Appendix B
MODEL TRAIN CONTROL SYSTEM

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

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

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

This patent is subject to a terminal disclaimer.

Appl. No.: 10/124,878
Filed: Apr. 17, 2002
Prior Publication Data
US 2002/0170458 A1 Nov. 21, 2002

Related U.S. Application Data
Continuation of application No. 09/858,222, filed on Apr. 17, 2002, now Pat. No. 6,460,467.

Int. Cl. A63H 19/00
U.S. Cl. 105/115, 246/167, 246/62

Field of Search 105/115, 1.4, 246/167, 246/177, 246/62

References Cited
U.S. PATENT DOCUMENTS
3,797,272 A 1974 Staples
3,976,272 A 1976 Murray et al.
4,307,302 A 1981 Russell
4,853,883 A 1989 Nickles et al. 348/121
5,072,940 A 1991 Maton
5,475,818 A 1995 Molyneaux et al. 701/20
5,493,642 A 1996 Dunn et al.
5,638,522 A 1997 Dunn et al.
5,681,015 A 1997 Kulik 246/167 R
5,696,689 A 1997 Okumura et al.
5,787,371 A 1998 Balakish et al. 246/167 A
5,828,979 A 1998 Plochke et al.
5,896,017 A 1999 Severson et al.
5,940,055 A 1999 Severson et al.
5,952,797 A 1999 Rossier
6,065,406 A 2000 Katzer 105/14
6,267,061 B1 2001 Katzer
6,270,406 B1 2001 Katzer

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

Primary Examiner—William A. Cuchinski, Jr.
Assistant Examiner—Olga Hernandez
Attorney, Agent, or Firm—Chernoff, Vilhauer, McClellan & Steiner, LLP

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

27 Claims, 3 Drawing Sheets
FIG. 1

RESIDENT EXTERNAL CONTROLLING INTERFACE

DIGITAL COMMAND STATIONS

COMMUNICATIONS TRANSPORT

CLIENT PROGRAM

U.S. Patent
Mar. 11, 2003
Sheet 1 of 3
US 6,530,329 B2
1
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
drawbridges. Train engines receive their power to travel on
the train track by electricity provided by a controller through
the track itself. The speed and direction of the train engine
is controlled by the level and polarity, respectively, of the
electrical power supplied to the train track. The operator
manually pushes buttons or pulls levers to cause the
switches or other electrically operated devices to function, as
desired. Such model railroad sets are suitable for a single
operator, but unfortunately they lack the capability of
adequately controlling multiple trains independently. In
addition, such model railroad sets are not suitable for being
controlled by multiple operators, especially if the operators
are located at different locations distant from the model
railroad, such as different cities.

A digital command control (DCC) system has been de-
volved 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 pro-
viding 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 commu-
nication 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 con-
nection without the additional expense associated therewith.

The foregoing and other objectives, features, and advan-
tages 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 communica-
tions 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 com-
mands representative of the first and second commands,
respectively, to a digital command station for execution on
the digitally controlled model railroad.

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

In another aspect of the present invention the first com-
mand 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 execu-
tion 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 proces-
sor 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 controller 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 acknowledgement prior to the execution of the command because the command executes quickly and most commands are sequential in nature. In other words, the execution of the next command is dependent upon proper execution of the prior command so there would be no motivation to provide an acknowledgement prior to its actual execution.

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 a 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 resident external controlling interface, and hence the model railroad.

The manner in which commands are executed for the model railroad under COM and DCOM may be as follows. The client program 14 makes requests in the same 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 DigToys Systems' software the execution of commands is slow.

The present invention came to the realization that unlike traditional distributed systems where the commands passed through 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 command 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 communications 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 communications transport 12 that is received by an asynchronous command processor 100. The asynchronous command processor 100 queries a local database storage 102 to determine if it is necessary to package a command to be transmitted to a command queue 104. The local database storage 102 primarily contains the state of the device of the model railroad, such as the speed of a train, the direction of a train, whether a drawbridge is up or down, whether a light is turned on or off, and the configuration of the model railroad layout. If the command received by the asynchronous command processor 100 is a query of the state of a device, then the asynchronous command processor 100 retrieves such information from the local database 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, whether or not the command has been executed, the combination of the asynchronous command processor 100 and the asynchronous response processor 106 both verifies the validity of the command and provides a response to the client program 14 thereby freeing up the communications transport 12 for additional commands. Without the asynchronous nature of the resident external controlling interface 16, the response to the client program 14 would be, in many circumstances, delayed thereby resulting in frustration to the operator that the model railroad is performing in a slow and painstaking manner. In this manner, the railroad 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 and the asynchronous command processor 100 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 asynchronous command processor 110 passes information to the command queue 104 that the command has been executed or the state of the device. The asynchronous response processor 106 fetches the information from the command queue 104 and provides a suitable response to the client program 14, if necessary, and updates the local database storage 102 to reflect the updated status of the railroad layout devices.

If the command fetched by the synchronous command processor 110 from the command queue 104 requires execution by external devices, such as the train engine, then the command is 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 106 clears the results
from the command queue 104 and updates the local database storage 102 and sends an asynchronous response to the client program 14, if needed. The response updates the client program 14 of the actual state of the railroad track devices, if changed, and provides an error message to the client program 14 if the devices actual state was previously improperly reported or a command did not execute properly.

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

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

The use of a single command queue 104 allows multiple instantiations of the asynchronous functionality, with one for each different client. The single command queue 104 also allows the sharing of multiple devices, multiple clients to communicate with the same device (locally or remote) in a 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 last 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 of 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 of 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

3.1 Introduction

3.2 Data Types

3.3 Commands to access the server configuration variable database
   - KamCvGetValue
   - KamCvPutValue
   - KamCvGetPosition
   - KamCvPutPosition
   - KamCvGetMaxPhysical
   - KamCvGetMaxRegister
   - KamCvGetMaxRegister

3.4 Commands to program configuration variables
   - KamProgramGetMode
   - KamProgramSetSta
   - KamProgramReadCV
   - KamProgram
   - KamProgramReadDecoderTdlDatabase
   - KamProgramDecoderFromDatabase

3.5 Commands to control all decoder types
   - KamDecoderGetMaxModels
   - KamDecoderGetModelName
   - KamDecoderSetModelToObj
   - KamDecoderSetMaxAddress
   - KamDecoderChangeOldNewAddr
   - KamDecoderMovePort
   - KamDecoderGetPort
   - KamDecoderCheckAddlnUse
   - KamDecoderGetModelFromObject
   - KamDecoderDecoderModelFacility
   - KamMiscDecoderObject
   - KamDecoderGetObjectIndex
   - KamDecoderPutIndex
   - KamDecoderPutIndex
   - KamDecoderGetMaxIndex
   - KamDecoderGetMaxSpeed

3.6 Commands to control locomotive decoders
   - KamEncoderGetSpeed
   - KamEncoderPutSpeed
   - KamEncoderPutSpeedSteps
   - KamEncoderGetFunction
   - KamEncoderPutFunction
   - KamEncoderPutFunctionMax
   - KamEncoderGetFunctionName
   - KamEncoderPutFunctionName
   - KamEncoderGetCargoMax
   - KamEncoderPutCargoParent
   - KamEncoderPutCargoChild
   - KamEncoderPutCargoRemoveObj

3.7 Commands to control accessory decoders
   - KamAccessFunction
   - KamAccessFunctionAll
   - KamAccessPutFunction
   - KamAccessPutFunctionAll
   - KamAccessPutFunctionMax
   - KamAccessPutFunction
   - KamAccessPutFunctionName
   - KamAccessPutFunctionName
   - KamAccessRegFeedback
   - KamAccessRegFeedbackAll
   - KamAccessRegFeedback
   - KamAccessRegFeedbackAll

3.8 Commands to control the command station
   - KamOpnPutTurnOutStation
   - KamOpnPutStation
   - KamOpnPutPosition
   - KamOpnPutPosition
   - KamOpnOpnPutPowerOn
   - KamOpnOpnPutPowerOff
   - KamOpnOpnPutReset
   - KamOpnOpnPutEmergencyStop
   - KamOpnOpnPutStationSteady

APPLICATION PROGRAMMING INTERFACE

5.1 Commands to configure the command station
   - KamCommandPutConfig
   - KamCommandSetConfig
   - KamCommandPutName
   - KamCommandPutMaxLogPorts
   - KamCommandPutMaxPhysical

5.2 Commands that control command flow to the command station
   - KamCommandConnect
   - KamCommandDisconnect
   - KamCommandCommand

5.3.1 Cab Control Commands
   - KamMiscCabMessage
   - KamMiscPutMessage
   - KamMiscGetCabAddr
   - KamMiscPutAddrToCab

5.3.2 Miscellaneous Commands
   - KamMiscGetErrorMsg
   - KamMiscGetClockTime
   - KamMiscPutClockTime
   - KamMiscSetInterfaceVersion
   - KamMiscSetInfoData
   - KamMiscMGetControllerName
   - KamMiscGetControllerNameAsPort
   - KamMiscGetCommandStationValue
   - KamMiscGetCommandStationValue
   - KamMiscGetCommandStationIndex
   - KamMiscMGetControllerID
   - KamMiscMGetControllerFacility

30 1. OVERVIEW

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

1. TUTORIAL

   A. Visual BASIC Throttle Example Application
      - The following application is created using the Visual BASIC source code in the next section. It controls all major locomotive functions such as speed, direction, and auxiliary functions.
      - Copyright 1998, KAM Industries. All rights reserved.

        `This is a demonstration program showing the integration of VisualBasic and Train ServerInterface.'

        `You may use this application for non-commercial usage.'

        $Desc: $   $Author: $   $Revision: $   $Log: $
APPLICATION PROGRAMMING INTERFACE

- (COM1 = COM6, LPT1, Others). You are required to set the number of the device to access an command station.
- Devices start from ID 0 to max ID (FYI: devices do not necessarily have to be serial channel. Always check the name of the device before you use it as well as the maximum number of devices supported.
- The Command
  - EngComm.KamPort.GetMaxPhysical(MaxPhysical, IsSerial, IPacket) provides means that. MaxPhysical = IsSerial + IPacket + x Other
- Controller - These are command the command station
  - like LENS, Digitrax, Narrowbloc, EasyDCX, Marklin... It is recommended that you check the command station ID before you use it.
- Errors - All commands return an error status. If the error value is non-zero, then the return arguments are invalid. If the value is zero, non zero error means command was not executed. To get the error message, you need to call KamMisc.ErrorMessage 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 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 KamComm.PayAdd ... 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 LogicalPort, Controller, ComPort, IPortWait, IPortWaitLog, IPortReturn, IPortCmd, IPortCmdLog, IPortCmdReturn, IPortCmdWait
- Dim EngineObject As Long, DEncoderClass As Integer, DEncoderType As Integer
- Dim MaxController As Long
- Dim MaxLogical As Long, IMaxPhysical As Long, IMaxSerial

Form level function

- Turn of the initial buttons
- Set interface information

Private Sub Form_Load()
  Dim strVer As String, strComm As String, strCtrl As String
  Dim Error As Integer
  'Get the interface version information
  SetButtonState (False)
  If (Error) Then
    MsgBox ("Train server not loaded. Check DCOM-95")
  Else
    LogicalPort = 0
    LogPortCaption = "LogicalPort"
    ComPortCaption = "777";
    Controller.Caption = "Unknown"
  EndIf
  MsgBox ("Simulation(0)\nTrain Server - " & strVer)
  'Configuration information; Only need to change these values to use a different controller.
  'UNKNOWN 0 // Unknown control type
  'SIMULAT 1 // Interface simulator

  5
  'LENZ_1x 2 // Lenz serial support module
  'LENZ_2x 3 // Lenz serial support module
  'DIGIT_D7200 4 // Digitrax direct drive support using DT200
  'DIGIT_DCS100 5 // Digitrax direct drive support using DCS100
  'MASTERSERIES 6 // General Coast engineering master Series
  'SYSTEMONE 7 // System One
  'RAMFLEX 8 // RAMflex system
  'DYNATROL 9 // Dynatrol system
  'Northeast binary 10 // North Coast binary
  'SERCAL 11 // SERCAL interface
  'EASYDCX 12 // SERCAL interface
  'MRK650 13 // 650 Marklin Interface
  'MRK620 14 // 620 Marklin hybrid Interface (AC)
  'ZTC 15 // ZTC System line
  'DIGIT_FRI 16 // Digitrax direct drive support using FRI
  'DIRECT 17 // Direct drive interface

  LogicalPort = 1 "Select logical port 1 for
  iController = 1 "Select controller from the list above.
  iComPort = 0 "use COM1; 0 means com1 (Digitrax must
  'Digitrax Baud rate requires 16.4K'
  'Most COM ports above Com2 do not
  'support 16.4K. Check the
  'manufacturer of your smart card
  'for the baud rate. Keep in mind that
  'Dumb com cards with serial port
  'support Com1 - Com4 only support
  '2 com ports (like com1:com2
  'or com1: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? 0 // Baud rate is 300
  "1? 1 // Baud rate is 1200
  "2? 2 // Baud rate is 2400
  "3? 3 // Baud rate is 4800
  "4? 4 // Baud rate is 9600
  "5? 5 // Baud rate is 14.4
  "6? 6 // Baud rate is 16.4
  "7? 7 // Baud rate is 19.2
  IPortRate = 4
  'Parity values 0-4 => no, odd, even, mark, space
  'InParity = 0
  'Stop bit 0,1,2 => 1, 1.5, 2
  'IPortStop = 0
  'IPortReturn = 10
  'IPortWaitLog = 2048
  'IPortFlow = 0
  'Data bits 0-3 = 7 Bits, 1-8 bits
  'IPortData = 1
  'Display the port and controller information
  IPortError = EngComm.KamPort.GetMaxLogPort(1)
  If (IPortError) Then
    MsgBox ("Com port out of range")
  EndIf
  GetPortName(ComPort, strCom)
  If (LogicalPort > IMaxSerial) Then MsgBox ("Com port out of range")
  If (LogicalPort > IMaxSerial) Then MsgBox ("Com port out of range")
APPLICATION PROGRAMMING INTERFACE

("Logical port out of range")
EndIf

If Display values is Throttle,
LogPort.Caption = rLogicalPort
ComPort.Caption = rComPort
Controller.Caption = rController
End Sub

***************

Send Command

Note:
1. 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 Error, IsSpeed As Integer
If Not Connect.Enabled Then
	 TimToolsInterface is a fake interface.
	 This means that you need to set up the CV's or other operations first, then execute the 'command.
	 IsSpeed = Speed.Text
EngCmd.KamEngPutFunction(EngineObject, 0, F0, Value)
Open Error = EngCmd.KamEngPutFunction(EngineObject, 1, F1, Value)
Error = EngCmd.KamEngPutFunction(EngineObject, 2, F2, Value)
Error = EngCmd.KamEngPutFunction(EngineObject, 3, F3, Value)
Error = EngCmd.KamEngPutSpeed(EngineObject, IsSpeed, Direction.Text)
If Error = 0 Then Error = EngCmd.KamEngPutCommand(EngineObject)
SetError (Error)
End If

End Sub

***************

Connect Controller

***************

Private Sub Connect_Click()
Dim Error As Integer

These are the index values for setting up the port

\* PORT_RETRANSM 0 / Retrans index
\* PORT_RATE 1 / Retrans index
\* PORT_PARTY 2 / Retrans index
\* PORT_STOP 3 / Retrans index
\* PORT_WATCHDOG 4 / Retrans index
\* PORT_FLOW 5 / Retrans index
\* PORT_DATABITS 6 / Retrans index
\* PORT_DEBUG 7 / Retrans index
\* PORT_PARALLEL 8 / Retrans index

These are the index values for setting up the...

\* PORT_RETRANSM 0 / Retrans index
\* PORT_RATE 1 / Retrans index
\* PORT_PARTY 2 / Retrans index
\* PORT_STOP 3 / Retrans index
\* PORT_WATCHDOG 4 / Retrans index
\* PORT_FLOW 5 / Retrans index
\* PORT_DATABITS 6 / Retrans index
\* PORT_DEBUG 7 / Retrans index
\* PORT_PARALLEL 8 / Retrans index

If Error = 0 Then
Error = EngCmd.KamPortPutConfig(rLogicalPort, 0, rPortReTrans, 0, rPortRate, 0, rPortParity, 0, rPortStop, 0, rPortWatchdog, 0, rPortFlow, 0, rPortDataBits, 0, rPortDebug, 0, rPortParallel, 0)
End If

End Sub

***************

APPLICATION PROGRAMMING INTERFACE

iError = EngCmd.KamPortPutConfig(iLogicalPort, 4
iPortWatchdog, 0) setting PORT...
US 6,530,329 B2

APPLICATION PROGRAMMING INTERFACE

-continued

SetError (Status)
if (EngineObject) Then
  Command.Enabled = True
  TurnOn the control
  Throttle.Enabled = True
  Turn on the throttle
else
  MsgBox ("Address not set, check error message")
  End If
End Sub
-continued
<table>
<thead>
<tr>
<th>Case 3:06-cv-01905-JSW</th>
<th>Document 191-3</th>
<th>Filed 12/12/2007</th>
<th>Page 15 of 27</th>
</tr>
</thead>
</table>

**APPLICATION PROGRAMMING INTERFACE**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
<td>Address</td>
</tr>
<tr>
<td>iCVRegist</td>
<td>0-1024</td>
<td>2</td>
<td>In</td>
<td>CV number</td>
</tr>
<tr>
<td>pCVValStatus</td>
<td>Int *</td>
<td>3</td>
<td>Out</td>
<td>Old decoder programming status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td>Description</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMaxModels</td>
<td>Int</td>
<td>1</td>
<td>Out</td>
<td>Pointer to Max model ID</td>
</tr>
<tr>
<td>return</td>
<td>Max value for this server given by KmdDecoderGetMaxModels.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td>Description</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMaxModels</td>
<td>Int</td>
<td>1</td>
<td>Out</td>
<td>Pointer to Max model ID</td>
</tr>
<tr>
<td>return</td>
<td>Max value for this server given by KmdDecoderGetMaxModels.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td>Description</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMaxModels</td>
<td>Int</td>
<td>1</td>
<td>Out</td>
<td>Pointer to Max model ID</td>
</tr>
<tr>
<td>return</td>
<td>Max value for this server given by KmdDecoderGetMaxModels.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td>Description</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMaxModels</td>
<td>Int</td>
<td>1</td>
<td>Out</td>
<td>Pointer to Max model ID</td>
</tr>
<tr>
<td>return</td>
<td>Max value for this server given by KmdDecoderGetMaxModels.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td>Description</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMaxModels</td>
<td>Int</td>
<td>1</td>
<td>Out</td>
<td>Pointer to Max model ID</td>
</tr>
<tr>
<td>return</td>
<td>Max value for this server given by KmdDecoderGetMaxModels.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td>Description</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pMaxModels</td>
<td>Int</td>
<td>1</td>
<td>Out</td>
<td>Pointer to Max model ID</td>
</tr>
<tr>
<td>return</td>
<td>Max value for this server given by KmdDecoderGetMaxModels.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case 3:06-cv-01905-JSW     Document 191-3      Filed 12/12/2007     Page 18 of 27

US 6,530,329 B2

continued

APPLICATION PROGRAMMING INTERFACE

iError short 1 Error flag
1: iError = 0 for success. Nonzero is an error number (see KaiMiscGetErrorMsg).
KaiEngGetSpeed takes the decoder object ID and point to the location to store the speed and the memory pointed to by lpDirection to the locomotive direction.

@KaiEngPutSpeed
Parameter List Type Range Direction Description
DecoderObjectID long 1 In Decoder object ID
lpSpeed int * 2 In Locomotive speed
lpDirection int * 3 In Locomotive direction

1: opaque object ID handle returned by KaiDecoderPutAdd.
2: Speed steps 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 maximum step for all modes.
3: Forward is boolean TRUE and reverse is boolean FALSE.

Return Value Type Range Description
Error short 1 Error flag
1: iError = 0 for success. Nonzero is an error number (see KaiMiscGetErrorMsg).
KaiEngGetSpeed takes the decoder object ID, new locomotive speed, and new locomotive direction as parameters. It sets the locomotive database speed to the speed and the locomotive database direction to the new direction. Note: This command only changes the locomotive database. The data is not sent to the decoder until execution of the KaiCmdCommand command. Speed is set to the maximum possible for the decoder if spdSpeed exceeds the decoders range.

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

1: opaque object ID handle returned by KaiDecoderPutAdd.
2: Return Value Type Range Description
Error short 1 Error flag
1: iError = 0 for success. Nonzero is an error number (see KaiMiscGetErrorMsg).
KaiEngGetSpeedSteps 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.

@KaiEngPutSpeedSteps
Parameter List Type Range Direction Description
DecoderObjectID long 1 In Decoder object ID
lpSpeedSteps int * 14,28,128 In Locomotive speed steps

1: opaque object ID handle returned by KaiDecoderPutAdd.
2: Return Value Type Range Description
Error short 1 Error flag
1: iError = 0 for success. Nonzero is an error number (see KaiMiscGetErrorMsg).
KaiEngPutSpeedSteps 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 lpSpeedSteps. Note: This command only changes the locomotive database. The data is not sent to the decoder until execution of the KaiCmdCommand command. KaiDecoderGetMaxSpeed 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.

@KaiEngGetFunction
Parameter List Type Range Direction Description
DecoderObjectID long 1 In Decoder object ID
lpFunction int * 0-4-2 In Function ID number

1: opaque object ID handle returned by KaiDecoderPutAdd.
2: Function is boolean TRUE and inactive is boolean FALSE.
3: Return Value Type Range Description
Error short 1 Error flag
10: iError = 0 for success. Nonzero is an error number (see KaiMiscGetErrorMsg).
KaiEngGetFunction 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.

@KaiEngPutFunction
Parameter List Type Range Direction Description
DecoderObjectID long 1 In Decoder object ID
FunctionID int 0-4-2 In Function ID number
lpFunction int * 3 In Function value

1: opaque object ID handle returned by KaiDecoderPutAdd.

Case 3:06-cv-01905-JSW     Document 191-3      Filed 12/12/2007     Page 19 of 27

### APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
</tr>
<tr>
<td>FunctionID</td>
<td>int</td>
<td>0-3</td>
<td>In</td>
</tr>
<tr>
<td>pBufFilenameString</td>
<td>BSTR</td>
<td>3</td>
<td>Out</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. **Opaque object ID handle returned by**
   - **KamDecoderPutAdd**.

2. **FL is 0.** FL-F8 are 1-8 respectively. Maximum for this decoder is given by **KamEngGetFunctionMax**.
   - Exact return type depends on language. It is **C++,** **Empty string on error.**
   - **Return Value**
     - **Error** short 1 **Error Flag**
     - **Type** short 1 **Range**
     - **Description**

3. **If error** 0 for success, **Nonzero** is an error number
   - **KamEngGetFunctionName** takes a decoder object ID, **KamEngGetFunctionMax**.

4. **Opaque object ID handle returned by**
   - **KamDecoderPutAdd**.

5. **FL is 0.** FL-F8 are 1-8 respectively. Maximum for this decoder is given by **KamEngGetFunctionMax**.
   - Exact return type depends on language. It is **LPCTSTR for C++**.
   - **Return Value**
     - **Type** short 1 **Range**
     - **Description**

6. **If error** 0 for success, **Nonzero** is an error number
   - **KamEngGetFunctionName** takes a decoder object ID, **KamEngGetFunctionMax**.
     - **Opaque object ID handle returned by**
     - **KamDecoderPutAdd**.

7. **Command station dependent.**
   - **Return Value**
     - **Type** short 1 **Range**
     - **Description**
   - **Error short** 1 **Error Flag**
   - **Type** short 1 **Range**
   - **Description**
   - **Parameters ConsistMax**
     - **Opaque object ID handle returned by**
     - **KamDecoderPutAdd**.

8. **IdCCParensObjID** long 1 In Parent decoder object ID

9. **IdCCBiasAdd** int 2 In Alias decoder address
   - **Opaque object handle returned by**
   - **KamDecoderPutAdd**.

10. **1-127** for short locomotive addresses. **1-10239 for long locomotive decoders.**

11. **Return Value**
    - **Type** short 1 **Range**
    - **Description**
    - **Error short** 1 **Error Flag**

12. **KamEngPutConsistParent** takes the parent object ID and an alias address as parameters. It makes the decoder object **IDCCConsistParent**.
    - **Opaque object ID handle returned by**
    - **KamDecoderPutAdd**.

13. **IdCCFAssAdd** int 2 In Parent decoder object ID

14. **IdCCFAssAdd** int 2 In Parent decoder object ID

15. **IdCCFAssAdd** int 2 In Parent decoder object ID

16. **IdCCFAssAdd** int 2 In Parent decoder object ID

17. **IdCCFAssAdd** int 2 In Parent decoder object ID

18. **IdCCFAssAdd** int 2 In Parent decoder object ID

19. **IdCCFAssAdd** int 2 In Parent decoder object ID

20. **IdCCFAssAdd** int 2 In Parent decoder object ID

21. **IdCCFAssAdd** int 2 In Parent decoder object ID

22. **IdCCFAssAdd** int 2 In Parent decoder object ID

23. **IdCCFAssAdd** int 2 In Parent decoder object ID

24. **IdCCFAssAdd** int 2 In Parent decoder object ID

25. **IdCCFAssAdd** int 2 In Parent decoder object ID

26. **IdCCFAssAdd** int 2 In Parent decoder object ID

27. **IdCCFAssAdd** int 2 In Parent decoder object ID

28. **IdCCFAssAdd** int 2 In Parent decoder object ID

29. **IdCCFAssAdd** int 2 In Parent decoder object ID

30. **IdCCFAssAdd** int 2 In Parent decoder object ID

31. **IdCCFAssAdd** int 2 In Parent decoder object ID

32. **IdCCFAssAdd** int 2 In Parent decoder object ID

33. **IdCCFAssAdd** int 2 In Parent decoder object ID

34. **IdCCFAssAdd** int 2 In Parent decoder object ID

35. **IdCCFAssAdd** int 2 In Parent decoder object ID

36. **IdCCFAssAdd** int 2 In Parent decoder object ID

37. **IdCCFAssAdd** int 2 In Parent decoder object ID

38. **IdCCFAssAdd** int 2 In Parent decoder object ID

39. **IdCCFAssAdd** int 2 In Parent decoder object ID

40. **IdCCFAssAdd** int 2 In Parent decoder object ID

41. **IdCCFAssAdd** int 2 In Parent decoder object ID

42. **IdCCFAssAdd** int 2 In Parent decoder object ID

43. **IdCCFAssAdd** int 2 In Parent decoder object ID

44. **IdCCFAssAdd** int 2 In Parent decoder object ID

45. **IdCCFAssAdd** int 2 In Parent decoder object ID

46. **IdCCFAssAdd** int 2 In Parent decoder object ID

47. **IdCCFAssAdd** int 2 In Parent decoder object ID

48. **IdCCFAssAdd** int 2 In Parent decoder object ID

49. **IdCCFAssAdd** int 2 In Parent decoder object ID

50. **IdCCFAssAdd** int 2 In Parent decoder object ID

51. **IdCCFAssAdd** int 2 In Parent decoder object ID

52. **IdCCFAssAdd** int 2 In Parent decoder object ID

53. **IdCCFAssAdd** int 2 In Parent decoder object ID

54. **IdCCFAssAdd** int 2 In Parent decoder object ID

55. **IdCCFAssAdd** int 2 In Parent decoder object ID

56. **IdCCFAssAdd** int 2 In Parent decoder object ID

57. **IdCCFAssAdd** int 2 In Parent decoder object ID

58. **IdCCFAssAdd** int 2 In Parent decoder object ID

59. **IdCCFAssAdd** int 2 In Parent decoder object ID

60. **IdCCFAssAdd** int 2 In Parent decoder object ID

61. **IdCCFAssAdd** int 2 In Parent decoder object ID

62. **IdCCFAssAdd** int 2 In Parent decoder object ID

63. **IdCCFAssAdd** int 2 In Parent decoder object ID

64. **IdCCFAssAdd** int 2 In Parent decoder object ID

65. **IdCCFAssAdd** int 2 In Parent decoder object ID
Case 3:06-cv-01905-JSW     Document 191-3      Filed 12/12/2007     Page 20 of 27

US 6,530,539 B2

-continued

APPLICATION PROGRAMMING INTERFACE

1. Opaque object ID handle returned by
   *KanDecoderPutAdd.
   2. Each bit represents a single function state.
   Maximum for this decoder is given by
   *KanAccGetFunctionMax.
   Return Value Type Range Description
   0xFF Error short 1 Error flag
   1 0x0 for success. Nonzero is an error number
   (see *KanMiscGetErrorMsg).
   *KanAccGetFunctionMax takes the decoder object ID and a
   pointer to a bit mask as parameters. It sets each bit in
   the memory pointed to by pValue to the corresponding
   function state.
   0KanAccPutFunction
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   iFunctionID int 0-31 2 In Function ID number
   iFunction int 3 In Function value
   5 0x0 opaque object ID handle returned by
   *KanDecoderPutAdd.
   2 0x0 Maximum for this decoder is given by
   *KanAccGetFunctionMax.
   3 0x0 Function active is boolean TRUE and inactive is
   boolean FALSE.
   Return Value Type Range Description
   iError short 1 Error flag
   1 0x0 for success. Nonzero is an error number
   (see *KanMiscGetErrorMsg).
   *KanAccPutFunction takes the decoder object ID, a function
   ID, and a new function state as parameters. It sets the
   specified accessor database function state to iFunction.
   Note: This command only changes the accessor database.
   The data is not sent to the decoder until execution of the
   *KanCmdCommand command.
   0KanAccPutFunctionAll
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   iValue int 2 In Pointer to function state array
   1 0x0 opaque object ID handle returned by
   *KanDecoderPutAdd.
   2 0x0 Each bit represents a single function state.
   Maximum for this decoder is given by
   *KanAccGetFunctionMax.
   Return Value Type Range Description
   iError short 1 Error flag
   1 0x0 for success. Nonzero is an error number
   (see *KanMiscGetErrorMsg).
   *KanAccPutFunctionAll 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
   *KanCmdCommand command.
   0KanAccGetFunctionMax
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   iMaxFunction int * 0-31 2 Out Pointer to maximum
   function number
   1 0x0 opaque object ID handle returned by
   *KanDecoderPutAdd.
   2 0x0 Maximum for this decoder is given by
   *KanAccGetFunctionMax.
   Return Value Type Range Description
   iError short 1 Error flag
   1 0x0 for success. Nonzero is an error number
   (see *KanMiscGetErrorMsg).
   *KanAccGetFunctionMax takes a decoder object ID and
   pointer to the maximum function number as parameters. It
   sets the memory pointed to by pMaxFunction to the
   maximum possible function number for the specified
   decoder.
   0KanAccGetName
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   pFuncNameString BSTR * 2 Out Function name
   1 0x0 opaque object ID handle returned by
   *KanDecoderPutAdd.
   2 0x0 Each parameter type depends on language. It is
   LPCTSTR for C++.
   Return Value Type Range Description
   iError short 1 Error flag
   1 0x0 for success. Nonzero is an error number
   (see *KanMiscGetErrorMsg).
   *KanAccGetObjectName takes a decoder object ID and a
   pointer to a string as parameters. It sets the memory pointed to by
   pObjectNameString to the name of the accessor.
   0KanAccPutName
   Parameter List Type Range Direction Description
   DecoderObjectID long 1 In Decoder object ID
   pAccNameString BSTR * 2 Out Accessory name
   1 0x0 opaque object ID handle returned by
   *KanDecoderPutAdd.
APPLICATION PROGRAMMING INTERFACE

Case 3:06-cv-01905-JSW     Document 191-3      Filed 12/12/2007     Page 22 of 27
APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iCommPortID</td>
<td>int 1-65535 3 In Physical comm port ID</td>
</tr>
<tr>
<td>iCommPortGetMaxLogPots</td>
<td>1 Maximum value for this server given by iCommPortGetMaxLogPots</td>
</tr>
<tr>
<td>iCommPortGetMaxLogPots</td>
<td>2 See FIG. 6: Controller ID to controller name mapping for values. Maximum value for this server is given by iCommPortGetMaxLogPots</td>
</tr>
<tr>
<td>iCommPortGetMaxPhysical</td>
<td>3 Maximum value for this server given by iCommPortGetMaxPhysical</td>
</tr>
<tr>
<td>iCommPortPutMapController</td>
<td>takes a logical port ID, a command station type ID, and a physical communications port ID as parameters. It maps iCommPortID to iCommPortID for the type of connected station specified by iControllerID.</td>
</tr>
<tr>
<td>iCommPortGetMaxLogPots</td>
<td>Parameter List</td>
</tr>
<tr>
<td>iMaxLogicalPots int*</td>
<td>1</td>
</tr>
<tr>
<td>iError short</td>
<td>1</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number (see iCommPortGetMaxLogPots).</td>
<td></td>
</tr>
<tr>
<td>iCommPortGetLogPots</td>
<td>Parameter List</td>
</tr>
<tr>
<td>iMaxLogicalPots int*</td>
<td>1</td>
</tr>
<tr>
<td>iError short</td>
<td>1</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number (see iCommPortGetLogPots).</td>
<td></td>
</tr>
<tr>
<td>iCommPortGetMaxPhysical</td>
<td>Parameter List</td>
</tr>
<tr>
<td>iMaxPhysical int*</td>
<td>1</td>
</tr>
<tr>
<td>iError short</td>
<td>1</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number (see iCommPortGetMaxPhysical).</td>
<td></td>
</tr>
<tr>
<td>iCommPortPutMaxPhysical</td>
<td>Parameter List</td>
</tr>
<tr>
<td>iMaxParallel int*</td>
<td>1</td>
</tr>
<tr>
<td>iError short</td>
<td>1</td>
</tr>
<tr>
<td>iError = 0 for success. Nonzero is an error number (see iCommPortPutMaxPhysical).</td>
<td></td>
</tr>
</tbody>
</table>

A. Commands that control command flow to the command stations

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

**iCommPortConnect**
- **Function**: Takes a logical port ID as a parameter. It connects the server to the specified command station.
- **Arguments**:
  - **LogicalPortID**: int 1-65535 1 In Logical port ID
- **Return Value**:
  - **Type**: Range Description
  - **iError short**: 1 Error flag
  - **iError = 0 for success. Nonzero is an error number (see iCommPortGetLogPots).**
  - **iCommPortGetLogPots**: Parameter List | Type | Range | Direction | Description |
  - **iMaxLogicalPots int***: 1 Out Maximum logical port ID
  - **iError short**: 1 Error flag
  - **iError = 0 for success. Nonzero is an error number (see iCommPortGetLogPots).**
  - **iCommPortGetMaxPhysical**: Parameter List | Type | Range | Direction | Description |
  - **iMaxPhysical int***: 1 Out Maximum physical port ID
  - **iError short**: 1 Error flag
  - **iError = 0 for success. Nonzero is an error number (see iCommPortGetMaxPhysical).**
  - **iCommPortPutMaxPhysical**: Parameter List | Type | Range | Direction | Description |
  - **iMaxParallel int***: 1 Out Maximum parallel port ID
  - **iError short**: 1 Error flag
  - **iError = 0 for success. Nonzero is an error number (see iCommPortPutMaxPhysical).**

B. Miscellaneous Commands

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

**iCommPortGetErrorMg**

**Parameter List** | Type | Range | Direction | Description |
| iMaxLogPots | int* | 1 | Out | Maximum logical port ID |
| iError short | 1 | Error flag |
| iError = 0 for success. Nonzero is an error number (see iCommPortGetLogPots).**
| **iCommPortPutAddrs**

**Parameter List** | Type | Range | Direction | Description |
| iMaxLogPots | int* | 1 | Out | Maximum logical port ID |
| iError short | 1 | Error flag |
| iError = 0 for success. Nonzero is an error number (see iCommPortGetLogPots).**
APPLICATION PROGRAMMING INTERFACE

Error short 1 Error flag
1 ifError = 0 for success. Nonzero is an error number
(see KsmGetMaxErrMsg).
KsmGetMaxCommandStationValue 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 pValue.

KsmGetMaxCommandStationIndex
Parameter List Type Range Direction Description
controllerID int 1-65535 1 In Command station
logicalPortID int 1-65535 2 In Logical port ID
pindex int 0-65535 Out Pointer to maximum

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

Return Value Type Range Description
error short 1 Error flag
1 ifError = 0 for success. Nonzero is an error number
(see KsmGetMaxErrMsg).
KsmMaxCommandStationIndex takes the controller ID, logical port, and a pointer to the location to store the
maximum index. It sets the memory pointed to by pindex
to the specified command station maximum miscellaneous
data index.

KsmMaxControllerID
Parameter List Type Range Direction Description
controllerID int * 1-65535 1 Out Maximum
controller type ID

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

Parameter List Type Range Direction Description
controllerID int 1-65535 1 In Command station
type ID
pCmdFacility long * 2 Out Pointer to command
station facility mask

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 pro-
gram to said interface; and
(c) sending third and fourth commands from said interface
representative of said first and second commands, respec-
tively, 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 inter-
face 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 inter-
face 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 stan-
tions; 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 con-
trolled 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 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 asynchronous command processor into a suitable format for execution by a digital command station for said digitally controlled model railroad.

24. The method of claim 23 further comprising the steps of:
   (a) receiving responses from said digital command station; and
   (b) updating a first database of the state of said digitally controlled model railroad based upon said responses from said digital command station.

25. The method of claim 24, further comprising the steps of:
   (a) sending a first response to said command queue from said asynchronous command processor where said asynchronous command processor considers said command queue the intended destination device of said first response; and
   (b) processing said first response by said asynchronous command processor into a suitable format for said first program.

26. The method of claim 25, further comprising the step of updating a second database of the state of said digitally controlled model railroad by said asynchronous command processor based upon said first response from said asynchronous command processor.

27. The method of claim 26, further comprising the step of querying said second database by said asynchronous command processor providing said acknowledgment to said first program providing the information requested and not sending said first command to said command queue.

* * * * *
CERTIFICATE OF CORRECTION

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

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

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

Signed and Sealed this

Fifteenth Day of March, 2005

[Signature]

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