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

27 Claims, 3 Drawing Sheets
MODEL TRAIN CONTROL SYSTEM

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

BACKGROUND OF THE INVENTION

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

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

A digital command control (DCC) system has been developed to provide additional controllability of individual train engines and other electrical devices. Each device the operator desires to control, such as a train engine, includes an individually addressable digital decoder. A digital command station (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 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 commands to the resident external controlling interface, and hence the model railroad. In addition by queuing by commands at a single resident external controlling interface permits controlled execution of the commands by the digitally controlled model railroad, would may otherwise conflict with one another.

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

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

In yet another aspect of the present invention the first command is transmitted from a first client program to a first processor through a first communications transport. The first command is received at the first processor. The first processor provides an acknowledgment to the first client program through the first communications transport indicating that
US 6,530,329 B2

3

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 controller interface to the digital command stations occur in a synchronous manner, such as a first-in-first-out queue. 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 controller 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. A description of COM (common object model) and DCOM (distributed common object model) is provided by Chappel in a book entitled Understanding ActiveX and OLE, Microsoft Press, and is incorporated by reference herein.

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

The manner in which commands are executed for the model railroad under COM and DCOM may be as follows. The client program 14 makes requests in a synchronous manner using COM/DCOM to the resident external 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 inventor came to the realization that unlike traditional distributed systems where the commands passed through a communications transport are executed nearly instantaneously by the server and then an acknowledgement is returned to the client, the model railroad application involves the use of extremely slow real-time interfaces between the digital command stations and the devices of the model railroad. The present inventor came to the further realization that in order to increase the apparent speed of execution to the client, other than using high-speed communication interfaces, the resident external controller interface 16 should receive the command and provide an
acknowledgement to the client program 12 in a timely manner before the execution of the command by the digital command stations 18. Accordingly, the execution of commands provided by the resident external controlling interface 16 to the digital command stations 18 occur in a synchronous manner, such as a first-in-first-out manner. The COM and DCOM communications transport 12 between the client program 14 and the resident external controlling interface 16 is operating in an asynchronous manner, namely providing an acknowledgement thereby releasing the communications transport 12 to accept further communications prior to the actual execution of the command. The combination of the synchronous and the asynchronous data communication for the commands provides the benefit that the operator considers the commands to occur nearly instantly while permitting the resident external controlling interface 16 to verify that the command is properly 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 additionally include additional information from the local database storage 102 to complete the client program 14 request, if necessary. Together with packaging the command for the command queue 104, the asynchronous command processor 100 provides a command to the asynchronous request processor 106 to provide a response to the client program 14 indicating that the event has occurred, even though such an event has yet to occur on the physical railroad layout.

As such, it can be observed that whether or not the command is valid, whether or not the information requested by the command is available, and whether or not the command has executed, the combination of the asynchronous command processor 100 and the asynchronous response processor 106 both verify 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 queue 104 and provides a suitable response to the client program 14, if necessary, and updates the local database storage 102 to reflect the updated status of the railroad layout devices.

If the command fetched by the synchronous command processor 110 from the command queue 104 requires execution by external devices, such as the train engine, then the command is passed to one of several external device control logic 114 blocks. The external device control logic 114 processes the command from the synchronous command processor 110 and initiates 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 invention 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 in which 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.
3.1 Introduction
3.2 Data Types
3.3 Commands to access the server configuration variable database
   KamCVCGet_Value
   KamCVCPut_Value
   KamCVCGetEnable
   KamCVCPutEnable
   KamCVCGet_Name
   KamCVCPutMinRegister
   KamCVCGetMaxRegister
3.4 Commands to program configuration variables
   KamProgramSetMode
   KamProgramGet_Status
   KamProgramRead_CV
   KamProgramSet_CV
   KamProgramReadDecoderToDatabase
   KamProgramDecoderFromDatabase
3.5 Commands to control all decoder types
   KamDecoderGetMax_Modes
   KamDecoderGetModel_Name
   KamDecoderSetModelTo_Obj
   KamDecoderGetMax_Addresses
   KamDecoderChangeUlNew_Addr
   KamDecoderMove_Port
   KamDecoderGetPort
   KamDecoderCheck_AddrInUse
   KamDecoderGetModelError
   KamDecoderGetModelFrom_Obj
   KamDecoderGetModel_Facility
   KamDecoderGet_ObjCount
   KamDecoderPutObjAddr
   KamDecoderPutAdd
   KamDecoderPutDel
   KamDecoderGet_Mgr_Name
   KamCVGetEnableMode
   KamDecoderGetMaxSpeed
3.6 Commands to control locomotive decoders
   KamLocGetSpeed
   KamLocGetSpeedSteps
   KamLocGetFunction
   KamLocGetFunction_MAX
   KamLocGet_Name
   KamLocPut_Name
   KamLocGetFunction_Name
   KamLocGetCosine_MAX
   KamLocGetCosine_Parent
   KamLocGetCosine_Child
   KamLocPutCosine_Remove_Obj
3.7 Commands to control accessory decoders
   KamAccGetFunction
   KamAccGetFunctionAll
   KamAccPutFunction
   KamAccPutFunctionAll
   KamAccGetFunction_MAX
   KamAccGet_Name
   KamAccPut_Name
   KamAccGetFunction_Name
   KamAccPutFunction_Name
   KamAccRegFeedback
   KamAccRegFeedbackAll
   KamAccDelFeedback
   KamAccDelFeedbackAll
3.8 Commands to control the command station
   KamOpTurnOnStation
   KamOpPutStation
   KamOpPutStopStation
   KamOpPutPowerOn
   KamOpPutPowerOff
   KamOpPutReset
   KamOpPutEmergencyStop
   KamOpGetStationStatus
   "-continued"
APPLICATION PROGRAMMING INTERFACE

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

Display values in throttle
LogPort.Capton = LogPort
ComPort.Capton = strCom
Controller.Capton = strCtrl
End Sub

'-----------------------------
'This command can only be sent if the following is true
'Controller is not connected
'Port has not been mapped
'Not slaveware version of application (Shareware)
'Always set to 130

Write Display Log Debug
File 'Win Level Value
'1 = 4, = 7 -> LEVEL1 -- put packets into
'1 = 4, = 11 -> LEVEL2 -- Status messages
'send to window
'1 = 12, = 19 -> LEVEL3 --
'1 = 2 = 32 = 35 -> LEVEL4 -- All system
'synophones/critical sections
'1 = 2 = 64 = 67 -> LEVEL5 -- detailed
'debugging information
'1 = 2 = 128 = 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
read users.

Note: 1. This does effect the performance of you
system; 130 is a save value for debug
2. The Digitax controls codes displayed are
encrypted. The information that you
determine from the control codes is that
information is sent (0) and a response is
received (R)

DebugMode = 130
Value = Value.Text Display value for reference

Error = EngCmd.KamPortPutConfig(LogicalPort, 7, IDebug,
0) setting PORT_DEBUG

Private Sub Connect_Click()
Dim Error As Integer

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

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

Once we make a connection, we use the
LogicalObject
as the reference object to send control information
If (Address.Text = "1") Then
iStatus = EngCmd.KamPortPutAdd(Address.Text, 
LogicalPort, 1, LogicalPort, 0, 
IDecoderType, LogicalObject)
US 6,530,329 B2

15

APPLICATION PROGRAMMING INTERFACE

SetError (Status)
If (EngineObject) 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 than 0 and
less than 128”) End If
End Sub

Private Sub Disconnect_Click()
  Dim Error As Integer
  Set Error = Err
  smallest undefinable value
  Set ButtonState (False)
End Sub

APPLICATION PROGRAMMING INTERFACE

Private Sub SetError(ByVal Error As Integer)
  Dim ErrorMessage As String
  Dim Status As String
  'This shows how to retrieve a sample error message
  Status = Err
  ErrorMessage = Err
  Set Error = Err
End Sub

Private Sub SetButtonState(ByVal State As Boolean)
  'We set the state of the buttons, either connected or
disconnected
  If (State) Then
    Connect.Enabled = True
    Disconnect.Enabled = False
  Else
    Connect.Enabled = False
    Disconnect.Enabled = True
End If

Private Sub Init()
  Private Sub Disconnect_Click()
    Dim Error As Integer
    SetError (Error)
    DisconnectButton (False)
  End Sub

End Sub

End Sub

16

APPLICATION PROGRAMMING INTERFACE

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

Throttle slider control

Private Sub Throttle_Click()
  If (EngineObject) Then
    If (Throttle.Value > 0) Then
      SpeedText = Throttle.Value
    Else
      End If
  End If
  SetButtonState
End Sub

10

APPLICATION PROGRAMMING INTERFACE

L. 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.

B. 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 you are
using.

The following primitive data types are used:

- IDL Type
  - BASIC Type
  - Java Type
  - Description

- Int
- Short
- Long
- Short signed integer

- NameID CV Range
- Valid CV’s
- Functions
- Address Range
- Speed Range

- NMRA Compatible
- 0 None
- 1-8 1-8
- 9 1-127

- Extended
- 2 1-106
- 9 1-1023
- 14 14,28,128

- 49, 66-95
- 1-1023
- 14,28,128

- All Mobile
- 3 1-106
- 1-1023
- 14,28,128

- Name ID
- CV Range
- Valid CV’s
- Functions
- Address Range
- Speed Range

- Accuracy
- 1
- 513-593
- 8 0-511

- All Stationary
- 5 1-1024
- 9 513-1024
- 8 0-511

A long (DecoderObjectID) value is returned by the
KamDecoderPutAdd 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.

B. KamCVGetVar

<table>
<thead>
<tr>
<th>Parameter</th>
<th>NameID</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDecodeObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>CVRegist</td>
<td>1-1024</td>
<td>2</td>
<td>In</td>
<td>CV register</td>
</tr>
<tr>
<td>PCSVValue</td>
<td>int</td>
<td>3</td>
<td>Out</td>
<td>Pointer to CV value</td>
</tr>
<tr>
<td>Opaque object ID</td>
<td>1</td>
<td>Error flag</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C. KamCVPutValue

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

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

Return Value | NameID | Range | Description |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iError short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>Parameter List</td>
<td>Type</td>
<td>Range</td>
<td>Direction</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>DecoderObjectID long 1 In Decoder object ID</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVPutValue**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecoderObjectID long 1 In Decoder object ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVGetEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecoderObjectID long 1 In Decoder object ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVPutEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVGetEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVPutEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVGetEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVPutEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVGetEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVPutEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVGetEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVPutEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Maximal CV is 1024.** Maximum CV for this decoder is given by **KamCVGetMaxRegister**.
- **Return Value** Type Range Description

**KamCVGetEnable**

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICValue int 0-255 In CV value</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Opaque object ID handle returned by KamDecoderPutAdd.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case 3:06-cv-01905-JSW</td>
<td>Document 174-6</td>
<td>Filed 10/31/2007</td>
<td>Page 15 of 27</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
<td>-----------------</td>
<td>----------------</td>
<td></td>
</tr>
</tbody>
</table>

**US 6,530,329 B2**

---

### APPLICATION PROGRAMMING INTERFACE

**Parameter List:** Type Range Direction Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>KamProgramGetMode</td>
<td></td>
<td></td>
<td><strong>KamProgramGetMode</strong></td>
</tr>
<tr>
<td>1</td>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
</tr>
<tr>
<td>2</td>
<td>iProgLogPort</td>
<td>int</td>
<td>1-65535</td>
<td>2</td>
</tr>
</tbody>
</table>

- **piProgMode** int * 3 Out | Programming mode
- **UNKNOWN** int * 3 Out | Programmed mode

1. **iProgMode 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramGetMode takes the decoder object ID, logical programming port ID, and a pointer to store the programming mode as parameters. It sets the memory pointed to by **piProgMode** to the present programming mode.

2. **KamProgramGetStatus**
   - Given: **Parameter List:** Type Range Direction Description
     - **DecoderObjectID** long | 1 | In | Decoder object ID |
     - **ICVNum** long | 0-65535 | 4 | In | CV number |

- **piICVStatus** int * 3 Out | CV decoder programming status

1. **iProgMode 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramGetStatus takes the decoder object ID and a pointer to store the **iProgMode** programming status as parameters. It sets the memory pointed to by **piProgMode** to the present programming mode.

3. **KamProgramReadCV**
   - Given: **Parameter List:** Type Range Direction Description
     - **DecoderObjectID** long | 1 | In | Decoder object ID |
     - **ICVNum** long | 0-255 | 2 | In | CV number |

1. **iProgMode 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramReadCV takes the decoder object ID, configuration variable (CV) number as parameters. It reads the specified CV variable value to the server database.

4. **KamProgramCV**
   - Given: **Parameter List:** Type Range Direction Description
     - **DecoderObjectID** long | 1 | In | Decoder object ID |
     - **ICVNum** long | 0-65535 | 2 | In | CV number |

1. **iProgMode 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramCV takes the decoder object ID, configuration variable (CV) number, and a new CV value as parameters. It programs (writes) a single decoder CV using the specified value as source data.

---

**Parameter List:** Type Range Direction Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
</tr>
</tbody>
</table>

1. **iProgMode 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramReadDecoderToDataBase takes the decoder object ID as parameter. It reads all embedded CV values from the decoder and stores them in the server database.

2. **KamProgramDecoderFromDataBase**
   - Given: **Parameter List:** Type Range Direction Description
     - **DecoderObjectID** long | 1 | In | Decoder object ID |

1. **iProgMode 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramDecoderFromDataBase takes the decoder object ID as parameter. It programs (writes) all embedded decoder CV values using the server copy of the CV's as source data.

---

**Parameter List:** Type Range Direction Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
</tr>
</tbody>
</table>

1. **iProgMode 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramGetMaxModels takes the decoder object ID as parameter. It programs (writes) all embedded decoder CV values using the server copy of the CV's as source data.

---

**Parameter List:** Type Range Direction Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
</tr>
</tbody>
</table>

1. **iProgMode 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramDecoderFromDataBase takes the decoder object ID as parameter. It programs (writes) all embedded decoder CV values using the server copy of the CV's as source data.

---

**Parameter List:** Type Range Direction Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>iProgMode</td>
<td>int</td>
<td>1</td>
<td>Out</td>
</tr>
</tbody>
</table>

1. Normally i-65535, 0 on error.
   - **iProgMode 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramGetMaxModels takes the decoder object ID as parameter. It reads the memory pointed to by **iProgMode** to the maximum decoder type ID.

---

**Parameter List:** Type Range Direction Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>iModel</td>
<td>int</td>
<td>1</td>
<td>Out</td>
</tr>
</tbody>
</table>

1. Normally i-65535, 0 on error.
   - **iModel 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramGetModelName takes the decoder object ID as parameter. It reads the memory pointed to by **iModel** to the model name.

---

**Parameter List:** Type Range Direction Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>pbModelName</td>
<td>BSTR</td>
<td>2</td>
<td>Out</td>
</tr>
</tbody>
</table>

1. Maximum value for this server given by KamDecoderGetMaxModels.

2. Exact return type depends on language. It is Caching * for C++. Empty string on error.
   - **iModel 0** if success. **Nonzero** is an error number.
   - (see KamMiscGetErrorMsg)
   - KamProgramGetModelName takes the decoder object ID as parameter. It reads the memory pointed to by **pbModelName** to a BSTR containing the decoder name.
Case 3:06-cv-01905-JSW     Document 174-6      Filed 10/31/2007     Page 16 of 27

US 6,530,329 B2

---continued---

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

1. Maximum value for this server given by

KamDecoderGetMaxModels

2. Model dependent. 0 returned on error.

Return Value Type Range Description

Error short int Error flag

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

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

---continued---

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OldObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
<td>Old decoder object ID</td>
</tr>
<tr>
<td>NewAddress</td>
<td>int</td>
<td>2</td>
<td>In</td>
<td>New decoder address</td>
</tr>
<tr>
<td>pNewObjectID</td>
<td>long</td>
<td>*</td>
<td>Out</td>
<td>New decoder object ID</td>
</tr>
</tbody>
</table>

1. Opague object ID handle returned by

KamDecoderPutAdd.

2. 1-127 for short locomotive addresses, 1-12,531 for long locomotive decoders, 0-511 for accessory decoders.

Return Value Type Range Description

Error short int Error flag

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

KamDecoderGetErrorMsg takes an error number as a parameter and returns a decoder error message.

---continued---

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>LogicalPortID</td>
<td>int</td>
<td>1</td>
<td>In</td>
<td>Logical port ID</td>
</tr>
</tbody>
</table>

1. Opague object ID handle returned by

KamDecoderGetMaxLogPorts.

2. Maximum value for this server given by

KamDecoderGetMaxLogPorts

Return Value Type Range Description

Error short int Error flag

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

KamDecoderGetMaxLogPorts takes a decoder object ID and a logical port ID as parameters. It sets the memory pointed to by pLogicalPortID to the maximum logical port ID associated with iModel.

---continued---

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>pLogicalPortID</td>
<td>int</td>
<td>*</td>
<td>Out</td>
<td>Pointers to logical port ID</td>
</tr>
</tbody>
</table>

1. Opague object ID handle returned by

KamDecoderPutAdd.

2. Maximum value for this server given by

KamDecoderGetMaxLogPorts

---continued---

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderClass</td>
<td>int</td>
<td>1</td>
<td>In</td>
<td>Class of decoder</td>
</tr>
<tr>
<td>iObjCount</td>
<td>int</td>
<td>*</td>
<td>Out</td>
<td>Count of active decoders</td>
</tr>
</tbody>
</table>

1. - DECODER_ENGINE_TYPE,

2. - DECODER_SWITCH_TYPE,

3. - DECODER_SENSOR_TYPE,

Return Value Type Range Description

Error short int Error flag

1 Error = 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 already exists. KamDecoderCheckAddUse takes a decoder address, logical port, and decoder class as parameters. It returns zero if the address is not in use. It will return an error number if the call fails.

---continued---

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

1. Opague object ID handle returned by

KamDecoderPutAdd.

2. Maximum value for this server given by

KamDecoderGetMaxModels

---continued---

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iModel</td>
<td>int</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>LogicalPortID</td>
<td>int</td>
<td>*</td>
<td>Out</td>
<td>Logical port ID</td>
</tr>
</tbody>
</table>

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

KamDecoderGetModelFromObj takes a decoder object ID and a logical port ID as parameters. It sets the memory pointed to by pModelInst to the decoder type ID associated with the decoder object ID.

---continued---

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecoderObjectID</td>
<td>long</td>
<td>1</td>
<td>In</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>pCertifier</td>
<td>long</td>
<td>*</td>
<td>Out</td>
<td>Pointer to decoder facility mask</td>
</tr>
</tbody>
</table>

1. Opague object ID handle returned by

KamDecoderPutAdd.

2. 0 - CODEPRO_MODE_ADDR

1. CODEPRO_MODE_ADDR

2. CODEPRO_MODE_REG

3. CODEPRO_MODE_PAGE

4. CODEPRO_MODE_PWSHIFT

5. CODEPRO_MODE_PFLYNYG

6. Reserved

7. Reserved

8. Reserved

9. Reserved

10. Reserved

11. Reserved

12. Reserved

13. CODEPRO_MODEOfClass

14. CODEPRO_MODE_PWSHIFT

15. CODEPRO_MODE_PFLYNYG

16. CODEPRO_MODE_PFLYNYG

17. CODEPRO_MODE_PFLYNYG

18. CODEPRO_MODE_PFLYNYG

19. CODEPRO_MODE_PFLYNYG

20. CODEPRO_MODE_PFLYNYG

21. CODEPRO_MODE_PFLYNYG

Return Value Type Range Description

Error short int Error flag

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

KamDecoderGetModelFacility takes a decoder object ID and a logical port ID as parameters. It sets the memory pointed to by pCertifier to the decoder facility mask associated with the decoder object ID and logical port ID.

---continued---

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iDecoderClass</td>
<td>int</td>
<td>1</td>
<td>In</td>
<td>Class of decoder</td>
</tr>
<tr>
<td>iObjCount</td>
<td>int</td>
<td>*</td>
<td>Out</td>
<td>Count of active decoders</td>
</tr>
</tbody>
</table>

1. - DECODER_ENGINE_TYPE,

2. - DECODER_SWITCH_TYPE,
US 6,530,329 B2

APPENDIX PROGRAMMING INTERFACE

3 - DECODER_SENSOR_TYPE

Parameter List | Type | Range | Description
--- | --- | --- | ---
DecoderIObjectID | long | * | 3 Out Pointer to decoder object ID

- continued

APPENDIX PROGRAMMING INTERFACE

5 - Parameter List | Type | Range | Direction | Description
--- | --- | --- | --- | ---
DecoderIObjectID | long | 1 In | Out | 3 Out Pointer to decoder object ID

- continued
Case 3:06-cv-01905-JSW     Document 174-6      Filed 10/31/2007     Page 19 of 27

APPLICATION PROGRAMMING INTERFACE

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

KamEngGetFunctionName

Parameter List

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

Opaque object ID handle returned by KamDecoderPutAdd.

FL is 0. FL=FL is 1-6 respectively Maximum for this decoder is given by KamEngGetFunctionMax. Exact return type depends on language. It is Casting* for C++. Empty string on error.

Return Value

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

KamEngPutFunctionName

Parameter List

Type Range Direction Description
IDecoderObjectID long 1 In Decoder object ID
iFunctionID int 0-8 2 In Function ID number
pwsFunctionName BSTR 3 In Function name

Opaque object ID handle returned by KamDecoderPutAdd.

FL is 0. FL=FL is 1-6 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 pwsFunctionName.

KamEngGetConsistMax

Parameter List

Type Range Direction Description
IDecoderObjectID long 1 In Decoder object ID
iMaxConsist int * 2 Out Pointer to max consist number

Opaque object ID handle returned by KamDecoderPutAdd.

2 Command stations dependent.

Return Value

Type Range Description
iError short 1 Error flag
1 iError = 0 for success. Nonzero is an error number (see 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 iMaxConsist 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. 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 KmAssCrlFunction 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 KmCmdCommand command.

KamEngPutConsistParent

Parameter List

Type Range Direction Description
IDCCPairObjID long 1 In Parent decoder object ID

Opaque object ID handle returned by KamDecoderPutAdd.

2 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).
KamEngGetConsistParent takes the parent object ID and an alias address as parameters. It makes the decoder

KamEngPutConsistChild

Parameter List

Type Range Direction Description
IDCCParentObjID long 1 In Parent decoder object ID

Opaque object ID handle returned by KamDecoderPutAdd.

FL is 0. FL=FL is 1-6 respectively Maximum for this decoder is given by KamEngGetFunctionMax. Exact return type depends on language. It is Casting* for C++. Empty string on error.

Return Value

Type Range Description
iError short 1 Error flag
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 IDCCParentObjID to the consist identified by IDCCParentObjID. 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.

KamEngPutConsistRemoveObj

Parameter List

Type Range Direction Description
IDecoderObjectID long 1 In Decoder object ID

Opaque object ID handle returned by KamDecoderPutAdd.

FL is 0. FL=FL is 1-6 respectively Maximum for this decoder is given by KamEngGetFunctionMax.

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.

KamEngPutConsistRemoveObj takes the decoder object ID as a parameter. It removes the decoder specified by IDecoderObjectID 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 KmAssCrlFunction 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 KmCmdCommand command.

KamAssCrlFunction

Parameter List

Type Range Direction Description
IDecoderObjectID long 1 In Decoder object ID
iFunctionID int * 2 In Function ID number
pFunctionValue int * 3 Out Pointer to function value

Opaque object ID handle returned by KamDecoderPutAdd.

2 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).
KamAssCrlFunction 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 pFunctionValue to the specified function state.

KamAssCrlFunctionAll

Parameter List

Type Range Direction Description
IDecoderObjectID long 1 In Decoder object ID

Opaque object ID handle returned by KamDecoderPutAdd.

FL is 0. FL=FL is 1-6 respectively Maximum for this decoder is given by KamEngGetFunctionMax. Exact return type depends on language. It is Casting* for C++. Empty string on error.

Return Value

Type Range Description
iError short 1 Error flag
1 iError = 0 for success. Nonzero is an error number (see KamMiscGetErrorMsg).
KamAssCrlFunctionAll takes the decoder object ID and a BSTR as parameters. It sets the memory pointed to by pwsFunctionString to the symbolic name of the specified function.
<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetMaxObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunctionID</td>
<td>int</td>
<td>0-31</td>
<td>Function ID number</td>
</tr>
<tr>
<td>GetFunctionValue</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
</tbody>
</table>

**Note:** This command only changes the accessory database. The data is not sent to the decoder until execution of the OpenAccessoryCommand command.

### OpenAccessoryCommand

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetMaxObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunctionID</td>
<td>int</td>
<td>0-31</td>
<td>Function ID number</td>
</tr>
<tr>
<td>GetFunctionValue</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
<tr>
<td>GetObjectID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetID</td>
<td>int</td>
<td>0-31</td>
<td>Decoder object ID</td>
</tr>
<tr>
<td>GetFunction</td>
<td>int</td>
<td>0-31</td>
<td>Function value</td>
</tr>
</tbody>
</table>
Case 3:06-cv-01905-JSW     Document 174-6      Filed 10/31/2007     Page 21 of 27

US 6,530,329 B2

-continued

APPLICATION PROGRAMMING INTERFACE

(see KsmMiscGetErrorMsg).

KsmAccGetFeedback takes a decoder object ID, node name string, and function ID, as parameters. It returns

interesting in the function given by the function given by the node name string $\text{bsAccNode}$. $\text{bsAccNode}$ identifies the server application and method to call if the function changes state. Its format is "\{\text{Server}\}\.\text{App}\.\text{Method}\" where $\text{Server}$ is the server

name, $\text{App}$ is the application name, and $\text{Method}$ is the method name.

 Parameter List | Type | Range | Direction | Description
-----------------|------|-------|-----------|------------------
\text{ksmDecObjID} | long | 1-255 | In | Decoder object ID
\text{bsAccNode} | BSTR | 2-64 | In | Server node name

1. opaque object ID handle returned by KsmDecodePathAdd.

2. exact parameter type depends on language. It is

LPCSTR for C++.

3. maximum for this decoder is given by

KsmAccGetMaxDec.

4. error flag is given by

KsmAccGetErrorFlag.

5. interesting in the function given by the function given by the node name string $\text{bsAccNode}$. $\text{bsAccNode}$ identifies the server application and method to call if the function changes state. Its format is "\{\text{Server}\}\.\text{App}\.\text{Method}\" where $\text{Server}$ is the server

name, $\text{App}$ is the application name, and $\text{Method}$ is the method name.

 Parameter List | Type | Range | Direction | Description
-----------------|------|-------|-----------|------------------
\text{ksmDecObjID} | long | 1-255 | In | Decoder object ID
\text{bsAccNode} | BSTR | 2-64 | In | Server node name

1. opaque object ID handle returned by KsmDecodePathAdd.

2. exact parameter type depends on language. It is

LPCSTR for C++.

3. maximum for this decoder is given by

KsmAccGetMaxDec.

4. error flag is given by

KsmAccGetErrorFlag.

5. interesting in the function given by the function given by the node name string $\text{bsAccNode}$. $\text{bsAccNode}$ identifies the server application and method to call if the function changes state. Its format is "\{\text{Server}\}\.\text{App}\.\text{Method}\" where $\text{Server}$ is the server

name, $\text{App}$ is the application name, and $\text{Method}$ 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
APPLICATION PROGRAMMING INTERFACE

Case 3:06-cv-01905-JSW     Document 174-6      Filed 10/31/2007     Page 22 of 27
35

APPLICATION PROGRAMMING INTERFACE

iCommPortID int 1-65535 3 In Physical comm port ID

1 Maximum value for this server given by KamPortGetMaxLogPorts.
2 See FIFO, 6: Controller ID to controller name
mapping for values. Maximum value for this server is
given by KamMacMaxControllerID.
3 Maximum value for this server given by KamPortGetMaxPhysical.
Return Value Type Range Description
iError short 1 Error flag
1 iError = 0 for success. Nonzero is an error number
(see KamMacGetErrorMsg).
KamPortPutMapController takes a logical port ID, a
command station type ID, and a physical communications
port ID as parameters. It maps iLogicalPortID to
iCommPortID for the type of command station specified by
iControllerID.

(i)KamPortGetMaxLogPorts
Parameter List Type Range Direction Description
pMaxLogPorts int * 1 Out Maximum logical port ID

1 Normally 1-65535. 0 returned on error.
Return Value Type Range Description
iError short 1 Error flag
1 iError = 0 for success. Nonzero is an error number
(see KamMacGetErrorMsg).
KamPortGetMaxLogPorts takes a pointer to a logical port ID as a parameter. It sets the memory pointed to by
pMaxLogPorts to the maximum logical port ID.

(ii)KamPortGetMaxPhysical
Parameter List Type Range Direction Description
pMaxPhysical int * 1 Out Maximum physical port ID

pMaxSerial int * 1 Out Maximum serial port ID
pMaxParallel int * 1 Out Maximum parallel port ID

1 Normally 1-65535. 0 returned on error.
Return Value Type Range Description
iError short 1 Error flag
1 iError = 0 for success. Nonzero is an error number
(see KamMacGetErrorMsg).
KamPortGetMaxPhysical takes a pointer to the number of physical
ports, the number of serial ports, and the number of parallel ports as parameters. It sets the
memory pointed to by the parameters to the associated values
A. Commands that control command flow to the command station
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.

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

(iv) KamCmdDisconnect
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 KamMacGetErrorMsg).
KamCmdDisconnect takes a logical port ID as a parameter.
It disconnects the server to the specified command station.

5

APPLICATION PROGRAMMING INTERFACE

KamCmdCommand
Parameter List Type Range Direction Description
(iDecoderObjectID 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 KamMacGetErrorMsg).
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 cabo Dickie to a command station.

(i) KamCabCtrlMessage
Parameter List Type Range Direction Description
(iCabAddress int 1-65535 1 In Cab address

(b) msgSTR * 2 Out Cab message string
1 Maximum value is command station dependent.
2 Each return type depends on language. It is
Circling * 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 KamMacGetErrorMsg).
KamCabSetMessage takes a cab address and a pointer to a message string as parameters. It sets the memory pointed to by bMsg in the present cab message.

(ii) KamCabPutMessage
Parameter List Type Range Direction Description
(iCabAddress int 1 In Cab address

(b) msgSTR * 2 Out Cab message string
1 Maximum value is command station dependent.
2 Each 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 KamMacGetErrorMsg).
KamCabPutMessage takes a cab address and a BSTR as
parameters. It sets the cab message to bMsg.

(iv) KamCabGetAddr
Parameter List Type Range Direction Description
(iDecoderObjectID long 1 In Decoder object ID
(iCabAddress 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 Description
iError short 1 Error flag
1 iError = 0 for success. Nonzero is an error number
(see KamMacGetErrorMsg).
KamCabGetAddr takes a decoder object ID and a pointer
to a cab address as parameters. It sets the memory
pointed to by pCabAddress to the address of the cab
attached to the specified decoder.

(v) KamCabPutAddToCab
Parameter List Type Range Direction Description
(iDecoderObjectID 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 KamMacGetErrorMsg).
KamCabPutAddToCab takes a decoder object ID and cab
address as parameters. It attaches the decoder specified by
IDCnAddr to the cab specified by CabAddress.
A. Miscellaneous Commands
This section describes miscellaneous commands that
do not fit into the other categories.
37

APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Error</th>
<th>int</th>
<th>0-65535</th>
<th>1</th>
<th>Error flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>iError</td>
<td>0</td>
<td>for success. Nonzero indicates an error.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value</td>
<td>Type</td>
<td>Range</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter List</td>
<td>Type</td>
<td>Range</td>
<td>Direction</td>
<td>Description</td>
</tr>
<tr>
<td>iLastError</td>
<td>int</td>
<td>0-65535</td>
<td>1</td>
<td>Location field</td>
</tr>
<tr>
<td>fError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>iError</td>
<td>0</td>
<td>for success. Nonzero indicates an error.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value</td>
<td>Type</td>
<td>Range</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Parameter List</td>
<td>Type</td>
<td>Range</td>
<td>Direction</td>
<td>Description</td>
</tr>
<tr>
<td>iLastError</td>
<td>int</td>
<td>0-65535</td>
<td>1</td>
<td>Location field</td>
</tr>
<tr>
<td>fError</td>
<td>short</td>
<td>1</td>
<td>Error flag</td>
<td></td>
</tr>
<tr>
<td>iError</td>
<td>0</td>
<td>for success. Nonzero indicates an error.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Case 3:06-cv-01905-JSW     Document 174-6     Filed 10/31/2007     Page 24 of 27

38

APPLICATION PROGRAMMING INTERFACE

<table>
<thead>
<tr>
<th>Parameter List</th>
<th>Type</th>
<th>Range</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iControllerID</td>
<td>int</td>
<td>1-65535</td>
<td>1</td>
<td>Command station type ID</td>
</tr>
<tr>
<td>iLogicalPortID</td>
<td>int</td>
<td>1-65535</td>
<td>2</td>
<td>Logical port ID</td>
</tr>
<tr>
<td>iModuleName</td>
<td>BSTR</td>
<td>2</td>
<td>Out</td>
<td>Command station type name</td>
</tr>
<tr>
<td>iModuleName</td>
<td>BSTR</td>
<td>1</td>
<td>Command station type name</td>
<td></td>
</tr>
</tbody>
</table>

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

2. Exact return type depends on language. It is iControllerID for C++, Empty string on error. Return Value Type Range Description
   Parameter List | Type | Range | Direction | Description |
   iLogicalPortID | int | 1-65535 | 2 | Logical port ID |
   iModuleName | BSTR | 2 | Out | Command station type name |

3. Maximum value for this server given by iModuleName.

4. Exactly return type depends on language. It is iModuleName for C++, Empty string on error. Return Value Type Range Description
   Parameter List | Type | Range | Direction | Description |
   iLogicalPortID | int | 1-65535 | 2 | Logical port ID |
   iModuleName | BSTR | 2 | Out | Command station type name |

5. iControllerID for C++, iModuleName for C++. Return Value Type Range Description
   Parameter List | Type | Range | Direction | Description |
   iLogicalPortID | int | 1-65535 | 2 | Logical port ID |
   iModuleName | BSTR | 2 | Out | Command station type name |
The terms and expressions which have been employed in the foregoing specification are used therein as terms of description and not of limitation, and there is no intention, in the use of such terms and expressions, of excluding equivalents of the features shown and described or portions thereof, it being recognized that the scope of the invention is defined and limited only by the claims which follow.

What is claimed is:

1. A method of operating a digitally controlled model railroad comprising the steps of:
   (a) transmitting a first command from a first program to an interface;
   (b) transmitting a second command from a second program to said interface; 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 digital command stations.
15. The method of claim 10 wherein said interface communicates in an asynchronous manner with said first program while communicating in a synchronous manner with said plurality of digital command stations.
16. A method of operating a digitally controlled model railroad comprising the steps of:
   (a) transmitting a first command from a first program to an interface;
   (b) transmitting a second command from a second program to said interface; and
   (c) said interface selectively sending a third and fourth command representative of said first command and said second command, respectively, to the same digital command station.
17. The method of claim 16 wherein said interface communicates in an asynchronous manner with said first and second programs while communicating in a synchronous manner with said digital command station.
18. The method of claim 16, further comprising the step of providing an acknowledgment to said first program in response to receiving said first command by said interface prior to sending said third command to said digital command station.
19. A method of operating a digitally controlled model railroad comprising the steps of:
   (a) transmitting a first command from a first program to a first processor; and
   (b) said first processor providing an acknowledgment to said first program indicating that said first command has properly executed prior to execution of commands related to said first command by said digitally controlled model railroad.
20. The method of claim 19, further comprising the step of sending said first command to a second processor which processes said first command into a state suitable for a digital command station.
21. The method of claim 19, further comprising the steps of:
   (a) transmitting a second command from a second program to said first processor; and
   (b) said first processor selectively providing an acknowledgment to said second program indicating that said second command has properly executed prior to execution of commands related to said second command by said digitally controlled model railroad.
22. The method of claim 21, further comprising the steps of:
   (a) sending a third command representative of said first command to one of a plurality of digital command stations based upon information contained within at least one of said first and third commands; and
   (b) sending a fourth command representative of said second command to one of said plurality of digital command stations based upon information contained within at least one of said second and fourth commands.
23. A method of operating a digitally controlled model railroad comprising the steps of:
   (a) transmitting a first command from a first program to an asynchronous command processor;
   (b) said asynchronous command processor providing an acknowledgment to said first program indicating that said first command has properly executed prior to execution of said first command by said digitally controlled model railroad;
   (c) sending said first command to a command queue where said asynchronous command processor considers the intended destination device of said first command; and
   (d) processing said first command by said 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.

* * * * *
It is certified that an 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
Director of the United States Patent and Trademark Office