

Computer Interface Application Programming for DCC

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

- History
- Train Server® Interface architecture
 - Key concepts and terms
 - Execution model
- How the API works
 - Logical devices
 - Decoder structure
 - Command structure
- API summary
- Using the API in Visual Basic/MS Java or C/C++
- Questions/Answers



Clinic API material 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 review the proposed API architecture**
- **Complete article on material is contained in the '99 clinic handbook.**
 - Tutorial manual will be available on Sept 15 via web.
- **Clinic will focus on API architecture**
 - we will talk application programming
 - programming languages
 - implementation example programs (C++ and VB)
- **What are your expectations?**



History of the API...

- **Train Server® API was developed to address an internal need at KAM**
 - **KAM needed a way to control software costs and improve schedules**
 - » non standard computer interfaces by command stations are costly to support
 - » every one had their own architecture
 - **Standardization was needed to address product development**
 - » API was required so KAM could decouple the GUI (client) from the backend (server) application
 - » Needed to implement a Internet backbone
 - » Needed a way to support Windows 95/98 and NT 4.0 and Windows 2000 in a distributed architecture
 - » Needed an standard interface for a family of software products



History (cont.)

- **KAM needed an API that was language friendly**
 - Need flexibility to implement Java, and Java RMI if required
 - Visual Basic/Visual Basic for Applications (Excel, Word access..)
 - Needed support for C/C++
 - Needed support for our web servers via distributed Common Object Model (DCOM) and the Internet
- **Our next generation software Computer Dispatcher® was an object driven model which required integrated network support.**
 - We needed an API that delivered functionality and implementation performance.
 - The API had to support COM and CORBA standards
 - The API had to have source level compatibility at the minimum
 - The API had to support marshalling across different Operating Systems
 - Required prototype and non prototype operation



History of the API

- **KAM Industries submitted the API in May 7, 98 to the NMRA DCC Working group for RP approval**
 - KAM updated submission to released version 2.121 of the API
 - Engine Commander®, Computer Dispatcher® Pro, Computer Dispatcher® Lite use version 2.121
 - 17 Visual basic applets in production (on CD-ROM)
 - 1- VC++/and 1- Java applets in production (on CD-ROM)
 - 5 new applications under development (turntable, switch logic processor, pick your flavor of a Roco crane, timetable import, and the Train Server® programmer (KAMs version of the PR1)
- **All of KAMs products include samples and tools on the CD-ROM**
 - Spiral bound API manual part of Engine Commander®
 - VB/VC++/MS-Java tutorial manual include with Engine Commander®

The API has met all of the Working group requirements

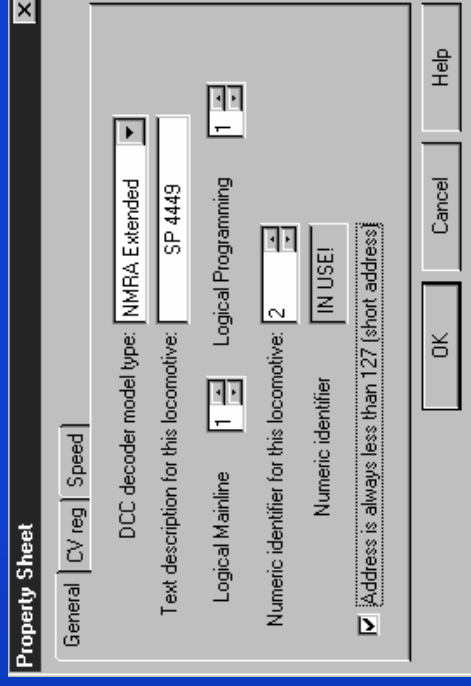
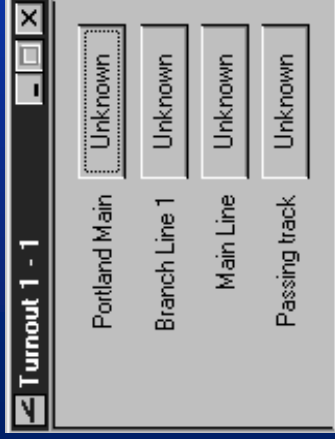


Engine Commander®

- **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..
 - A sensor displays state

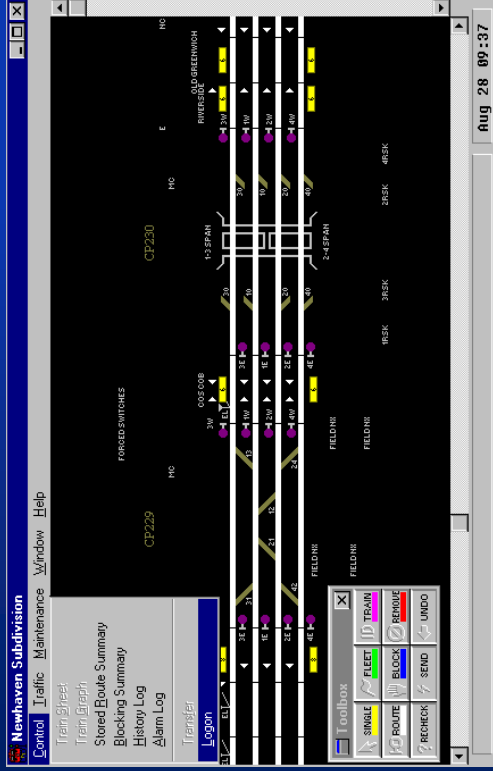


- **All applications can run at the same time**

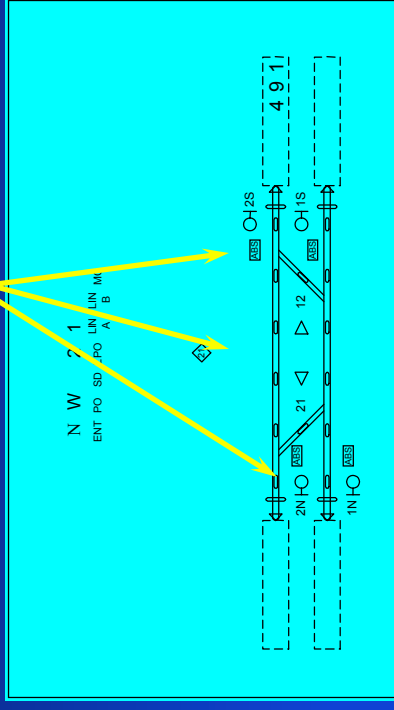


Computer Dispatcher® Pro1

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



CTC Panel view in Computer Dispatcher®



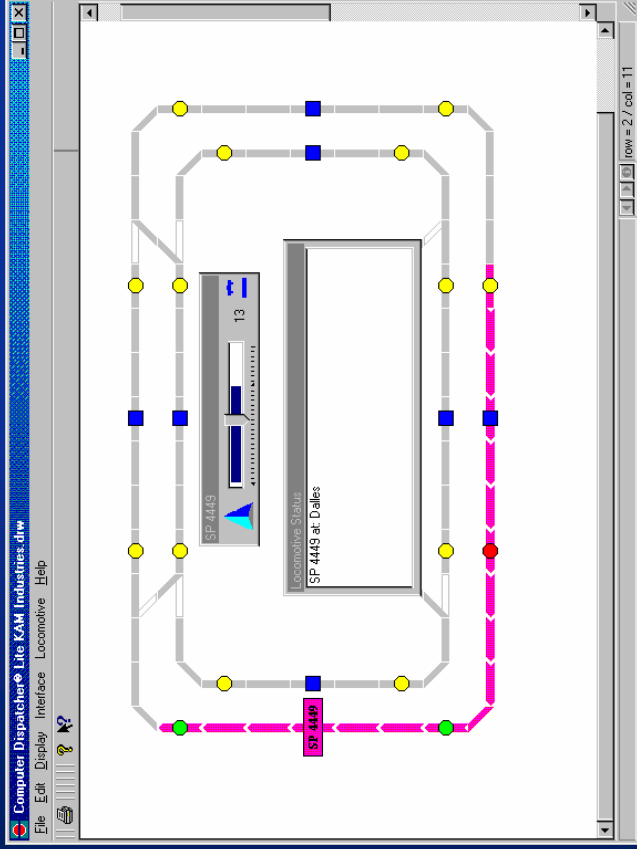
Computer Dispatcher®
model view of an active element
with full Entry/Exit (route) control



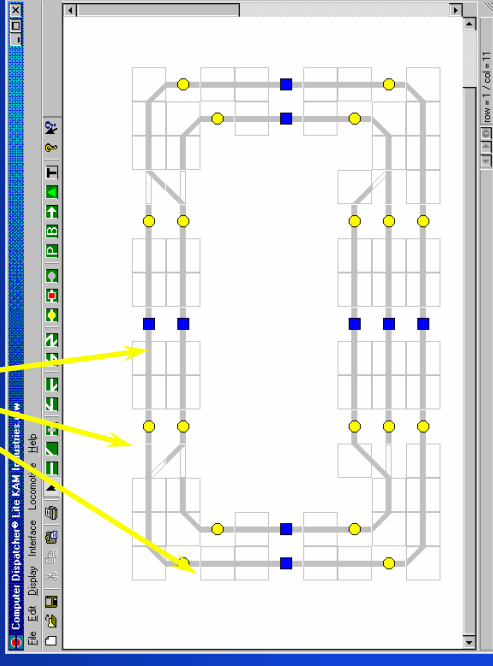
Computer Dispatcher® Lite

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

Switches and sensors



CTC panel view in Computer Dispatcher® with automatic block operation



Computer Dispatcher® Lite active elements



Agenda

- History
- Train Server® Interface architecture
 - Key concepts and terms
 - Execution model
- How the API works
 - Logical Devices
 - Decoder Structure
 - Command structure
- API summary
- Using the API in Visual Basic/MS Java or C/C++
- Questions/Answers

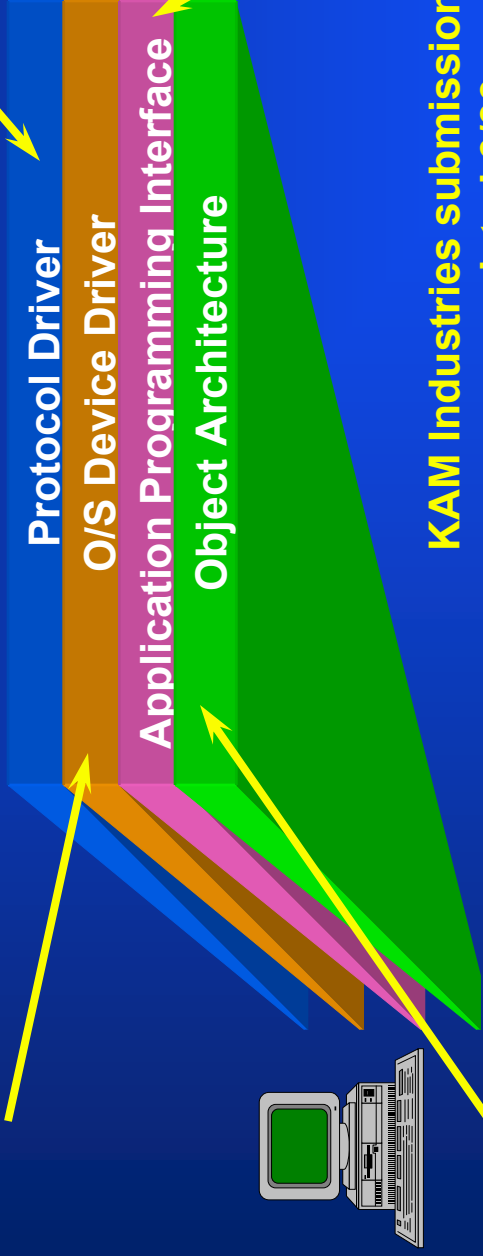


NMRA Application S/W Architecture Model

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

Katzer/Rice Draft Protocol Specification (7/98)

no activity



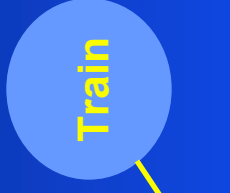
KAM Industries submission (7/98)
updated 6/99
specification located on [Http://www.kamind.com](http://www.kamind.com)

Rosa Proposal by Tannersoft (7/97)



API Architecture

- **API is a combination of a property/method model; with an execution framework**
 - Objects are not passed in the API; 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**



Agenda

- History
- Train Server® interface architecture
 - Key concepts and terms
 - Execution model
- How the API works
 - Logical devices
 - Decoder structure
 - Command structure
- API summary
- Using the API in Visual Basic/MS Java or C/C++
- Questions/Answers

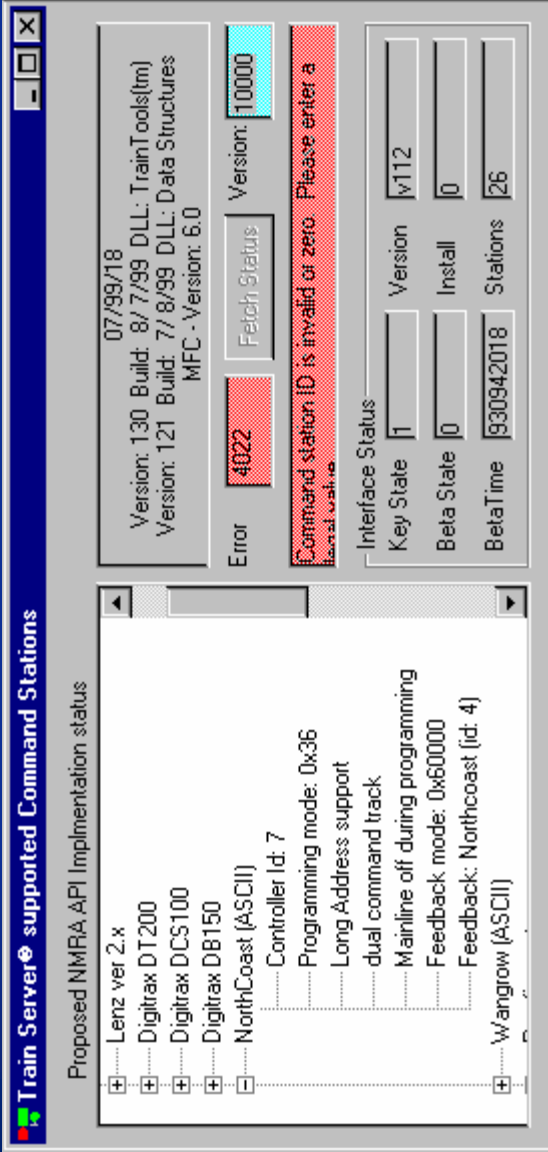


Architecture

- **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.



Demo

- Command station support
- 
- The screenshot shows the 'Train Server supported Command Stations' window. It displays a list of supported command stations and their implementation status. The list includes:
- 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)
- Below the list, there are several fields and buttons:
- Error:** 4022
 - Fetch Status:** Version: 10000
 - Command station ID is invalid or zero. Please enter a valid value:** (highlighted in red)
 - Interface Status:**
 - Key State: 1
 - Beta State: 0
 - BetaTime: 930942018
 - Version: v112
 - Install: 0
 - Stations: 26

- 26 different command stations supported

DCC Object (Cookie)

- **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
 - 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

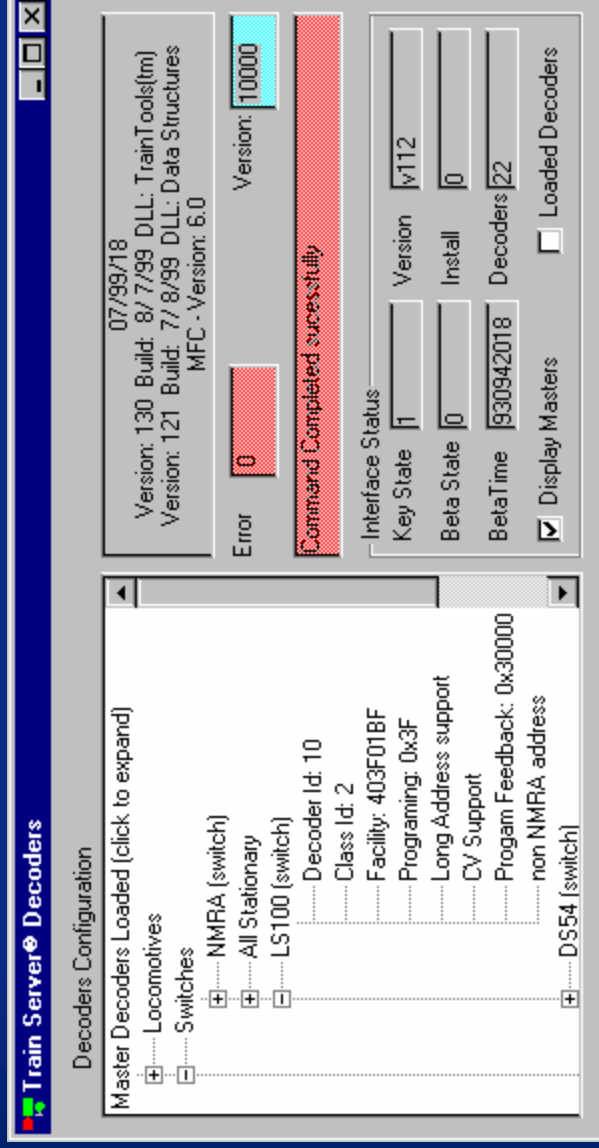


Architecture (cont.)

- **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 Detector1
 - » A set of decoder management functions were added to support application development
 - DccDecoderGetMaxModels(...)
 - DccDecoderGetModelName(...)
 - DccDecoderGetMaxAddress(...)
 - DccDecoderGetMfgName(...)
 - DccDecoderGetPowerMode(...)
 - DccDecoderGetModelFacility(...)
 - DccDecoderSetModelToObject(...)
- **DccObjects are mapped to master models**

Demo

- Decoder support and logical devices



- 22 master decoder models supported
 - Grouped by loco, sensors and switches
 - DB store can be updated from web servers

Agenda

- History
- Train Server® interface architecture
 - Key concepts and terms
 - Execution model
- How the API works
 - Logical devices
 - Decoder structure
 - Command structure
- API summary
- Using the API in Visual Basic/MS Java or C/C++
- Questions/Answers



API command summary

- **API Command classes**

- CV
- Engine
- Consist
- Accessory
- Command
- Programming
- Communications
- Command
- Decoder
- Cab
- Dispatcher operation
- Block definitions

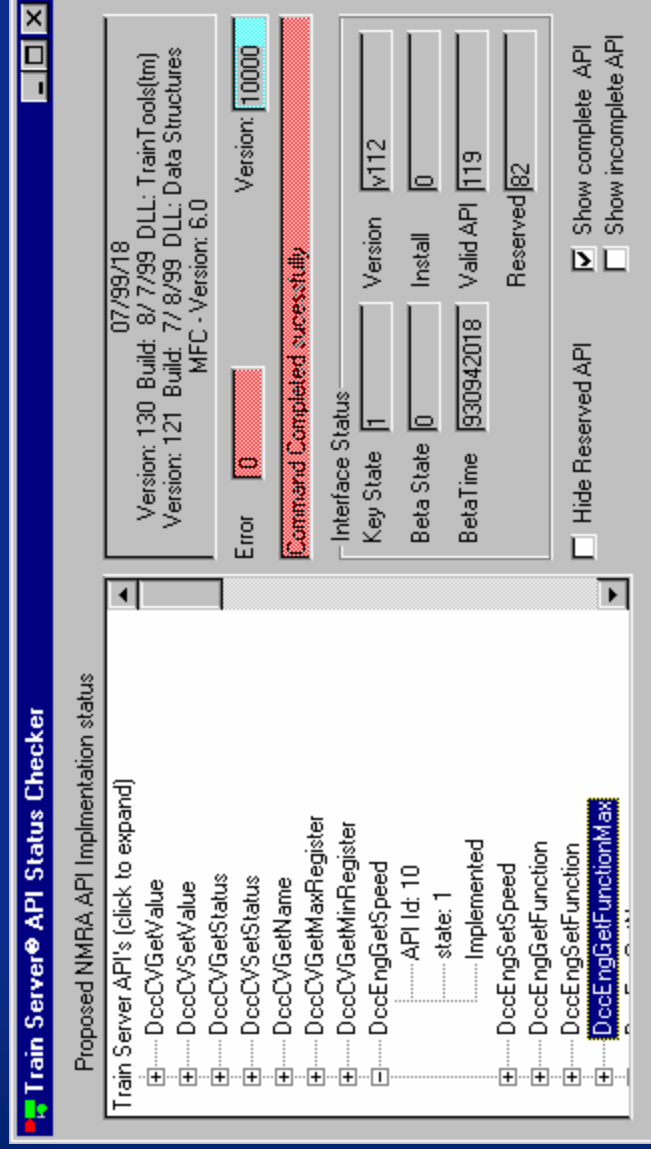
These are the major classes of commands needed in most DCC software applications.

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



Demo

- API supported in KAMs Train Server®



- 119 API's defined!



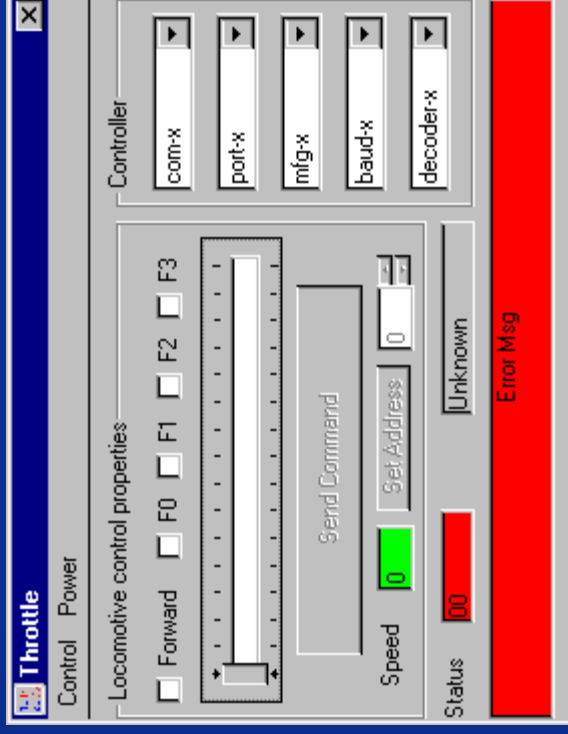
Agenda

- History
- Train Server® interface architecture
 - Key concepts and terms
 - Execution model
- How the API works
 - Logical devices
 - Decoder structure
 - Command structure
- API summary
- Using the API in Visual Basic/MS Java or C/C++
- Questions/Answers



Demo: Visual Basic Throttle?

- How is this Visual Basic application built?



- Lets look at how you program it



Visual Basic 5/6

- First step is to add the object reference

This is the key for all programming languages
We create an object reference

```
' This first command adds the reference to the TrainTools Interface object
Dim EngCmd As New EngComIfc
'
' Engine Commander uses the term Ports, Devices and Controllers
' Ports -> These are logical ids where Decoders are assigned to. Train Tools
' Interface supports a limited number of logical ports. You can
' also think of ports as mapping to a command station type. This
' allows you to move decoders between command station without
' losing any information about the decoder
'
' Devices -> These are communications channels configured in your computer.
' You may have a single device (com1) or multiple devices
' (COM 1 - COM8, LPT1, Other). You are required to map a port to
' a device to access a command station. Devices start from
' ID 0 -> max id (FYI; devices do not necessarily have to be
' serial channel. Always check the name of the device before you use
' it as well as the maximum number of devices supported.
' The Command
' EngCmd.KamPortGetMaxPhysical(lMaxPhysical, lSerial, lParallel)
' provides means that... lMaxPhysical = lSerial + lParallel + lOther
'
' Controller - These are command the command station like LENZ, Digitrax
' Northcoast, EasyDCC, marklin... It is recommend that
' you check the command station ID before you use it.
'
' Errors - All commands return an error status. If the error value is
' non zero, then the other return areguments are invalid. In
' general, non zero errors means command was not executed. To
' get the error message, you need to call KamMiscErrorMessage
' and supply the error number
'
' To Operate your layout you will need to perform a mapping between
```



Visual Basic 5/6 (cont.)

- next,
 - Write the subroutine to control the loco

```

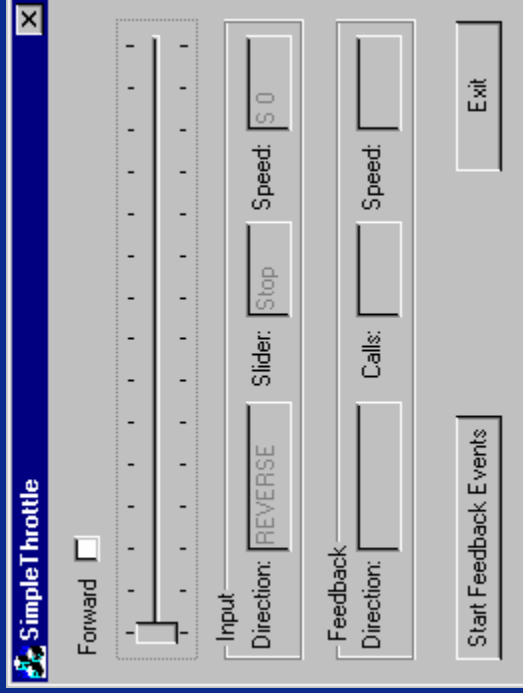
|*****|
| Send Command
| Note:
| Load the state of the decoder first, then send the command
|*****|
Private Sub Command_Click()
    'Send the command from the interface to the command station, use the engineObject
    Dim iError, iSpeed As Integer
    If Not Connect.Enabled Then
        ' TrainTools interface is a caching interface. This means that you need to set
        ' the CV's or other operations first; then execute the command.
        iSpeed = Speed.Text
        iError = EngCmd.DccEngSetFunction(lEngineObject, 0, FO.Value)
        iError = EngCmd.DccEngSetFunction(lEngineObject, 1, F1.Value)
        iError = EngCmd.DccEngSetFunction(lEngineObject, 2, F2.Value)
        iError = EngCmd.DccEngSetFunction(lEngineObject, 3, F3.Value)
        iError = EngCmd.DccEngSetSpeed(lEngineObject, iSpeed, Direction.Value)
        If iError = 0 Then iError = EngCmd.DccCmdCommand(lEngineObject)
        SetError (iError)
    End If
End Sub

```



Demo

- C++ example



Lets look at a C++ model

```

// Identify the interface of the object that we want to use...
MULTI_QI qi = {&IID_IEngComIfc, NULL, 0};
hr = CoCreateInstanceEx(CLSID_EngComIfc, NULL,
    CLSCTX_LOCAL_SERVER | CLSCTX_REMOTE_SERVER,
    pServerInfo, 1, &qi);
// add the security call at this point for compatibility for DCOM objects
//CoInitializeSecurity
// Now make the com conenction for the interface
if (SUCCEEDED(qi.hr))
{
    // Now get the remote TrainTools interface
    short sError;
    m_pEngIfc = (IEngComIfc*)qi.pIIf;
    GetVersion(&m_csIfcVersion );
    m_pEngIfc->DccPortGetMaxLogPorts(&m_iMaxLogicalPorts, &sError);
    m_pEngIfc->DccPortGetMaxPhysical(&m_iMaxPhysicalPorts, &m_iMaxSerialPorts, &m_iMaxParallelPorts, &sError);
    m_pEngIfc->DccMiscMaxControllerID(&m_iMaxControllerId, &sError);

```

This is the key for all programming languages
We create an object reference

Looks just like VB or Java, that is part of the design



C++ cont.

```

/*
 * NAME
 * DecoderGetModelFromCookie() - Get controller facilities.
 * RETURN VALUE
 * iModel - Decoder model ID.
 * DESCRIPTION
 * DecoderGetModelFromCookie() gets the decoder model ID.
 */

int TInterfaceDevice::DecoderGetModelFromCookie(long ICookie ) const
{
TRACE( "TInterfaceDevice::DecoderGetModelFromCookie( 0x%08lx ) - Entering\n",ICookie );
short iError;
int iLogCmdPort, iLogProgPort, iDCCAddr, iDecoderClass, iDecoderModel;
m_pEnglfc->DccDecoderTranslate(ICookie, &iLogCmdPort, &iLogProgPort, &iDCCAddr,
&iDecoderClass, &iDecoderModel, &iError);
TRACE( "TInterfaceDevice::DecoderGetModelFromCookie( 0x%08lx ) - Exiting: (%X)- Error\n", ICookie, iError );
return ( iDecoderModel );
}

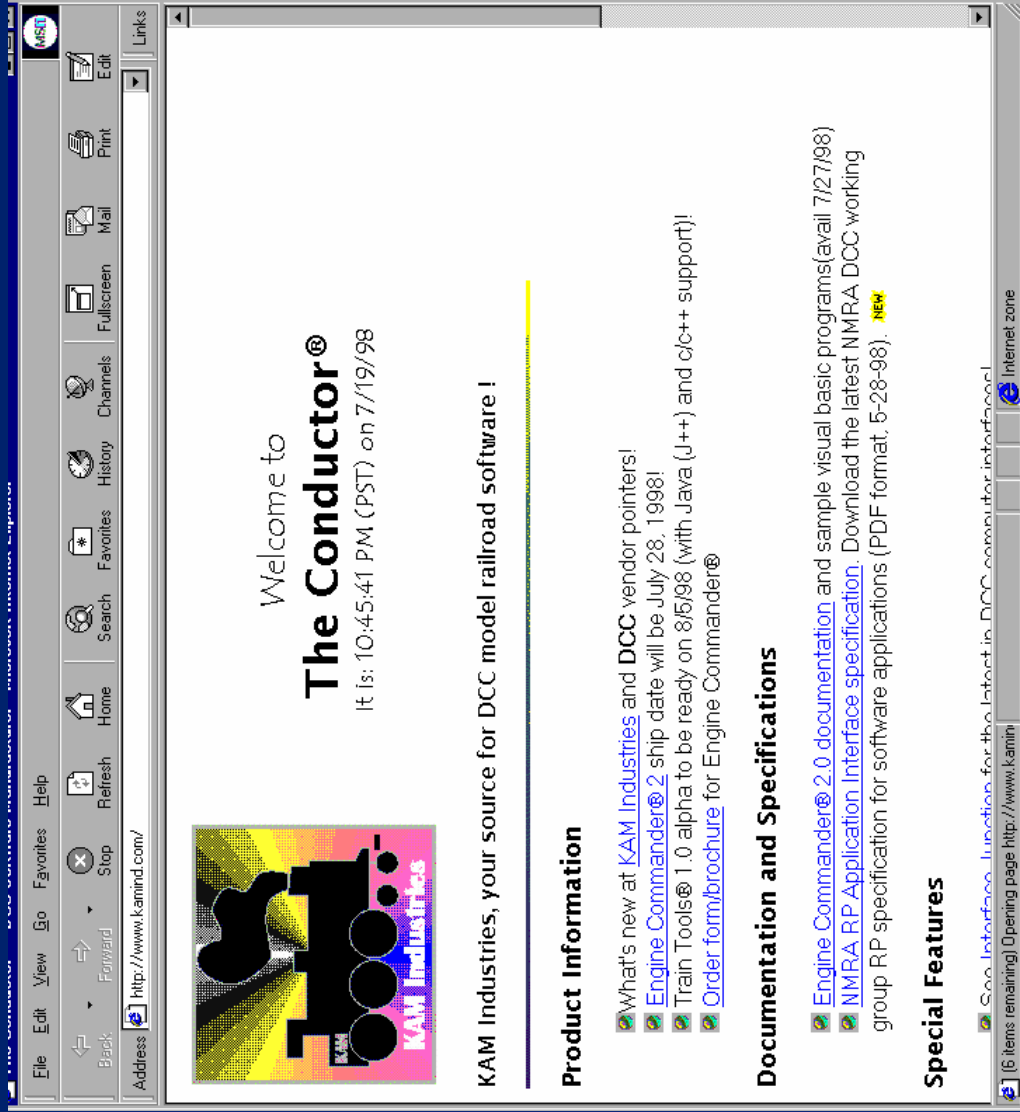
```

Where to from here?

- The API is on the CD-ROM
- If you wish a printed copy come to KAMs booth on Friday morning, we have about 30 copies with us (first come basis)
- Engine Commander® has a completed set of printed documentation
- All KAMs CD-ROM demos have full printed versions
- The proposed API is in the NMRA approval process



http://www.kamind.com



Questions ?

Matt Katzer
email: 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.



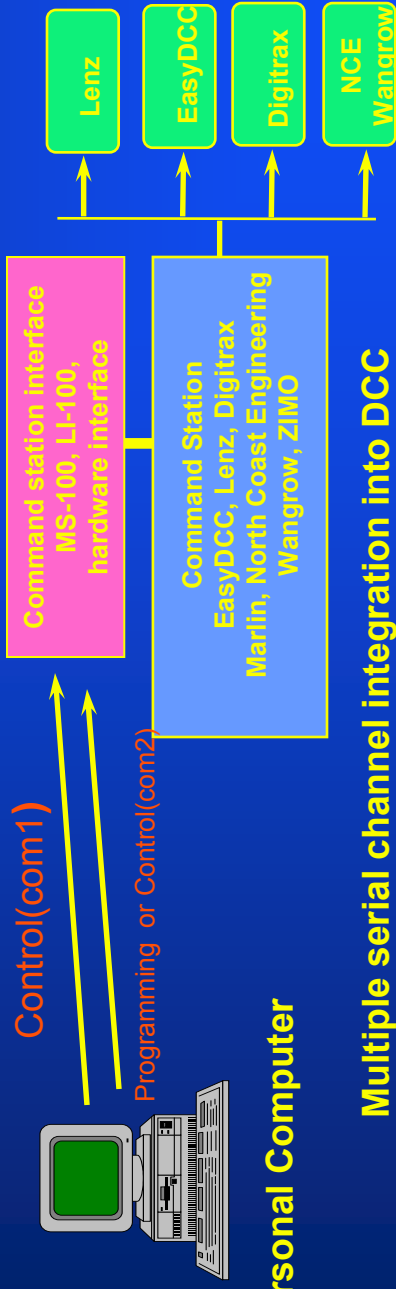
Appendix

- **Railroad environment**
- **Proposed NMRA API version 1.1**
(Train Server® version 2.121)



Railroad Environment

- **Must have NMRA DCC compatible engines**
 - Pick a DCC supplier based on current requirement for your locomotive
 - By 2000, all locomotives in a price range above \$100 will most likely have a decoder integrated into the unit
- **Command station equipment**
 - Expect a hybrid; plan for multiple command stations on layout
 - Model expected; one for programming the other for command and control



Personal Computer

Multiple serial channel integration into DCC



NMRA Proposed API

- **Engine**

```

DccEngGetSpeed();
DccEngSetSpeed();
DccEngGetFunction();
DccEngSetFunction();
DccEngGetFunctionMax();
DccEngGetName();
DccEngSetName();
DccEngGetFunctionName();
DccEngSetFunctionName();
DccEngGetSpeedSteps();
DccEngSetSpeedSteps();

```
- **Consist**

```

DccEngConsistGetMax();
DccEngConsistSetParent();
DccEngConsistAddUnit();
DccEngConsistRemoveUnit();
DccEngConsistGetParent();

```

NMRA Proposed API (cont.)

- **Functions**
 - `DccCVGetValue();`
 - `DccCVSetValue();`
 - `DccCVGetStatus();`
 - `DccCVSetStatus();`
 - `DccCVGetName();`
 - `DccCVGetMaxRegister();`
 - `DccCVGetMinRegister();`
- **Accessory Commands**
 - `DccAccGetFunction();`
 - `DccAccSetFunction();`
 - `DccAccGetFunctionAll();`
 - `DccAccSetFunctionAll();`
 - `DccAccGetFunctionMax();`
 - `DccAccGetName();`
 - `DccAccSetName();`
 - `DccAccGetFunctionName();`
 - `DccAccSetFunctionName();`

NMRA Proposed API (cont.)

- **Command Station**

```
DccOprGetStationStatus();
DccOprTurnOnStation();
DccOprStartStation();
DccOprClearStation();
DccOprStopStation();
DccOprPowerOn();
DccOprPowerOff();
DccOprHardReset();
DccOprEmergencyStop();
```

- **Programming**

```
DccProgramGetStatus();
DccProgramSetMode( );
DccProgramGetMode();
DccProgramWriteCV();
DccProgramReadCV();
DccProgramWriteDecoderToDataBase();

DccProgramReadDecoderFromDataBase( );
```



NMRA Proposed API (cont.)

- Communications**

```

DccProgramGetStatus();
DccProgramSetMode();
DccProgramGetMode();
DccProgramWriteCV();
DccProgramReadCV();
DccProgramWriteDecoderToDataBase();
DccProgramReadDecoderFromDataBase();

```
- Command**

```

DccCmdCommand();
DccCmdConnect();
DccCmdDisconnect();

```
- Cab**

```

DccCabWriteMessage();
DccCabReadMessage();
DccCabSetDccObject();
DccCabGetDccObject();
DccCabAdd();
DccCabDelete();
DccCabTranslate();
DccCabLookupDccObject();

```



NMRA Proposed API (cont.)

- **Decoder**

```

DccDecoderGetMaxModels();
DccDecoderGetModelName();
DccDecoderGetMaxAddress();
DccDecoderCheckAddrInUse();
DccDecoderGetMfgName( );
DccDecoderGetPowerMode( );
DccDecoderAddAddr()
DccDecoderGetModelFacility()
DccDecoderReconnectObject( );
DccDecoderChangeAddress( )
DccDecoderTranslate( )
DccDecoderSetModelToObject()
DccDecoderGetMaxSpeed( );
DccDecoderGetObjectCount()
DccDecoderGetObjectAtIndex()
DccDecoderDel();
DccDecoderGetErrorState( )

```

NMRA Proposed API (cont.)

- **Time**

```
DccMiscGetClockTime();
DccMiscSetClockTime();
```
- **Command Station**

```
DccMiscGetControllerName();
DccMiscGetControllerNameAtPort();
DccMiscGetCommandStationIndex();
DccMiscMaxControllerID();
DccMiscSetCommandStationValue();
DccMiscGetCommandStationValue();
DccMiscGetControllerFacility();
```
- **Misc**

```
DccMiscGetErrorMsg ();
DccMiscGetApiName();
DccMiscGetInterfaceVersion();
DccMiscSaveData();
```