

# Accessory Programming with Visual Basic

**Matt Katzer**  
KAM Industries  
Portland, Or.



# Clinics sponsored by KAM

- **Computer Interface Application Programming for DCC**  
– 8:30 AM Monday, Governors 1
- **Accessory Programming with Visual Basic and Java**  
– 10:00 AM Monday, Governors 1
- **Prototype CTC dispatching with Train Driver Professional TDPro 32**  
– 4:00 PM Friday, Governors



# Agenda

- **Train Server® interface architecture**
  - KAM feedback implementation
  - Execution model
- **Accessory control**
  - Logical devices connection
    - » Command station, decoder model
  - Accessory output
  - Accessory feedback (state change)
- **Questions/Answers**
  - Software supplied by KAM

Clinic Accessory tutorial is in the convention handbook  
slides will be posted on KAMs website  
<http://www.kamind.com> by 7/23/99



# Why are you here

- **Clinic will focus on the following..**
  - Write a VB application to control accessories
  - Implement a monitor application to read the exact switch state
- **Complete article on material is contained in your clinic handbook.**
  - Tutorial manual will be available on Sept 15.
  - Examples in VB, VC++ and MS Java
  - Source code is on the CD-ROM
- **Clinic will show working examples**
  - Engine Commander® switch control
  - Computer Dispatcher® Lite switch control
  - implementation example programs in VB
- **What are your expectations?**

# Accessory Programming 1-2-3

- 1. Connect to the command station**
  - Select the command station to connect to
  - Define a logical port as a reference
- 2. Register the decoder object**
  - Define a master decoder as a model
  - Create a reference object
- 3. Send commands to the decoder**
  - Turn the state on/off
  - Send the commandor
  - process feedback

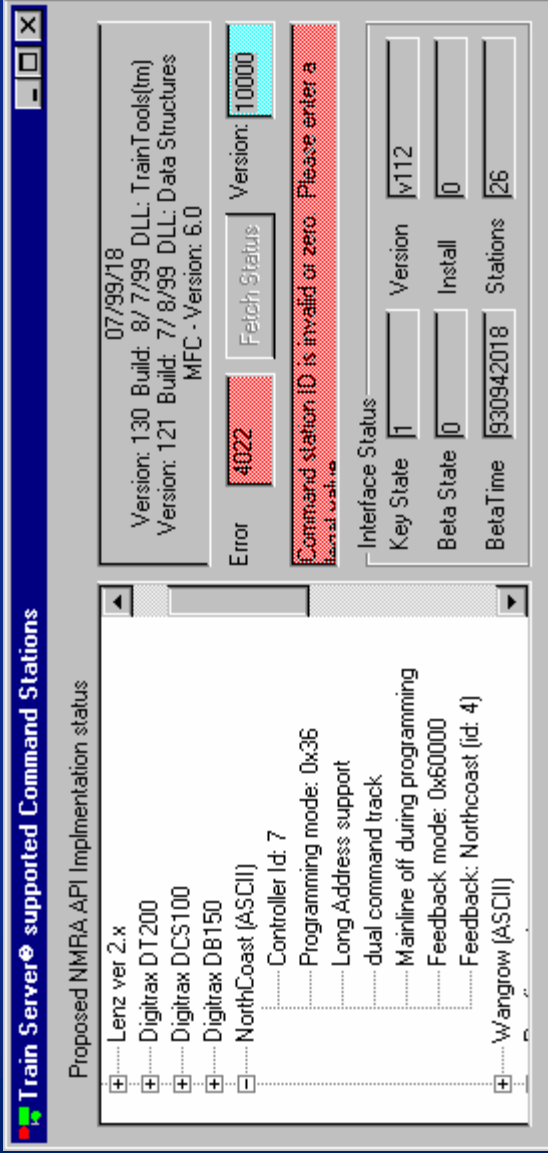


# Before we begin..

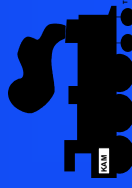
- **A few API concepts**
  - All commands are sent to logical devices
  - The computer is much faster than the interface it is connected to
  - Information is communicated to objects
- **This is nice but what is....**
  - A logical device?
  - An object?
  - Do I care?



# Demo

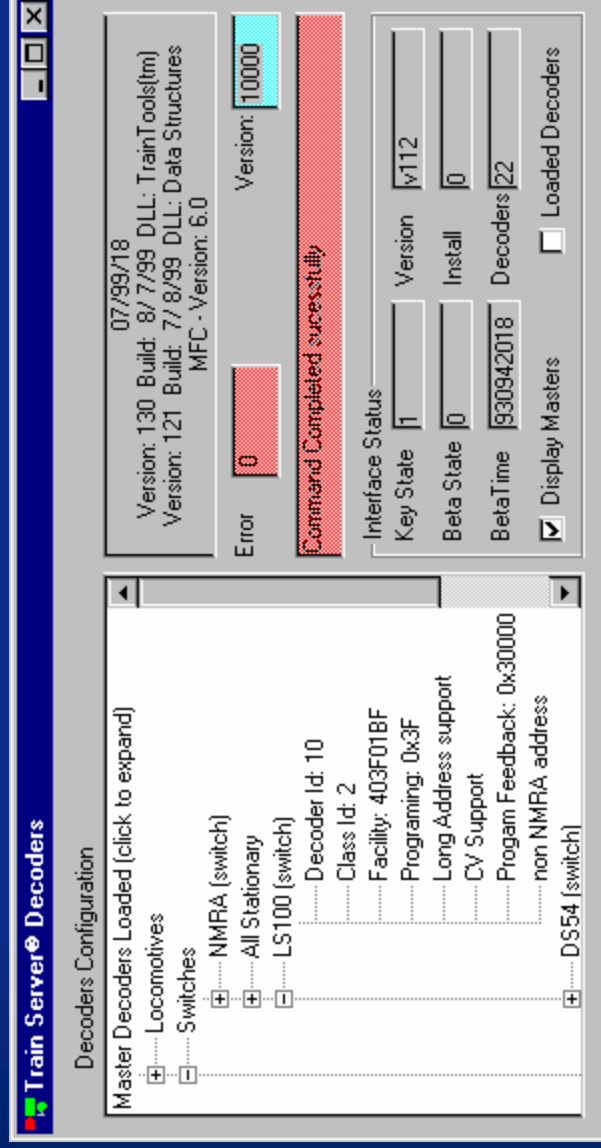
- Command station support
- 
- The screenshot shows the 'Train Server supported Command Stations' window. The main pane lists supported stations with expandable details:
- Lenz ver 2.x:
    - Digitrax DT200
    - Digitrax DCS100
    - Digitrax DB150
    - NorthCoast (ASCII)
      - Controller Id: 7
      - Programming mode: 0x36
      - Long Address support
      - dual command track
      - Mainline off during programming
      - Feedback mode: 0x60000
      - Feedback: Northcoast (id: 4)
    - Wangrow (ASCII)
- Metadata and error information:
- Date: 07/99/18
  - Version: 130 Build: 8/7/99 DLL: TrainTools(tm)
  - Version: 121 Build: 7/8/99 DLL: Data Structures MFC - Version: 6.0
  - Error: 4022
  - Fetch Status: Version: 10000
  - Message: Command station ID is invalid or zero. Please enter a valid value.
- Interface Status:
- Key State: 1
  - Beta State: 0
  - BetaTime: 930942018
  - Version: v112
  - Install: 0
  - Stations: 26

- Train Server® Version 2.121 supports 26 command station types



# Demo

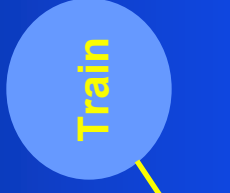
- Decoder Support and logical devices



- Master models support are 22
  - Database may be updated from the web

# API Architecture(review)

- **API is a combination of a property/method model; with an execution framework**
  - Objects are not passed in the API; rather states are passed
  - The state model reduces overhead on clients and improves the ability to port the API to different architecture (marshalling is expensive in software)
  - States are set; and execution is passed
    - » DccEngSetFunction(.....)      **Set something(,,)**
    - » DccEngGetSpeed(...)            **Get something(,,)**
    - » DccCommand(ObjectId)



- **The API was designed to support prototype and non prototype operations**

# Our Goal...

- **To write an application that we can use with all command stations**
  - Must be independent of communications ports
  - Must support mix mode operation
  - Must be easy to use
- **How do we do this...**
  - Use the API!
  - Use logical devices!
  - Allow the interface to do the configuration for the command station
  - Abstraction!



# Agenda

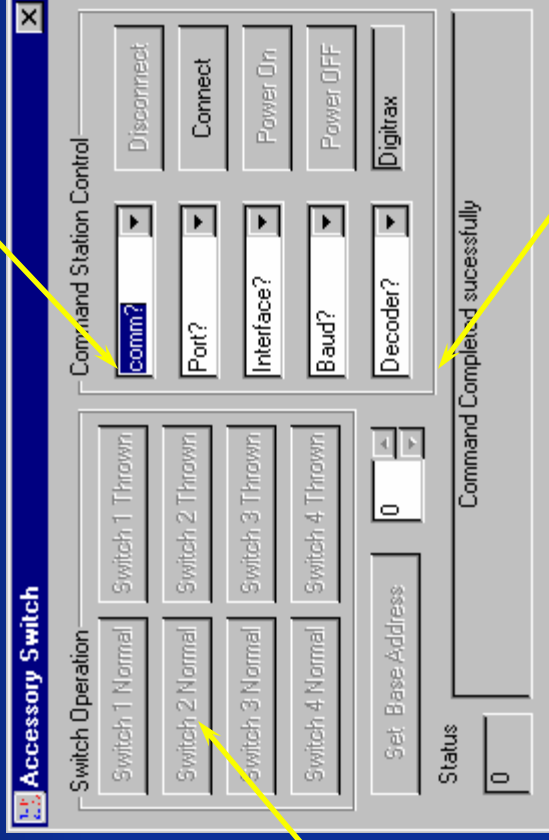
- **Train Server® interface architecture**
  - KAM feedback implementation
  - Execution model
- **Accessory control**
  - Logical devices connection
    - » Command station, decoder model
  - Accessory output
  - Accessory feedback (state change)
- **Questions/Answers**
  - Software supplied by KAM



# Demo

- Accessory Control

1. Define the command station



3. Send the command

- Remember 1 – 2 – 3

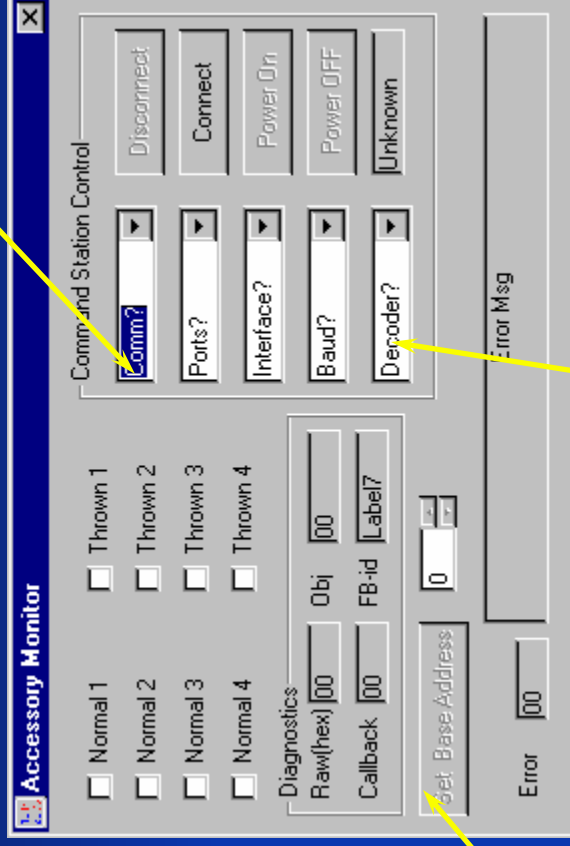
2. Define the decoder



# Demo

- Accessory Monitor

1. Define the command station



3. Send the command

• Remember 1 – 2 – 3

2. Define the decoder



# 1 - 2 - 3

## 1. Connect to the command station

```

AccessorySwitch.frm - Notepad
File Edit Search Help
iError = Engcmd.DccPortSetConfig(iLogicalPort, 0, iPortRetrans, 0) ' setting PORT_RETRANS
iError = Engcmd.DccPortSetConfig(iLogicalPort, 1, iPortRate, 0) ' setting PORT_RATE
iError = Engcmd.DccPortSetConfig(iLogicalPort, 2, iPortParity, 0) ' setting PORT_PARITY
iError = Engcmd.DccPortSetConfig(iLogicalPort, 3, iPortStop, 0) ' setting PORT_STOP
iError = Engcmd.DccPortSetConfig(iLogicalPort, 4, iPortWatchdog, 0) ' setting PORT_WATCH
iError = Engcmd.DccPortSetConfig(iLogicalPort, 5, iPortFlow, 0) ' setting PORT_FLOW
iError = Engcmd.DccPortSetConfig(iLogicalPort, 6, iPortData, 0) ' setting PORT_DATABITS
iError = Engcmd.DccPortSetConfig(iLogicalPort, 7, iDebugMode, 10809) ' setting PORT_DEBUG

' Now map the Logical Port, Physical device, Command station and Controller
iError = Engcmd.DccPortSetMapController(iLogicalPort, iController, iComPort)
iError = Engcmd.DccCmdConnect(iLogicalPort)
iError = Engcmd.DccOpnTurnOnStation(iLogicalPort)
If (iError) Then
    SetButtonState (False)
Else
    SetButtonState (True)
    iError = Engcmd.DccMiscGetControllerName(iController, strCntrl)
    SetError (iError)
    Controller.Caption = strCntrl
End If

```



## 1 - 2 - 3

## 2. Register the decoder

```

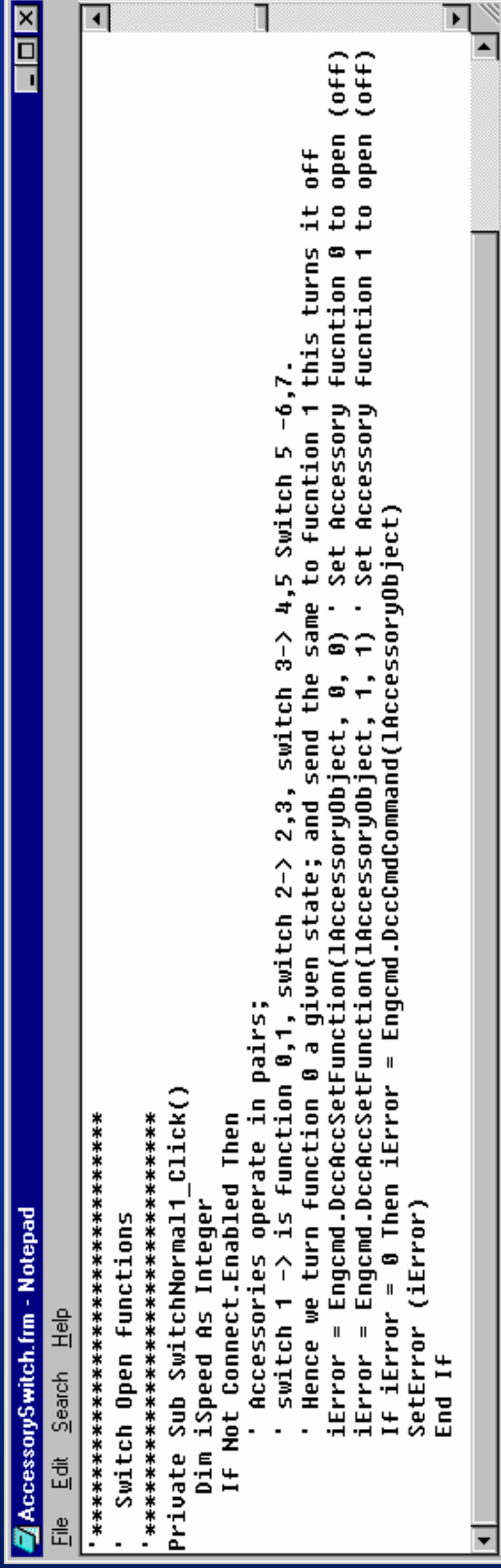
AccessorySwitch.frm - Notepad
File Edit Search Help
'*****
'Set the address button
'*****
Private Sub DCCAddr_Click()
    Dim lClass As Long, lFacility As Long, lFeedback As Long
    iDecoderType = Decoder.ListIndex + 1 ' Offset by six quick fix
    iError = Engcmd.DccDecoderGetModelFacility(iDecoderType, lFacility, lFeedback, lClass)
    If (lClass = 1) Then
        Call EngineDisplay
    Else
        Call AccessoryDisplay
    End If
'
' Once we make a connection, we use the lAccessoryObject as the reference object to
' send control information
If (Address.Text > 0) Then
    iStatus = Engcmd.DccDecoderAddAddr(Address.Text, 0, iLogicalPort, 0, iDecoderType)
    SetError (iStatus)
    If (lAccessoryObject) Then
        SetButtonState (True)
    Else
        MsgBox ("Address not set, check error message")
        SetButtonState (False)
    End If
End If

```



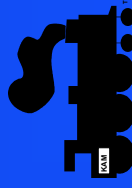
# 1-2-3

## 3. Send the command



```
File Edit Search Help
'*****
' Switch Open functions
'*****
Private Sub SwitchNormal1_Click()
    Dim iSpeed As Integer
    If Not Connect.Enabled Then
        ' Accessories operate in pairs;
        ' switch 1 -> is function 0,1, switch 2-> 2,3, switch 3-> 4,5 Switch 5 -6,7.
        ' Hence we turn function 0 a given state; and send the same to function 1 this turns it off
        iError = Engcmd.DccAccSetFunction(lAccessary0bject, 0, 0) ' Set Accessory function 0 to open (off)
        iError = Engcmd.DccAccSetFunction(lAccessary0bject, 1, 1) ' Set Accessory function 1 to open (off)
        If iError = 0 Then iError = Engcmd.DccCmd6ommand(lAccessary0bject)
        SetError (iError)
    End If
End Sub
```

Remember, switches are paired devices



# Train Server® Review

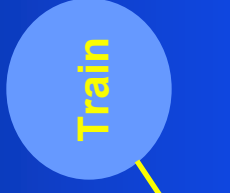
- The VB program turned the switch on /off
- Train Server® supports exact feedback,
  - state of switch is output to the logical port
  - Data accuracy is dependent on Command Station mfg.
- Return state information is based on switch command being sent.
  - States are set; and execution is passed

» DccCmdCommand(.....)

Do something(,;)

Tell us what happen

- Lets look at the monitor application to see what happened.



# 1 - 2 - 3

## 1. Connect to the command station

```

FeedbackMonitor.frm - Notepad
File Edit Search Help
iError = Engcmd.DccPortSetConfig(iLogicalPort, 0, iPortRetrans, 0) ' setting PORT_RETRANS
iError = Engcmd.DccPortSetConfig(iLogicalPort, 1, iPortRate, 0) ' setting PORT_RATE
iError = Engcmd.DccPortSetConfig(iLogicalPort, 2, iPortParity, 0) ' setting PORT_PARITY
iError = Engcmd.DccPortSetConfig(iLogicalPort, 3, iPortStop, 0) ' setting PORT_STOP
iError = Engcmd.DccPortSetConfig(iLogicalPort, 4, iPortWatchdog, 0) ' setting PORT_WATCHDOG
iError = Engcmd.DccPortSetConfig(iLogicalPort, 5, iPortFlow, 0) ' setting PORT_FLOW
iError = Engcmd.DccPortSetConfig(iLogicalPort, 6, iPortData, 0) ' setting PORT_DATABITS
iError = Engcmd.DccPortSetConfig(iLogicalPort, 7, iDebugMode, 10801) ' setting PORT_DEBUG

' Now map the Logical Port, Physical device, Command station and Controller
iError = Engcmd.DccPortSetMapController(iLogicalPort, iController, iComPort)
iError = Engcmd.DccCmdConnect(iLogicalPort)
iError = Engcmd.Dcc0prTurnOnStation(iLogicalPort)
If (iError) Then
    SetButtonState (False)
Else
    SetButtonState (True)
iError = Engcmd.DccMiscGetControllerName(iController, strCtrl)
SetError (iError)

```

## 1 - 2 - 3

## 2. Register the decoder, and the feedback event

```

FeedbackMonitor.frm - Notepad
File Edit Search Help

iError = Engcmd.DccDecoderGetModelFacility(iDecoderType, IFacility, IFeedback, IClass)

    If (IClass = 2) Then
        Call AccessoryDisplay
    Else
        If (IClass = 3) Then
            Call SensorDisplay
        Else
            Call EngineDisplay
        End If
    End If

: Once we make a connection, we use the IAccessoryObject as the reference object to
: send control information
If (Address.Text > 0) Then
    iStatus = Engcmd.DccDecoderAddAddr(Address.Text, 0, iLogicalPort, iLogicalPort, 0, iD
SetError (iStatus)
If (IAccessoryObject) Then
    SetButtonState (True)

' Kill the old object reference if it was set...
IF (IObjRef) Then
    iError = objFeedback.DccFeedbackAccessoryAll(IAccessoryObject, IObjRef)
    SetError (iError)

```



# 1-2-3

## 3. Receive the information

```


FeedbackMonitor.frm - Notepad
File Edit Search Help
////////////////////////////////////
//
// Mapping definitions
// Engine -> DccEngineReponse
// Address: bit 0 - 15 - 65K address
// Functions: bit 16 - 22 - F0 & F1 - F6
// Direction: bit 23 - 1-> forward, 0 Reverse
// Speed: bit 25 - 31 - 0 - 256
// Accessory -> DccResponseAccessoryAll
// Functions: bit 0 - 31 - F1 -> bit 0, F32 -> bit 31
//
//
// RESPONSE_ENGINE_ADDR = 0x0000FFFF; // Address mask
// RESPONSE_ENGINE_FUNCTIONS = 0x007F0000; // Function mask
// RESPONSE_ENGINE_DIRECTION = 0x00800000; // Direction mask
// RESPONSE_ENGINE_SPEED = 0xFF000000; // Speed mask
//
// *****
// Private Sub objFeedback_DccResponseAccessoryAll(ByVal lDccObject As Long, ByVal lFuncVal As Long)
//
// Dim lResults0 As Long, lResults1 As Long, lResults2 As Long, lResults3 As Long
// Dim lResults4 As Long, lResults5 As Long, lResults6 As Long, lResults7 As Long

```

And update the display...



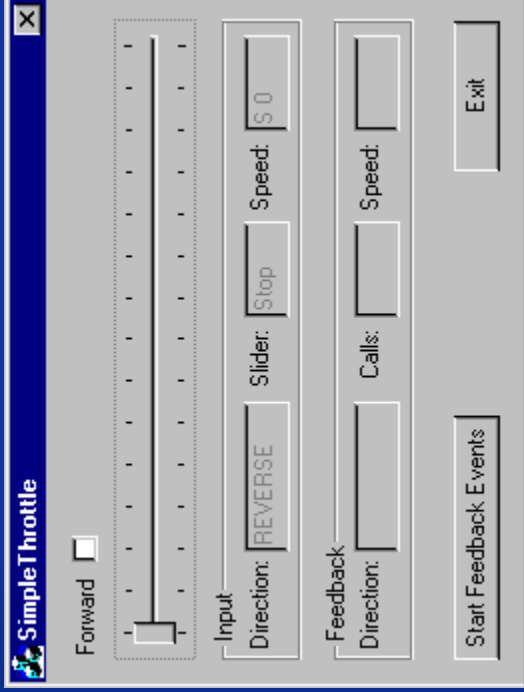
# Accessory Control

- It is very easy to do....
  - The code is identical in VB, Java and C++
  - This the power of the API.
- Do something(,,)
- Tell us what happen
- KAMs model is a full feedback model,
    - This is unique and novel in its implementation
    - Train Server® supports Asynchronous feedback across the net
- 



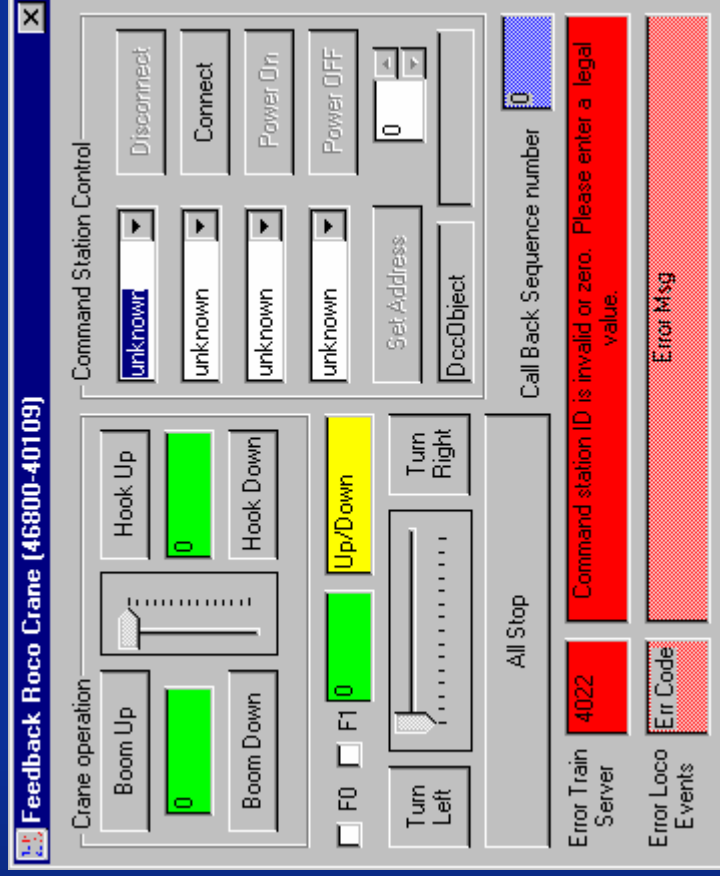
# Demo: Feedback Throttle

- Throttle devices work the same



# Demo: Feedback Crane

- Roco Crane supports feedback



# Software Supplied by KAM

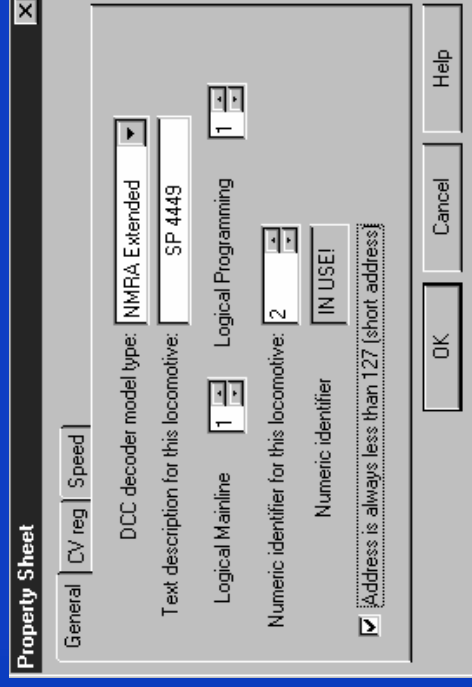
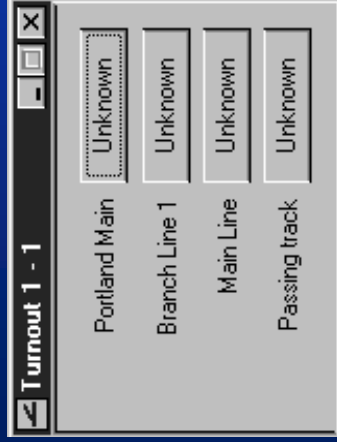
- **Engine Commander®**
  - Manual control
- **Computer Dispatcher® Lite**
  - Non prototype operation
- **Computer Dispatcher® Pro**
  - Prototype operation
  - Based on Train Tracks TdPro 32 software
  - In production use at BNSF, UP, NS and GCT

**All of KAM products use Event Feedback!**



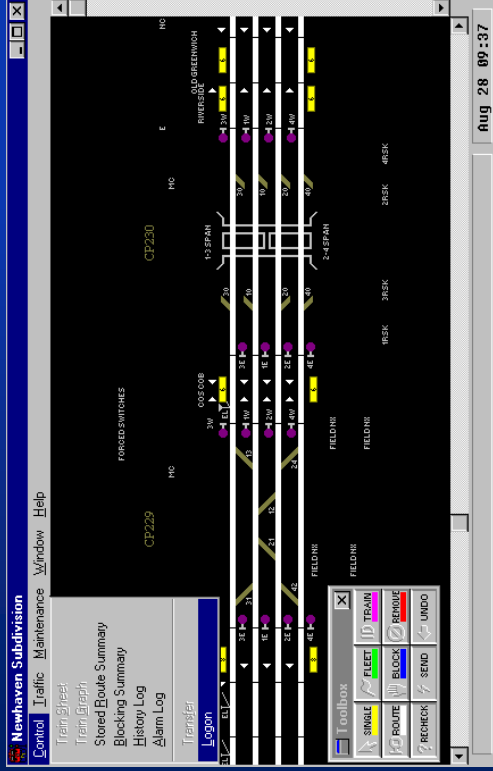
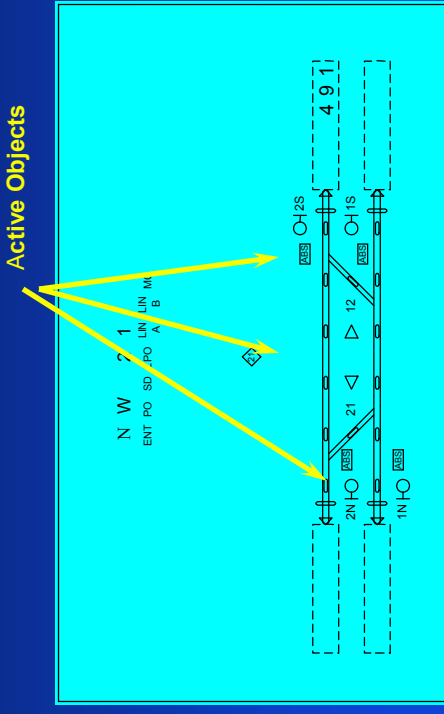
# Engine Commander®

- **Built on a modular philosophy**
  - Implements all of the API's
  - Simple interface, but uses abstraction to reduce complexity of task
  - An accessory through switches..
  - A throttle run trains..
  - Sensors show state



# Computer Dispatcher® Pro1

- KAM needed to build an infrastructure so we could support prototype operation....



**Computer Dispatcher®**  
**model view of an active element**  
**with full Entry/Exit (route) control**

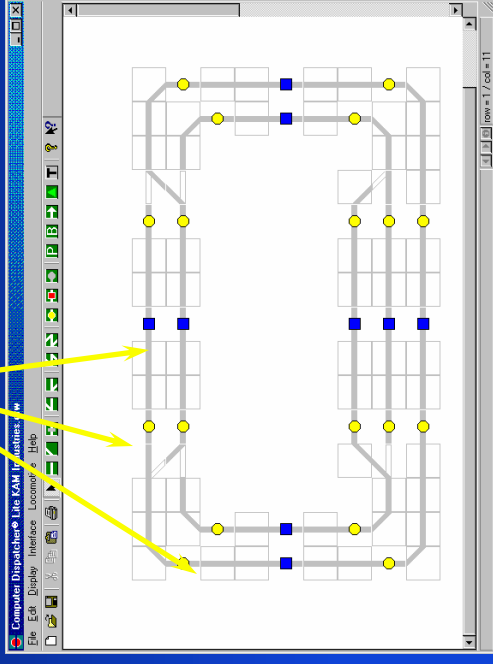
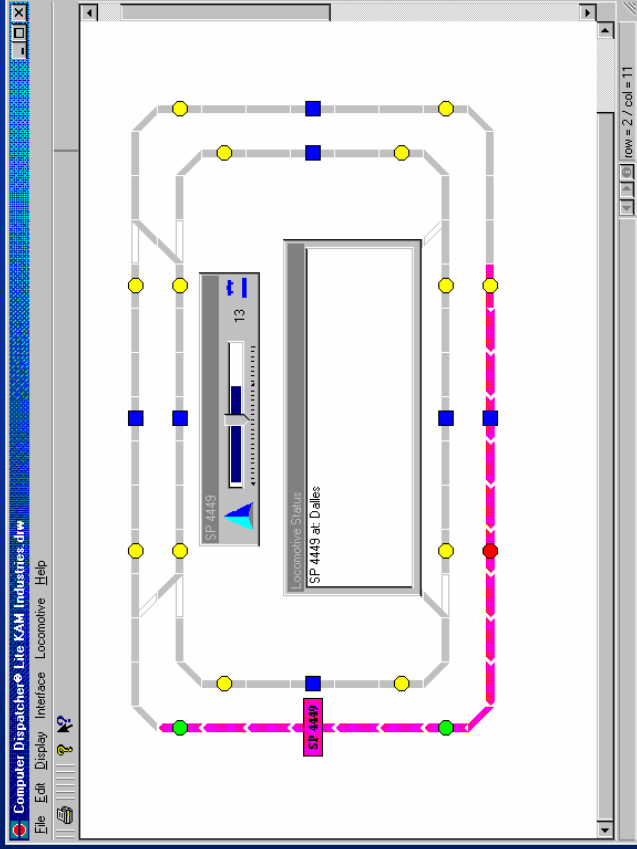
**CTC Panel view in Computer Dispatcher®**



# Computer Dispatcher® Lite

- Development of a non prototype application for the consumer....

Switches and sensors

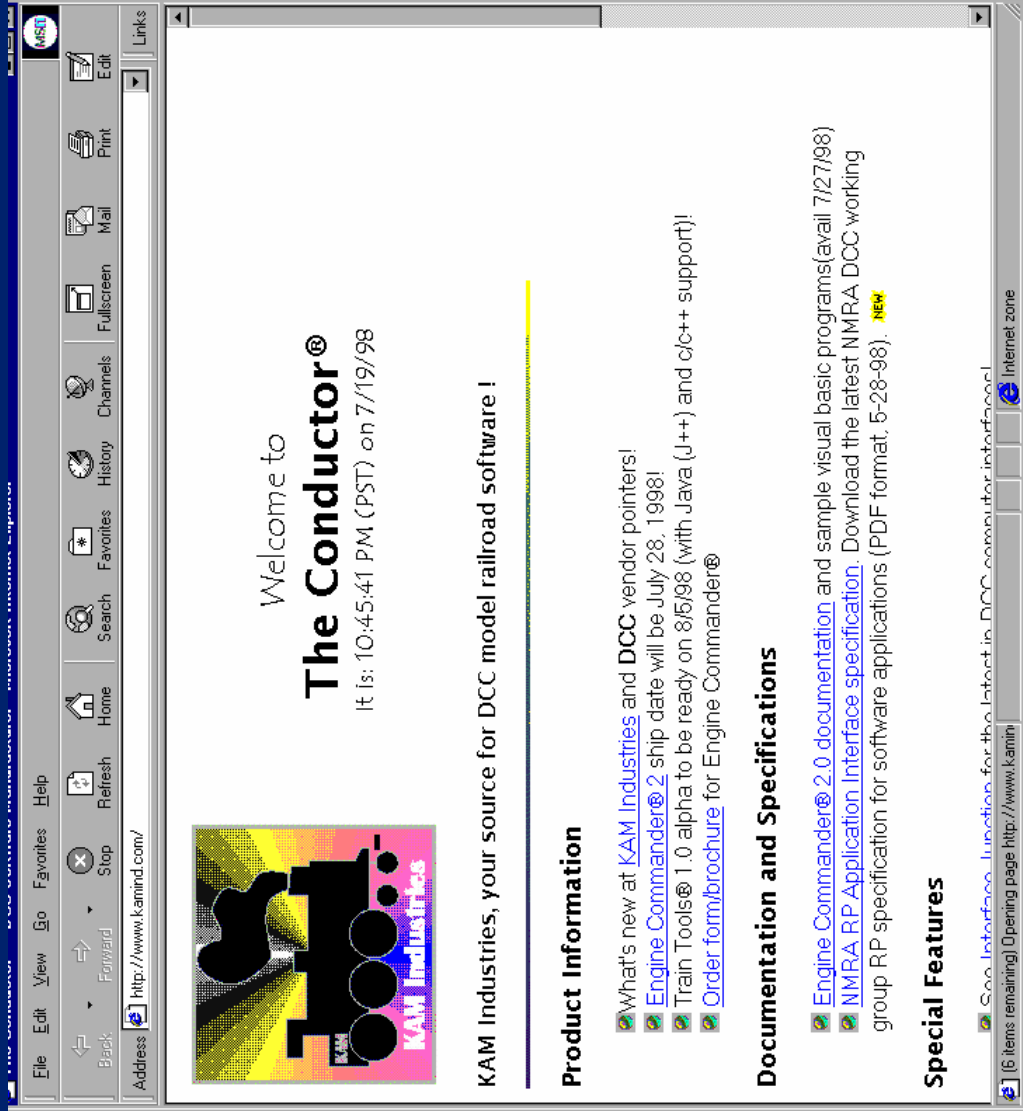


CTC panel view in Computer Dispatcher® Lite  
 with automatic block operation

Computer Dispatcher® Lite  
 active elements



http://www.kamind.com



# Questions ?

**Matt Katzer**  
**email: [mkatzer@kamind.com](mailto:mkatzer@kamind.com)**  
**web: <http://kamind.com>**  
**home: 503-291-1221**

**Computer Dispatcher®**, **Engine Commander®**, **The Conductor®**, **Train Server®**, **kamind®**,  
and **Train Tools®** are registered trademarks of KAM Industries.  
KAM Industries is a division of KAMIND associates, Inc.



# Appendix

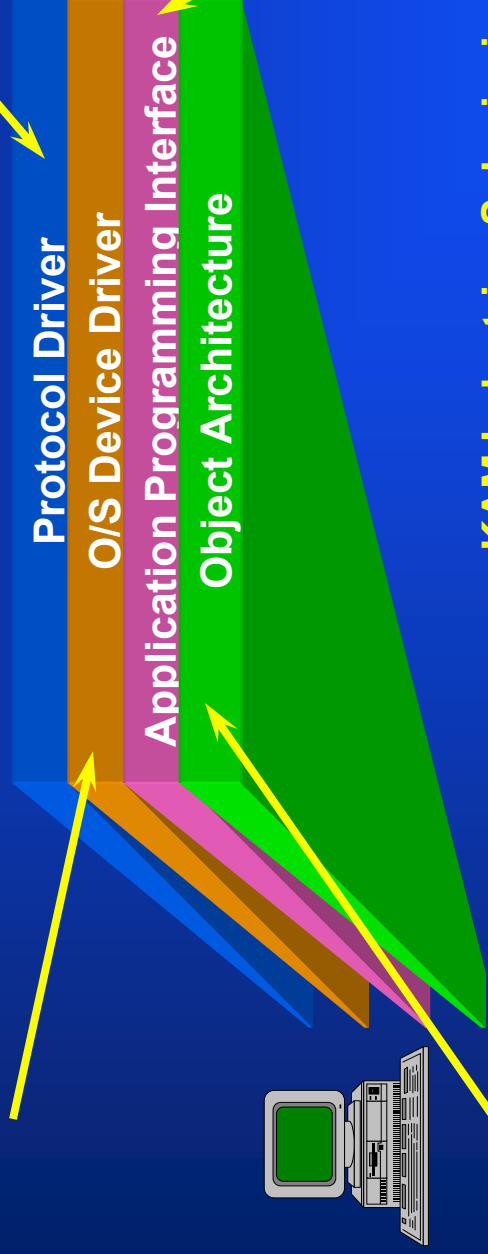
- **NMRA application model**
- **API Architecture**
- **API summary (APIStatus.exe)**

# NMRA Application S/W Architecture Model

- There are four parts to the NMRA DCC software architecture model

Katzer/Rice Draft Protocol Specification (7/98)

no activity



KAM Industries Submission (7/98)  
specification located on [Http://www.kamind.com](http://www.kamind.com)

Rosa Proposal by Tannersoft (7/97)



# Architecture (cont.)

- **API is built on the following concepts**
  - **Devices are logical devices. There is a mapping between logical to physical**
    - » DccPortGetMaxLogPorts(IMaxLogical)
    - » PortGetMaxPhysical(IMaxPhysical, IMaxSerial, IMaxParallel)
    - » DccPortGetName(iComPort, strComPort)
    - » DccMiscGetControllerName(iController, strCntrl)
    - » DccPortSetConfig(iLogicalPort, 0, iPortRetrans, 0)
    - » DccPortSetMapController(iLogicalPort, iController, iPhysicalPort)
  - **Abstraction for the client was the key.**
    - » Client does not need configuration ability
    - » Client only needs to know how map a logical to a physical device
    - » The configuration extension was added to accommodate new manufactures equipment using a standard driver.



# Architecture (DCC Object)

- **DCC addresses are integrated in an object**
  - Objects have a reference and can be translated
  - The object must be complete enough to use the API with as little information as possible
  - Hence all information to control accessories or locomotives require and object as a reference
    - » This allows developers to implement the sever as an object store independent of the Operating System architecture.
    - » The objects then become a “DccCookie” .
      - DccCookie encapsulate programming ports, command ports, decoder class and DCC address
- The DCC Cookie becomes the reference token for system calls and can easily be validated



# Architecture (cont.)

- **Abstraction also extends to decoders**
  - we needed a model that allowed flexibility and growth
    - » Decoder classes were created to group decoders.
    - » Each decoder class supports multiple decoder models
      - Classes are “Loco”, “Switch”, “Sensor”
      - Models are DH84, K87, LS110, Chub Detector<sup>1</sup>
  - » A set of decoder management functions were added to support application development
    - DccDecoderGetMaxModels(...)
    - DccDecoderGetModelName(...)
    - DccDecoderGetMaxAddress(...)
    - DccDecoderGetMfgName(...)
    - DccDecoderGetPowerMode(...)
    - DccDecoderGetModelFacility(...)
    - DccDecoderSetModelToObject( ...)
- **Objective was abstraction of the Interface**

# API command summary

- **API Command classes**

- CV
- Engine
- Consist
- Accessory
- Command
- Programming
- Communications
- Command
- Decoder
- Cab
- Feedback
- Callback methods

These are the major classes of commands needed in most DCC software applications.

We have implemented **Engine Commander®** and **Computer Dispatcher® Pro/Lite**



# Demo

- API supported in KAMs Train Server®

