	Type	Range	Direction	Description
iModel	int	1–65535 1	In	Decoder type ID
pbsModelName	BSTR *	2	Out	Decoder name string

1 Maximum value for this server given by KamDecoderGetMaxModels.

2 Exact return type depends on language. It is Cstring * for C++. Empty string on error.

Return Value	Type	Range	Description
iError	short	1	Error flag

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg). KamPortGetModelName takes a decoder type ID and a pointer to a string as parameters. It sets the memory pointed to by pbsModelName to a BSTR containing the decoder name.

0KamDecoderSetModelToObj

Parameter List	Type	Range	Direction	Description
iModel	int	1	In	Decoder model ID
lDecoderObjectID	long	1	In	Decoder object ID

1 Maximum value for this server given by KamDecoderGetMaxModels.

2 Opaque object ID handle returned by KamDecoderPutAdd.

Return Value	Type	Range	Description
iError	short	1	Error flag

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

KamDecoderSetModelToObj takes a decoder ID and decoder object ID as parameters. It sets the decoder model type of the decoder at address lDecoderObjectID to the type

-continued

APPLICATION PROGRAMMING INTERFACE

specified by iModel.

0KamDecoderGetMaxAddress

Parameter List	Type	Range	Direction	Description
iModel	int	1	In	Decoder type ID
piMaxAddress	int *	2	Out	Maximum decoder address

1 Maximum value for this server given by KamDecoderGetMaxModels

2 Model dependent. 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 KamMiscGetErrorMsg).

KamDecoderGetMaxAddress takes a decoder type ID and a pointer to store the maximum address as parameters. It sets the memory pointed to by piMaxAddress to the maximum address supported by the specified decoder.

0KamDecoderChangeOldNewAddr

Parameter List	Type	Range	Direction	Description
lOldObjID	long	1	In	Old decoder object ID
iNewAddr	int	2	In	New decoder address
plNewObjID	long *	1	Out	New decoder object ID

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 1–127 for short locomotive addresses. 1–10239 for long locomotive decoders. 0–511 for accessory decoders.

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamDecoderChangeOldNewAddr takes an old decoder object ID and a new decoder address as parameters. It moves the specified locomotive or accessory decoder to iNewAddr and sets the memory pointed to by plNewObjID to the new object ID. The old object ID is now invalid and should no longer be used.

0KamDecoderMovePort

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iLogicalPortID	int	1–65535 2	In	Logical port ID

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamDecoderMovePort takes a decoder object ID and logical port ID as parameters. It moves the decoder specified by lDecoderObjectID to the controller specified by iLogicalPortID.

0KamDecoderGetPort

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
piLogicalPortID	int *	1–65535 2	Out	Pointer to logical port ID

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamDecoderMovePort takes a decoder object ID and pointer to a logical port ID as parameters. It sets the memory pointed to by piLogicalPortID to the logical port ID associated with lDecoderObjectID.

0KamDecoderCheckAddrInUse

Parameter List	Type	Range	Direction	Description
iDecoderAddress	int	1	In	Decoder address
iLogicalPortID	int	2	In	Logical Port ID
iDecoderClass	int	3	In	Class of decoder

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Maximum value for this server given by KamPortGetMaxLogPorts.

3 1 - DECODER_ENGINE_TYPE,
2 - DECODER_SWITCH_TYPE,
3 - DECODER_SENSOR_TYPE.

Return Value	Type	Range	Description
iError short	1		Error flag

1 iError = 0 for successful call and address not in use. Nonzero is an error number (see KamMiscGetErrorMsg). IDS_ERR_ADDRESSEXIST returned if call succeeded but the address exists.

KamDecoderCheckAddrInUse takes a decoder address, logical port, and decoder class as parameters. It returns zero if the address is not in use. It will return IDS_ERR_ADDRESSEXIST if the call succeeds but the address already exists. It will return the appropriate non zero error number if the calls fails.

0KamDecoderGetModelFromObj

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
piModelint *		1–65535 2	Out	Pointer to decoder type ID

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Maximum value for this server given by KamDecoderGetMaxModels.

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamDecoderGetModelFromObj takes a decoder object ID and pointer to a decoder type ID as parameters. It sets the memory pointed to by piModel to the decoder type ID associated with iDCCAddr.

0KamDecoderGetModelFacility

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
pdwFacility	long *	2	Out	Pointer to decoder facility mask

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 0 - DCODE_PRGMODE_ADDR
1 - DCODE_PRGMODE_REG
2 - DCODE_PRGMODE_PAGE
3 - DCODE_PRGMODE_DIR
4 - DCODE_PRGMODE_FLYSHT
5 - DCODE_PRGMODE_FLYLNG
6 - Reserved
7 - Reserved
8 - Reserved
9 - Reserved
10 - Reserved
11 - Reserved
12 - Reserved
13 - DCODE_FEAT_DIRLIGHT
14 - DCODE_FEAT_LNGADDR
15 - DCODE_FEAT_CVENABLE
16 - DCODE_FEDMODE_ADDR
17 - DCODE_FEDMODE_REG
18 - DCODE_FEDMODE_PAGE
19 - DCODE_FEDMODE_DIR
20 - DCODE_FEDMODE_FLYSHT
21 - DCODE_FEDMODE_FLYLNG

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamDecoderGetModelFacility takes a decoder object ID and pointer to a decoder facility mask as parameters. It sets the memory pointed to by pdwFacility to the decoder facility mask associated with iDCCAddr.

0KamDecoderGetObjCount

Parameter List	Type	Range	Direction	Description
iDecoderClass	int	1	In	Class of decoder
piObjCount	int *	0–65535	Out	Count of active decoders

1 1 - DECODER_ENGINE_TYPE,
2 - DECODER_SWITCH_TYPE,

-continued

APPLICATION PROGRAMMING INTERFACE

3 - DECODER_SENSOR_TYPE.

Return Value	Type	Range	Description*
iError short	1	Error flag	

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

KamDecoderGetObjCount takes a decoder class and a pointer to an address count as parameters. It sets the memory pointed to by piObjCount to the count of active decoders of the type given by iDecoderClass.

0KamDecoderGetObjAtIndex

Parameter List	Type	Range	Direction	Description*
iIndex	int	1	In	Decoder array index
iDecoderClass	int	2	In	Class of decoder
plDecoderObjectID	long *	3	Out	Pointer to decoder object ID

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	

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

KamDecoderGetObjCount takes a decoder index, decoder class, and a pointer to an object ID as parameters. It sets the memory pointed to by plDecoderObjectID to the selected object ID.

0KamDecoderPutAdd

Parameter List	Type	Range	Direction	Description
iDecoderAddress	int	1	In	Decoder address
iLogicalCmdPortID	int	1–65535 2	In	Logical command port ID
iLogicalProgPortID	int	1–65535 2	In	Logical programming port ID
iClearState	int	3	In	Clear state flag
iModel	int	4	In	Decoder model type ID
plDecoderObjectID	long *	5	Out	Decoder object ID

1 1–127 for short locomotive addresses. 1–10239 for long locomotive decoders. 0–511 for accessory decoders.

2 Maximum value for this server given by KamPortGetMaxLogPorts.

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	

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

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 plDecoderObjectID to the decoder object ID used by the server as a key.

0KamDecoderPutDel

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iClearState	int	2	In	Clear state flag

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	

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

KamDecoderPutDel takes a decoder object ID and clear flag as parameters. It deletes the locomotive object specified by lDecoderObjectID from the locomotive database.

0KamDecoderGetMfgName

-continued

APPLICATION PROGRAMMING INTERFACE

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
pbsMfgName	BSTR *	2	Out	Pointer to manufacturer name

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Exact return type depends on language. It is Cstring * for C++. Empty string on error.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamDecoderGetMfgName takes a decoder object ID and pointer to a manufacturer name string as parameters. It sets the memory pointed to by pbsMfgName to the name of the decoder manufacturer.

0KamDecoderGetPowerMode

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object TD
pbsPowerMode	BSTR *	2	Out	Pointer to decoder power mode

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Exact return type depends on language. It is Cstring * for C++. Empty string on error.

Return Value	Type	Range	Description*
iError short	1	Error flag	

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

KamDecoderGetPowerMode takes a decoder object ID and a pointer to the power mode string as parameters. It sets the memory pointed to by pbsPowerMode to the decoder power mode.

0KamDecoderGetMaxSpeed

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
piSpeedStep	int *	2	Out	Pointer to max speed step

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 14, 28, 56, or 128 for locomotive decoders. 0 for accessory decoders.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamDecoderGetMaxSpeed takes a decoder object ID and a pointer to the maximum supported speed step as parameters. It sets the memory pointed to by piSpeedStep to the maximum speed step supported by the decoder.

A. Commands to control locomotive decoders

This section describes the commands that control locomotive decoders. These commands control things such as locomotive speed and direction. For efficiency, a copy of all the engine variables such speed is stored in the server. Commands such as KamEngGetSpeed 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 KamCmdCommand command.

0KamEngGetSpeed

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
lpSpeed	int *	2	Out	Pointer to locomotive speed
lpDirection	int *	3	Out	Pointer to locomotive direction

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Speed range is dependent on whether the decoder is set to 14, 18, or 128 speed steps and matches the values defined by NMRA S9.2 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

-continued

APPLICATION PROGRAMMING INTERFACE

Return Value	Type	Range	Description
iError	short	1	Error flag

1 iError = 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 lpSpeed to the locomotive speed and the memory pointed to by lpDirection to the locomotive direction.

0KamEngPutSpeed

Parameter List	Type	Range	Direction	Description*
lDecoderObjectID	long	1	In	Decoder object ID
iSpeed	int *	2	In	Locomotive speed
iDirection	int *	3	In	Locomotive direction

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Speed range is dependent on whether the decoder is set to 14, 18, or 128 speed steps and matches the values defined by NMRA S9.2 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

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

KamEngPutSpeed 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 decoders range.

0KamEngGetSpeedSteps

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
lpSpeedSteps	int *	14,28,128	Out	Pointer to number of speed steps

1 Opaque object ID handle returned by KamDecoderPutAdd.

Return Value	Type	Range	Description
iError	short	1	Error flag

1 iError = 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 lpSpeedSteps to the number of speed steps.

0KamEngPutSpeedSteps

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iSpeedSteps	int	14,28,128	In	Locomotive speed steps

1 Opaque object ID handle returned by KamDecoderPutAdd.

Return Value	Type	Range	Description
iError	short	1	Error flag

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

KamEngPutSpeedSteps 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. KamDecoderGetMaxSpeed returns the maximum possible speed for the decoder. An error is generated if an attempt is made to set the speed steps beyond this value.

0KamEngGetFunction

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iFunctionID	int	0–8 2	In	Function ID number
lpFunction	int *	3	Out	Pointer to function value

1 Opaque object ID handle returned by KamDecoderPutAdd.

-continued

APPLICATION PROGRAMMING INTERFACE

2 FL is 0. F1–F8 are 1–8 respectively. Maximum for this decoder is given by KamEngGetFunctionMax. 3 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 (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 lpFunction to the specified function state.

0KamEngPutFunction

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iFunctionID	int	0–8 2	In	Function ID number
iFunction	int	3	In	Function value

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 FL is 0. F1–F8 are 1–8 respectively. Maximum for this decoder is given by KamEngGetFunctionMax.

3 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 (see KamMiscGetErrorMsg).

KamEngPutFunction 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.

0KamEngGetFunctionMax

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
piMaxFunction	int *	0–8	Out	Pointer to maximum function number

1 Opaque object ID handle returned by KamDecoderPutAdd.

Return Value	Type	Range	Description
iError	short	1	Error flag

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

KamEngGetFunctionMax takes a decoder object ID and a pointer to the maximum function ID as parameters. It sets the memory pointed to by piMaxFunction to the maximum possible function number for the specified decoder.

0KamEngGetName

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
pbsEngName	BSTR *	2	Out	Pointer to locomotive name

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Exact return type depends on language. It is Cstring * for C++. Empty string on error.

Return Value	Type	Range	Description
iError	short	1	Error flag

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

KamEngGetName takes a decoder object ID and a pointer to the locomotive name as parameters. It sets the memory pointed to by pbsEngName to the name of the locomotive.

0KamEngPutName

Parameter List	Type	Range	Direction	Description*
lDecoderObjectID	long	1	In	Decoder object ID
bsEngName	BSTR	2	Out	Locomotive name

1 Opaque object ID handle returned by KamDecoderPutAdd.

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

Return Value	Type	Range	Description
iError	short	1	Error flag

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

-continued

APPLICATION PROGRAMMING INTERFACE

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

0KamEngGetFunctionName

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iFunctionID	int	0–8 2	In	Function ID number
pbsFcnNameString	BSTR *	3	Out	Pointer to function name

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 FL is 0. F1–F8 are 1–8 respectively. Maximum for this decoder is given by KamEngGetFunctionMax. 3 Exact return type depends on language. It is Cstring * for C++. Empty string on error.

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamEngGetFuncntionName takes a decoder object ID, function ID, and a pointer to the function name as parameters. It sets the memory pointed to by pbsFcnNameString to the symbolic name of the specified function.

0KamEngPutFunctionName

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iFunctionID	int	0–8 2	In	Function ID number
bsFcnNameString	BSTR	3	In	Function name

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 FL is 0. F1–F8 are 1–8 respectively. Maximum for this decoder is given by KamEngGetFunctionMax.

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

Return Value	Type	Range	Description
iError short	1		Error Flag

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

KamEngPutFunctionName takes a decoder object ID, function ID, and a BSTR as parameters. It sets the specified symbolic function name to bsFcnNameString.

0KamEngGetConsistMax

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

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Command station dependent.

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamEngGetConsistMax takes the decoder object ID and a pointer to a location to store the maximum consist as parameters. It sets the location pointed to by piMaxConsist to the maximum number of locomotives that can but 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.

0KamEngPutConsistParent

Parameter List	Type	Range	Direction	Description
lDCCParentObjID	long	1	In	Parent decoder object ID
iDCCAliasAddr	int	2	In	Alias decoder address

1 Opaque object ID handle returned by KamDecoderPutAdd.

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

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamEngPutConsistParent takes the parent object ID and an alias address as parameters. It makes the decoder specified by lDCCParentObjID 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 KamEngPutConsistRemoveObj.

0KamEngPutConsistChild

Parameter List	Type	Range	Direction	Description
lDCCParentObjID	long	1	In	Parent decoder object ID
lDCCObjID	long	1	In	Decoder object ID

1 Opaque object ID handle returned by KamDecoderPutAdd.

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamEngPutConsistChild takes the decoder parent object ID and decoder object ID as parameters. It assigns the decoder specified by lDCCObjID to the consist identified by lDCCParentObjID. 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 KamEngPutConsistParent.

0KamEngPutConsistRemoveObj

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

1 Opaque object ID handle returned by KamDecoderPutAdd.

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamEngPutConsistRemoveObj takes the decoder object ID as a parameter. It removes the decoder specified by lDecoderObjectID 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 KamAccGetFunction 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 KamCmdCommand command.

0KamAccGetFunction

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iFunctionID	int	0–31 2	In	Function ID number
lpFunction	int *	3	Out	Pointer to function value

1 Opaque object ID handle returned by KamDecoderPutAdd.

2 Maximum for this decoder is given by KamAccGetFunctionMax.

3 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 (see KamMiscGetErrorMsg).

KamAccGetFunction 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 lpFunction to the specified function state.

0KamAccGetFunctionAll

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
piValue	int *	2	Out	Function bit mask

-continued

APPLICATION PROGRAMMING INTERFACE

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Each bit represents a single function state. Maximum for this decoder is given by KamAccGetFunctionMax.

Return Value	Type	Range	Description
iError	short	i	Error flag

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

KamAccGetFunctionAll takes the decoder object ID and a pointer to a bit mask as parameters. It sets each bit in the memory pointed to by piValue to the corresponding function state.

0KamAccPutFunction

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iFunctionID	int	0–31 2	In	Function ID number
iFunction	int	3	In	Function value

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Maximum for this decoder is given by KamAccGetFunctionMax.
3 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 (see KamMiscGetErrorMsg).

KamAccPutFunction takes the decoder object ID, a function ID, and a new function state as parameters. It sets the specified accessory database function state to lFunction. Note: This command only changes the accessory database. The data is not sent to the decoder until execution of the KamCmdCommand command.

0KamAccPutFunctionAll

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iValue	int	2	In	Pointer to function state array

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 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 (see KamMiscGetErrorMsg).

KamAccPutFunctionAll takes the decoder object ID and a bit mask as parameters. It sets all decoder function enable states to match the state bits in iValue. The possible enable states are TRUE and FALSE. The data is not sent to the decoder until execution of the KamCmdCommand command.

0KamAccGetFunctionMax

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
piMaxFunction	int *	0–31 2	Out	Pointer to maximum function number

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 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 (see KamMiscGetErrorMsg).

KamAccGetFunctionMax takes a decoder object ID and pointer to the maximum function number as parameters. It sets the memory pointed to by piMaxFunction to the maximum possible function number for the specified decoder.

0KamAccGetName

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
pbsAccNameString	BSTP *	2	Out	Accessory name

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

Return Value	Type	Range	Description
iError	short	1	Error flag

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

KamAccGetName takes a decoder object ID and a pointer to a string as parameters. It sets the memory pointed to by pbsAccNameString to the name of the accessory.

0KamAccPutName

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
bsAccNameString	BSTR	2	In	Accessory name

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Exact parameter type depends on language. It is LPCSTR for C++.

Return Value	Type	Range	Description
iError	short	1	Error flag

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

KamAccPutName takes a decoder object ID and a BSTR as parameters. It sets the symbolic accessory name to bsAccName.

0KamAccGetFunctionName

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iFunctionID	int	0–31 2	In	Function ID number
pbsFcnNameString	BSTR *	3	Out	Pointer to function name

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Maximum for this decoder is given by KamAccGetFunctionMax.
3 Exact return type depends on language. It is Cstring * for C++. Empty string on error.

Return Value	Type	Range	Description•
iError	short	1	Error flag

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

KamAccGetFuncntionName takes a decoder object ID, function ID, and a pointer to a string as parameters. It sets the memory pointed to by pbsFcnNameString to the symbolic name of the specified function.

0KamAccPutFunctionName

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iFunctionID	int	0–31 2	In	Function ID number
bsFcnNameString	BSTR	3	In	Function

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Maximum for this decoder is given by KamAccGetFunctionMax.
3 Exact parameter type depends on language. It is LPCSTR for C++.

Return Value	Type	Range	Description
iError	short	1	Error flag

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

KamAccPutFunctionName takes a decoder object ID, function ID, and a BSTR as parameters. It sets the specified symbolic function name to bsFcnNameString.

0KamAccRegFeedback

Parameter List	Type	Range	Direction	Description•
lDecoderObjectID	long	1	In	Decoder object ID
bsAccNode	BSTR	1	In	Server node name
iFunctionID	int	0–31 3	In	Function ID number

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Exact parameter type depends on language. It is LPCSTR for C++.
3 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

-continued

APPLICATION PROGRAMMING INTERFACE

(see KamMiscGetErrorMsg).
KamAccRegFeedback takes a decoder object ID, node name string, and function ID, as parameters. It registers interest in the function given by iFunctionID by the method given by the node name string bsAccNode. bsAccNode 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.

0KamAccRegFeedbackAll

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
bsAccNode	BSTR	2	In	Server node name

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Exact parameter type depends on language. It is LPCSTR for C++.

Return Value	Type	Range	Description
iError short	1		Error flag

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg).
KamAccRegFeedbackAll takes a decoder object ID and node name string as parameters. It registers interest in all functions by the method given by the node name string bsAccNode. bsAccNode 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.

0KamAccDelFeedback

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
bsAccNode	BSTR	2	In	Server node name
iFunctionID	int	0–31 3	In	Function ID number

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Exact parameter type depends on language. It is LPCSTR for C++.
3 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 (see KamMiscGetErrorMsg).
KamAccDelFeedback takes a decoder object ID, node name string, and function ID, as parameters. It deletes interest in the function given by iFunctionID by the method given by the node name string bsAccNode. bsAccNode 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.

0KamAccDelFeedbackAll

Parameter List	Type	Range	Direction	Description•
lDecoderObjectID	long	1	In	Decoder object ID
bsAccNode	BSTR	2	In	Server node name

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Exact parameter type depends on language. It is LPCSTR for C++.

Return Value	Type	Range	Description
iError short	1		Error flag

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg).
KamAccDelFeedbackAll takes a decoder object ID and node name string as parameters. It deletes interest in all functions by the method given by the node name string bsAccNode. bsAccNode 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.

A. Commands to control the command station

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

0KamOprPutTurnOnStation

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg).
KamOprPutTurnOnStation takes a logical port ID as a parameter. It performs the steps necessary to turn on the command station. This command performs a combination of other commands such as KamOprPutStartStation, KamOprPutClearStation, and KamOprPutPowerOn.

0KamOprPutStartStation

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg).
KamOprPutStartStation takes a logical port ID as a parameter. It performs the steps necessary to start the command station.

0KamOprPutClearStation

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg).
KamOprPutClearStation takes a logical port ID as a parameter. It performs the steps necessary to clear the command station queue.

0KamOprPutStopStation

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg).
KamOprPutStopStation takes a logical port ID as a parameter. It performs the steps necessary to stop the command station.

0KamOprPutPowerOn

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg).
KamOprPutPowerOn takes a logical port ID as a parameter. It performs the steps necessary to apply power to the track.

0KamOprPutPowerOff

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg).
KamOprPutPowerOff takes a logical port ID as a parameter. It performs the steps necessary to remove power from the track.

-continued

APPLICATION PROGRAMMING INTERFACE

0KamOprPutHardReset

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value tor this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamOprPutHardReset takes a logical port ID as a parameter. It performs the steps necessary to perform a hard reset of the command station.

0KamOprPutEmergencyStop

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamOprPutEmergencyStop takes a logical port ID as a parameter. It performs the steps necessary to broadcast an emergency stop command to all decoders.

0KamOprGetStationStatus

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID
pbsCmdStat	BSTR *	2	Out	Command station status string

1 Maximum value for this server given by KamPortGetMaxLogPorts.

2 Exact return type depends on language. It is Cstring * for C++.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamOprGetStationStatus takes a logical port ID and a pointer to a string as parameters. It set the memory pointed to by pbsCmdStat to the command station status. The exact format of the status BSTR is vendor dependent.

A. Commands to configure the command station communication port

This section describes the commands that configure the command station communication port. These commands do things such as setting BAUD rate. Several of the commands in this section use the numeric controller ID (iControllerID) to identify a specific type of command station controller. The following table shows the mapping between the controller ID (iControllerID) and controller name (bsControllerName) for a given type of command station controller.

iControllerID	bsControllerName	Description
0	UNKNOWN	Unknown controller type
1	SIMULAT	Interface simulator
2	LENZ_1x	Lenz version 1 serial support module
3	LENZ_2x	Lenz version 2 serial support module
4	DIGIT_DT200	Digitrax direct drive support using DT200
5	DIGIT_DCS100	Digitrax direct drive support using DCS100
6	MASTERSERIES	North coast engineering master series
7	SYSTEMONE	System one
8	RAMFIX	RAMFIxx system
9	SERIAL	NMRA serial interface
10	EASYDCC	CVP Easy DCC
11	MPK6050	Marklin 6050 interface (AC and DC)
12	MPK6023	Marklin 6023 interface (AC)
13	DIGIT_PR1	Digitrax direct drive using PR1
14	DIRECT	Direct drive interface routine
15	ZTC	ZTC system ltd
16	TRIX	TRIX controller

iIndex	Name	iValue Values
0	RETRANS	10–255
1	RATE	0 - 300 BAUD, 1 - 1200 BAUD, 2 - 2400 BAUD, 3 - 4800 BAUD, 4 - 9600 BAUD, 5 - 14400 BAUD, 6 - 16400 BAUD, 7 - 19200 BAUD
2	PARITY	0 - NONE, 1 - ODD, 2 - EVEN, 3 - MARK, 4 - SPACE
3	STOP	0 - 1 bit, 1 - 1.5 bits, 2 - 2 bits
4	WATCHDOG	500 - 65535 milliseconds. Recommended value 2048
5	FLOW	0 - NONE, 1 - XON/XOFF, 2 - RTS/CTS, 3 BOTH
6	DATA	0 - 7 bits, 1 - 8 bits
7	DEBUG	Bit mask. Bit 1 sends messages to debug file. Bit 2 sends messages to the screen. Bit 3 shows queue data. Bit 4 shows UI status. Bit 5 is reserved. Bit 6 shows semaphore and critical sections. Bit 7 shows miscellaneous messages. Bit 8 shows comm port activity. 130 decimal is recommended for debugging.
8	PARALLEL	

-continued

APPLICATION PROGRAMMING INTERFACE

0KamPortPutConfig

Parameter List	Type	Range	Direction	Description*
iLogicalPortID	int	1–65535 1	In	Logical port ID
iIndex	int	2	In	Configuration type index
iValue	int	2	In	Configuration value
iKey	int	3	In	Debug key

1 Maximum value for this server given by KamPortGetMaxLogPorts.

2 See FIG. 7: Controller configuration Index values for a table of indexes and values.

3 Used only for the DEBUG iIndex value. Should be set to 0.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamPortPutConfig takes a logical port ID, configuration index, configuration value, and key as parameters. It sets the port parameter specified by iIndex to the value specified by iValue. For the DEBUG iIndex value, the debug file path is C:\Temp\Debug{PORT}.txt where {PORT} is the physical comm port ID.

0KamPortGetConfig

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID
iIndex	int	2	In	Configuration type index
piValue	int *	2	Out	Pointer to configuration value

1 Maximum value for this server given by KamPortGetMaxLogPorts.

2 See FIG. 7: Controller configuration Index values for a table of indexes and values.

Return Value	Type	Range	Description
iError short	i	Error flag	

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

KamPortGetConfig takes a logical port ID, configuration index, and a pointer to a configuration value as parameters. It sets the memory pointed to by piValue to the specified configuration value.

0KamPortGetName

Parameter List	Type	Range	Direction	Description
iPhysicalPortID	int	1–65535 1	In	Physical port number
pbsPortName	BSTR *	2	Out	Physical port name

1 Maximum value for this server given by KamPortGetMaxPhysical.

2 Exact return type depends on language. It is Cstring * for C++. Empty string on error.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamPortGetName takes a physical port ID number and a pointer to a port name string as parameters. It sets the memory pointed to by pbsPortName to the physical port name such as "COMM1."

0KamPortPutMapController

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID
iControllerID	int	1–65535 2	In	Command station type ID

-continued

APPLICATION PROGRAMMING INTERFACE

iCommPortID int 1–65535 3 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 KamMiscMaxControllerID.
3 Maximum value tor 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 KamMiscGetErrorMsg).

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.

0KamPortGetMaxLogPorts

Parameter List	Type	Range	Direction	Description*
piMaxLogicalPorts	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 KamMiscGetErrorMsg).

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

0KamPortGetMaxPhysical

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 KamMiscGetErrorMsg).

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

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.

0KamCmdConnect

Parameter List	Type	Range	Direction	Description*
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

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

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

0KamCmdDisConnect

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

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

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

-continued

APPLICATION PROGRAMMING INTERFACE

0KamCmdCommand

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

1 Opaque object ID handle returned by KamDecoderPutAdd.

Return Value	Type	Range	Description
iError short	1		Error flag

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

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

A. Cab Control Commands

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

0KamCabGetMessage

Parameter List	Type	Range	Direction	Description
iCabAddress	int	1–65535 1	In	Cab address
pbsMsg	BSTR *	2	Out	Cab message string

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

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamCabGetMessage takes a cab address and a pointer to a message string as parameters. It sets the memory pointed to by pbsMsg to the present cab message.

0KamCabPutMessage

Parameter List	Type	Range	Direction	Description
iCabAddress	int	1	In	Cab address
bsMsg	BSTR	2	Out	Cab message string

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

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamCabPutMessage takes a cab address and a BSTR as parameters. It sets the cab message to bsMsg.

0KamCabGetCabAddr

Parameter List	Type	Range	Direction	Description*
lDecoderObjectID	long	1	In	Decoder object ID
piCabAddress	int *	1–65535 2	Out	Pointer to Cab address

1 Opaque object ID handle returned by KamDecoderPutAdd.
2 Maximum value is command station dependent.

Return Value	Type	Range	Descriptioni
iError short	1		Error flag

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

KamCabGetCabAddr takes a decoder object ID and a pointer to a cab address as parameters. It set the memory pointed to by piCabAddress to the address of the cab attached to the specified decoder.

0KamCabPutAddrToCab

Parameter List	Type	Range	Direction	Description
lDecoderObjectID	long	1	In	Decoder object ID
iCabAddress	int	1–65535 2	In	Cab address

1 Opaque object ID handle 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 KamMiscGetErrorMsg).

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

A. Miscellaneous Commands

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

0KamMiscGetErrorMsg

Parameter List	Type	Range	Direction	Description

-continued

APPLICATION PROGRAMMING INTERFACE

iError	int	0–65535 1	In	Error flag

1 iError = 0 for success. Nonzero indicates an error.

Return Value	Type	Range	Description
bsErrorString	BSTR	1	Error string

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

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

0KamMiscGetClockTime

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID
iSelectTimeMode	int	2	In	Clock source
piDay	int *	0–6	Out	Day of week
piHours	int *	0–23	Out	Hours
piMinutes	int *	0–59	Out	Minutes
piRatio	int *	3	Out	Fast clock ratio

1 Maximum value for this server given by KamPortGetMaxLogPorts.

2 0 - Load from command station and sync server. 1 - Load direct from server. 2 - Load from cached server copy of command station time.

3 Real time clock ratio.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamMiscGetClockTime takes the port ID, the time mode, and pointers to locations to store the day, hours, minutes, and fast clock ratio as parameters. It sets the memory pointed to by piDay to the fast clock day, sets pointed to by piHours to the fast clock hours, sets the memory pointed to by piMinutes to the fast clock minutes, and the memory pointed to by piRatio to the fast clock ratio. The servers local time will be returned if the command station does not support a fast clock.

0KamMiscPutClockTime

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID
iDay	int	0–6	In	Day of week
iHours	int	0–23	In	Hours
iMinutes	int	0–59	In	Minutes
iRatio	int	2	In	Fast clock ratio

1 Maximum value for this server given by KamPortGetMaxLogPorts. 2 Real time clock ratio.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamMiscPutClockTime takes the fast clock logical port, the fast clock day, the fast clock hours, the fast clock minutes, and the fast clock ratio as parameters. It sets the fast clock using specified parameters.

0KamMiscGetInterfaceVersion

Parameter List	Type	Range	Direction	Description
pbsInterfaceVersion	BSTR *	1	Out	Pointer to interface version string

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

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamMiscGetInterfaceVersion takes a pointer to an interface version string as a parameter. It sets the memory pointed to by pbsInterfaceVersion to the interface version string. The version string may contain multiple lines depending on the number of interfaces supported.

0KamMiscSaveData

Parameter List	Type	Range	Direction	Description
NONE				

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamMiscSaveData takes no parameters. It saves all server data to permanent storage. This command is run automatically whenever the server stops running. Demo versions of the program cannot save data and this command will return an error in that case.

0KamMiscGetControllerName

Parameter List	Type	Range	Direction	Description
iControllerID	int	1–65535 1	In	Command station type ID
pbsName	BSTR *	2	Out	Command station type name

1 See FIG. 6: Controller ID to controller name mapping for values. Maximum value for this server is given by KamMiscMaxControllerID.

2 Exact return type depends on language. It is Cstring * for C++. Empty string on error.

Return Value	Type	Range	Description
pbsName	BSTR	1	Command station type name

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamMiscGetControllerName takes a command station type ID and a pointer to a type name string as parameters. It sets the memory pointed to by pbsName to the command station type name.

0KamMiscGetControllerNameAtPort

Parameter List	Type	Range	Direction	Description
iLogicalPortID	int	1–65535 1	In	Logical port ID
pbsName	BSTR *	2	Out	Command station type name

1 Maximum value for this server given by KamPortGetMaxLogPorts.

2 Exact return type depends on language. It is Cstring * for C++. Empty string on error.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamMiscGetControllerName takes a logical port ID and a pointer to a command station type name as parameters. It sets the memory pointed to by pbsName to the command station type name for that logical port.

0KamMiscGetCommandStationValue

Parameter List	Type	Range	Direction	Description
iControllerID	int	1–65535 1	In	Command station type ID
iLogicalPortID	int	1–65535 2	In	Logical port ID
iIndex	int	3	In	Command station array index
piValue	int *	0–65535	Out	Command station value

1 See FIG. 6: Controller ID to controller name mapping for values. Maximum value for this server is given by KamMiscMaxControllerID.

2 Maximum value for this server given by KamPortGetMaxLogPorts.

3 0 to KamMiscGetCommandStationIndex.

Return Value	Type	Range	Description
iError short	1	Error flag	

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

KamMiscGetCommandStationValue takes the controller ID, logical port, value array index, and a pointer to the location to store the selected value. It sets the memory pointed to by piValue to the specified command station miscellaneous data value.

0KamMiscSetCommandStationValue

Parameter List	Type	Range	Direction	Description
iControllerID	int	1–65535 1	In	Command station type ID
iLogicalPortID	int	1–65535 2	In	Logical port ID
iIndex	int	3	In	Command station array index
iValue	int	0–65535	In	Command station Value

1 See FIG. 6: Controller ID to controller name mapping for values. Maximum value for this server is given by KamMiscMaxControllorID.

2 Maximum value for this server given by KamPortGetMaxLogPorts. 3 0 to KamMiscGetCommandStationIndex.

Return Value	Type	Range	Description

-continued

APPLICATION PROGRAMMING INTERFACE

iError short 1 Error flag

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

KamMiscSetCommandStationValue takes the controller ID, logical port, value array index, and new miscellaneous data value. It sets the specified command station data to the value given by piValue.

0KamMiscGetCommandStationIndex

Parameter List	Type	Range	Direction	Description
iControllerID	int	1–65535 1	In	Command station type ID
iLogicalPortID	int	1–65535 2	In	Logical port ID
piIndex	int	0–65535	Out	Pointer to maximum index

1 See FIG. 6: Controller ID to controller name mapping for values. Maximum value for this server is given by KamMiscMaxControllerID.

2 Maximum value for this server given by KamPortGetMaxLogPorts.

Return Value	Type	Range	Description
iError short	1		Error flag

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

KamMiscGetCommandStationIndex takes the controller ID, logical port, and a pointer to the location to store the maximum index. It sets the memory pointed to by piIndex to the specified command station maximum miscellaneous data index.

0KamMiscMaxControllerID

Parameter List	Type	Range	Direction	Description
piMaxControllerID	int *	1–65535 1	Out	Maximum controller type ID

1 See FIG. 6: Controller ID to controller name mapping for a list of controller ID values. 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 KamMiscGetErrorMsg).

KamMiscMaxControllerID takes a pointer to the maximum controller ID as a parameter. It sets the memory pointed to by piMaxControllerID to the maximum controller type ID.

0KamMiscGetControllerFacility

Parameter List	Type	Range	Direction	Description
iControllerID	int	1–65535 1	In	Command station type ID
pdwFacility	long *	2	Out	Pointer to command station facility mask

1 See FIG. 6: Controller ID to controller name mapping for values. Maximum value for this server is given by KamMiscMaxControllerID.

2
- 0 - CMDSDTA_PRGMODE ADDR
- 1 - CMDSDTA_PRGMODE_REG
- 2 - CMDSDTA_PRGMODE_PAGE
- 3 - CMDSDTA_PRGMODE_DIR
- 4 - CMDSDTA_PRGMODE_FLYSHT
- 5 - CMDSDTA_PRGMODE_FLYLNG
- 6 - Reserved
- 7 - Reserved
- 8 - Reserved
- 9 - Reserved
- 10 - CMDSDTA_SUPPORT_CONSIST
- 11 - CMDSDTA_SUPPORT_LONG
- 12 - CMDSDTA_SUPPORT_FEED
- 13 - CMDSDTA_SUPPORT_2TRK
- 14 - CMDSDTA_PROGRAM_TRACK
- 15 - CMDSDTA_PROGMAM_POFF
- 16 - CMDSDTA_FEDMODE_ADDR
- 17 - CMDSDTA_FEDMODE_REG
- 18 - CMDSDTA_FEDMODE_PAGE
- 19 - CMDSDTA_FEDMODE_DIR
- 20 - CMDSDTA_FEDMODE_FLYSHT
- 21 - CMDSDTA_FEDMODE_FLYLNG
- 30 - Reserved
- 31 - CMDSDTA_SUPPORT_FASTCLK

Return Value	Type	Range	Description

-continued

APPLICATION PROGRAMMING INTERFACE

iError short 1 Error flag

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

KamMiscGetControllerFacility takes the controller ID and a pointer to the location to store the selected controller facility mask. It sets the memory pointed to by pdwFacility to the specified command station facility mask.

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 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

Jon W. Dudas

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