Designer User Manual (Version 2.13)
1/27/2012

1 Introduction

This document is a tutorial on how to use **designer**, the layout description program. **designer** is a graphics oriented program for describing a model railroad layout. It creates an XML file that is used with **CATS** for controlling the model railroad through a dispatcher panel. Please see the **CATS** documentation for information on how to use **CATS**. Note: some of the features alluded to in this document have not been implemented yet in **CATS**.

This version of **designer** was written with Java 1.5.0_22 but should run under a 1.4 compatible JRE (Java Runtime Environment). It does rely on log4j and jdom, included with JMRI.

2 What is New or Different?

Date	Version	
July 29, 2005	0.15	• Added job descriptions (11) • Created metered interface for Loconet messages (9.1.3)
September 8, 2005	0.16	• Added a "turn off" option for an IOSpec (9.3.1) • Removed LOCKOFFCMD and UNLOCKOFFCMD because "turn off" covers them (9.3.1) • Changed file menus to start looking in the directory the program is run from (7) • Added file name filters • Added DTC (9.1.1) • Added a fast clock flag (14.1.6)
November 11, 2005	0.17	• Replaced the LL JMRI prefix with LH • Reworked selection menus to use JTables with fields that can be customized (10.4)
November 15, 2005	0.18	• Added a time on duty field to the Crew (12.1) • Added a way to set the Dead on Hours time (12.5)
February 5, 2006	0.19	• Minor bug fix
April 26, 2005	0.20	• Added station name (9.1.4)
September 18, 2006	0.21	• Implemented approach lighting (8.1) • Implemented software light flashing (8.1, 14.1.9) • Implemented file backup (7.1) • Reworked decoder lists so that all outputs can

		be a list (8.2) • Added drop downs so new JMRI device types can be added without rewriting this (8.3) • Added train label selection (14.1.7) • Added a panel file name (14.1.8) • Add keyboard shortcuts (14.1.17)
October 22, 2006	0.22	• Bug fix – no operational changes
February 10, 2007	0.23	• Added MR prefix (8.3) • Add instructions on creating a diode matrix (14.6) • Fixed bugs in constructing menus • Fixed bug in saving fonts • Adjustments (e.g. debounce) can be set in **designer** (14.1.5) • Can set **CATS** screen size in **designer** (14.1.16) • Added instructions for starting **designer** (3)
March 25,2007	0.24	• Fixed a bug that allowed a 0 length file to be backed up (see Section 7.1) • Pop up a warning when a zero length file is created (7.1) • Fixed a bug where "LN" was used for a JMRI turnout, rather than "LT" (8.3)
April 20, 2007	0.25	• Added an option for an inverted "tee" as the base to a signal (9.2.1, 14.1.10) • Added an option to remove the arrowheads on routes (14.1.10) • Added an option to not shrink track (6, 14.1.12) • Added an option to turnoff "smart" row wrapping (6,14.1.13) • Added an option to provide a lock on decoder commands (9.3.2,14.1.14) • Fixed a bug in reading the layout where text strings would be mangled (missing spaces, missing segments).
July 13, 2007	0.26	• Revised how signals work to account for intermediates. • Fixed a bug resulting in a zero length file when the layout is saved, if a signal has 0 heads.
August 18, 2007	0.27	• Reworked the screen that assigns aspects to each indication to make it shorter (8.1). • Added "Stop and Proceed" to the indication table (8.1). • Added "Advance Approach" indications to the

		indication table (8.1).
January 13, 2008	2.0	• Changed from using jdom-jdk11.jar to jdom.jar, as was done with JMRI 2. This makes designer incompatible with versions of JMRI prior to version 2.
March 2, 2008	2.01	• Modified crossings (diamonds) so that the crossing tracks are not in the same Block. This change will affect older panels that have crossings (9.3.3). • Added filled in circles on track ends supporting intermediate signals (9.2.1). • Added a Format when writing XML files. This fixes a problem that appeared in 2.0. • Added a user name field when defining signal heads to refer to a JMRI external SignalHead (9.2.1, 14.7).
April 4, 2008	2.02	• Added a minimum width on text fields when entering data • Added more explanation on how **CATS** determines what signal indication to use (8.1) • Fixed a bug when connecting non-adjacent tracks on block boundaries
July 15, 2008	2.03	• The cursor can be moved with the arrow keys. • The screen will scroll, if the cursor would move off the screen. • The screen does not scroll to the upper left corner when inserting or deleting a column or row. • Fixed up the designer.bat file under Windows so XML methods could be found.
December 28, 2008	2.04	• Added network configuration (14.8) • Expanded use of "Station" field (9.1.4)
February 2, 2009	2.05	• Fixed a bug in which CATS chains could not be defined because 0 is an invalid address (8.2). • Resynchronized with latest JMRI names (8.3). • Removed enforcement of an address being numeric to support things like CBUS.
July 4, 2009	2.06	• Added formatting of Train, Crew, and Jobs XML files • Reworked presentation of Train, Crew, and Job editor panels. • Added an "alignment" attribute for text in Train, Crew, and Job tables. • Added a global setting so that local control of

		turnouts can be locked out in CATS (9.3.2.1). • Added a trace to the log file capability when reading in an XML file, allowing a user to see which section in the file an error occurred by setting DEBUG (17.3). • Fixed some problems created by removing Sections and saving the resulting layout. • Added panels to define possible track crossings; this implements track work like "scissors" (9.3.3).
September 18, 2009	2.07	• Added separate color definitions for stations (9.2.2) • Added separate fonts for labels (9.2.3) and trains (10) • Improved the error handling when reading in an XML file (this should be transparent) • Fixed a bug when editing Chains • Changed the model used for tables (this should be transparent) • Fixed a bug when changing the type of user defined fields in tables
November 20, 2009	2.08	• Added a color to panel signals so that the user can define the color when the signal is not involved in a route (9.2.1 and 14.1.1) • Bug fixes (see the release notes)
February 28, 2010	2.09	• Added libraries for importing design elements (7.5). A result is that File->New and File->Open clear out any work that has been done. • Designer keeps track of where the last file used is located (7) • Removed some dead code when saving a layout • Fixed a bug when deleting a row or column • Added track coloring to highlight problems (6.3, 9.3, 17.5)
November 23, 2010	2.10	• Added details on how to create a single slip • Fixed a bug where non-adjacent track connections were lost • Fixed a bug where a station would disappear when the station dialog was invoked • Compiled under Java 1.5 so designer runs on older systems • Rewrote for Java 1.5 generics
January 7, 2012	2.11	• Complies with JMRI compiler settings • Completed lunar signal definitions

		• Added configuration information for working with JMRI Operations (15.3, 16)
January 23, 2012	2.12	• Fixed bug in editing switchpoints – only one leg would be saved
January 27, 2012	2.13	• Fixed a bug where new names could not be created on Sections

Table 1: Revision History

3 Installation

Designer requires some files supplied with JMRI - the log utility, XML parser, and the Java Collections so it (designer.jar) should be copied into the same directory as the JMRI executables. A "launcher" is supplied for each operating system (Windows, MacOS X, Linux) that contains the magic for setting the Java environment for running **designer**. The launcher also should be placed in the main JMRI directory. Alternatively, it can be edited to supply the execution path to the libraries.

Designer requires JMRI version 2.14, or later.

3.1 Starting designer – Windows

The easiest way to launch **designer** under Windows is to create a short cut to designer.bat and place the shortcut on your desktop. As explained in the troubleshooting section (17.1), it is also possible to bring up a command window, "cd" to the JMRI directory and just enter "designer".

3.2 Starting designer – Macintosh

There are at least two ways to launch **designer** on a Macintosh (OS X). One is to open a terminal window. In that window, navigate to where designer.csh is installed and execute it. Alternatively, you can download the Mac launcher and uncompress (unzip) it in the JMRI directory. This will create two icons. If you double click on the "CATS designer" icon, you will launch **designer**

3.3 Starting designer – Linux

Since **designer** requires graphics, you must install a windowing package (e.g. gnome or kde) on your computer. With the window manager running, you can open a terminal window. In that window, navigate to the JMRI directory, where you installed **designer**. You can execute **designer** from there (e,g, "./designer.csh", "csh designer.csh", etc). Your window manager may also allow you to create a shortcut that you can place on your desktop.

3.4 Windows 7

There are two keys in getting **designer** to work under Windows 7 (Vista): file permissions and picking up the correct Java run time environment. If you have problems see the troubleshooting section of this document for clues, but it is likely the execution paths are not set correctly or the file permissions are too restrictive.

4 Overview

Designer borrows heavily from the spread sheet paradigm of cells arranged in a grid of columns and rows into a rectangle. Columns can be added or deleted to make the grid array wider or narrower. Rows can be added or deleted to make the grid array taller or shorter. Each cell may contain track, a label, a picture, or a combination of the three[1]. Cells can be copied to the clipboard, removed or pasted from the clipboard, facilitating rapid duplication and editing of layout pieces.

A cell is a rectangle, with 4 edges (naturally). A section of track begins on one edge and terminates on another, without making any curves. All six possible orientations of tracks are supported. Thus, each track segment has two ends. If two (or three) track segments terminate on the same edge, then the program recognizes the switch points (however, see Section 9.3.3 for distinguishing between switch points and a crossing). If an end does not have switch points, then it is a candidate to be a block boundary. If it is a block boundary, then the cell that shares its edge also has a block boundary on its sharing edge. **Designer** recognizes blocks as all track segments that share adjoining edges without block boundaries. Signals can be associated with edges that are block boundaries. The signals can appear on the layout, on the dispatcher panel, or on both. Finally, all of these elements can have DCC addresses associated with them, which is how **CATS** controls the layout. Hopefully, all of this will be clearer as we walk through the manual.

There are many ways to wire up a model railroad. In designing this program, I have tried to support common ways of doing the wiring, that work in ways that are expected.

5 Getting Started

Begin by running **designer.bat** (or **designer.csh** for Linux and Macintosh systems). If everything installed correctly, you will see a window with a menu bar and a 2 by 2 grid of blank cells.

[1] A picture can be used to place something like a logo on the dispatcher panel. Because a picture overwrites all drawing, it doesn't make sense to place a picture and a label or track in the same cell.

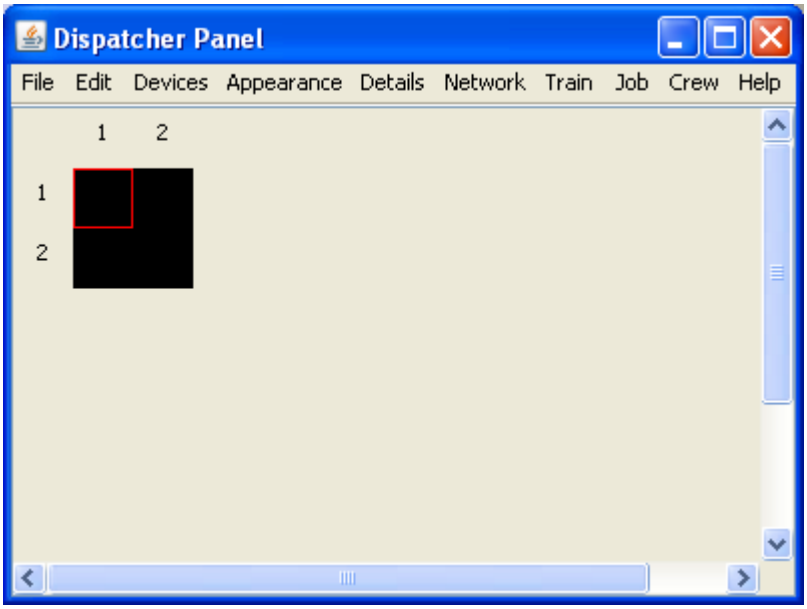


Figure 1: Opening Screen

The menu bar has the following categories of things you can do:

- *File* has operations on files, similar to the way other windows based programs have.
- *Edit* has operations for cutting, pasting, and changing the overall geometry of the grid array.
- *Devices* is for describing signals and decoders.
- *Appearance* is for changing the appearance of classes of things on the dispatcher panel (such as colors).
- *Details* is used for specifying the contents of a cell.
- *Network* is used when the **CATS** panel acts as a TrainStatus Server (See Section 14.8)
- *Train* is for describing frequent trains. These are trains which are usually or occasionally run.
- *Job* is for describing the Jobs on the railroad, which crew members perform.
- *Crew* is for showing who is assigned to which train.
- *Help* currently shows the Version number of the program and the date it was released.

The body of the window contains the drawing surface where the grid appears. As it grows, scroll bars will appear, allowing you to create layouts that are larger than the drawing surface and to move around the drawing. You can also resize the window by pulling on the lower right hand corner, but if you pull too far, then space will appear between the cells. Notice that there are headings on the left

most column and top row, to provide navigation references. When the drawing surface is scrolled, these references may scroll off the drawing surface. This will be fixed in a later version.

With that overview out of the way, let's get started.

6 Laying track

Probably the first thing to do is lay the track and add the details later. So, get the gross track plan down, and then add the stations, signals, etc. However, if you are comfortable with using this program and know how you want the track plan to look, you may save some effort by starting with full details on something like a passing siding, then using "cut and paste" to duplicate it. I have found that I make good progress starting by laying the mainline – a long straight horizontal track – and adding sidings and spurs off of it.

One other way to lay track is to add in a design element from a file – see Section 7.5.

6.1 The Grid Cell Cursor

Note that one cell is framed in red. That cell is where the track will be placed. Think of the red frame as a cell cursor. Click on Details -> Tracks. A pop-up window (Figure 2: Track Selection Pop-Up) will show all six possible track segments that can be placed in the cell. Select one or more by clicking the mouse on either the picture or the checkbox next to it. When done selecting segments (you can select more than one), click on Accept (or Cancel to not make any changes). Then, the tracks will be painted on the cell and the red frame will not have moved.

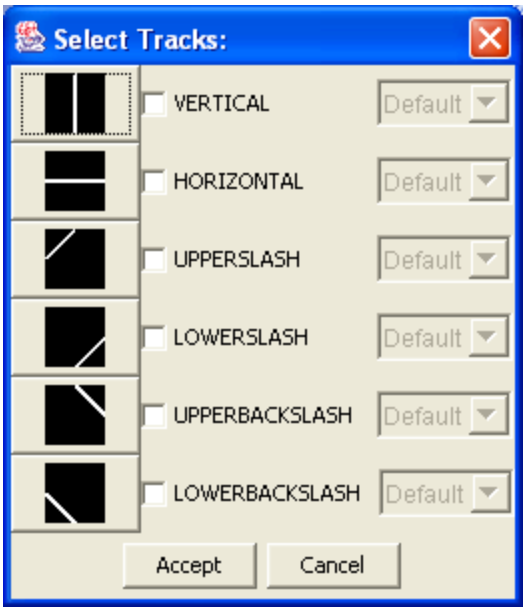


Figure 2: Track Selection Pop-Up

To pick another cell, position the mouse over it and click the left mouse button. When you need more cells on the drawing surface, click on Edit->Insert Row Above, Edit->Insert Row Below, Edit->Insert Column Before, or Edit->Insert Column After to add a new row or column of blank cells. Alternatively, you can move the cursor with the arrow keys.

The right column of the pop-up indicates the speed through a turnout. You can manually select one or use the default (which is correct the vast majority of the time). When "Default" is selected, the speed will be "Normal" for tracks not involved in turnouts. For the legs of a turnout, the "Default" will yield "Normal" speed for the "Normal" route (see Section 9.3) and "Medium" for all other legs. The speed is used in determining the signal (see Section 8.1).

6.2 Copy, Cut, and Paste

You can cut, copy, and paste multiple cells. The way to do this is place the cell cursor over one corner of the rectangle of cells that you want to cut or copy. Move the mouse cursor to the diagonal corner of the rectangle, and click the left mouse button while holding down the shift key. At that time, all selected cells will be framed in red and the Details menu item will be disabled[2]. Then, if you want to copy the cells, use Edit->Copy. Move the mouse to the upper left corner of where you want the copy placed and touch the left mouse button. That will move the cell cursor to the insertion point. Use Edit->Paste to place a copy of the cells at that point. Note that this operation will not change the number of columns

[2] The Details menu item is enabled only when a single cell has been selected.

or rows. If the area being copied will not fit on the existing cells, then nothing will be copied. If the operation is not what you expected, use the Edit->Undo operation to remove the paste. If you decide that you really did like the operation, you can use Edit->Redo to "undo" the "undo". The "undo" stack is not infinite, but will hold multiple operations[3].

Edit->Clear works similar to Edit->Copy except the selected cells are blanked.

You can also delete columns or rows. If you select Edit->Delete Row, then the row(s) where the cell cursor is located will be removed from the track plan. If you select Edit->Delete Column, then the column(s) containing the cell cursor will be removed. Should you do one of those operations in error, Edit->Undo will restore it.

6.3 Track Color and Block Definitions

Notice that the track you just placed is colored red. That is **designer** telling you that **CATS** is going to be unhappy with that track. **CATS** demands that every track segment be included in a block and for every block to have a name. Assuming that you have not created any block boundaries, **CATS** will see your layout as a single detection block. **CATS** will show only those blocks that are named (this is something else that I will fix someday), **so you need to name the block**. Position the cell cursor over a cell containing a track segment that is somewhere on the edge of the grid array, say in column 1. Select Details->Track Ends. This will pop up a window that lets you describe the edges of the cell that have tracks terminating on them. There is one sub-window for each edge that has a track. The sub-windows are labeled with the grid coordinates and the name of the edge. Check the "Block Boundary" box on the sub-window identifying the track end that is on the edge of the grid array. This will enable the Define Block button. Push that button and another window will pop up. Add a name for the block in the Block Name field. You can call it anything you want, but if you have your feeder wires or block detectors named in some way, then using that name is a handy documentation scheme.

[3] The undo/redo operations are pretty course. They only apply to changes at a cell level. They do not apply to changes in Details. They also may not give you back exactly what you had (or expect), if you string several together or do other things (such as Details between Edits).

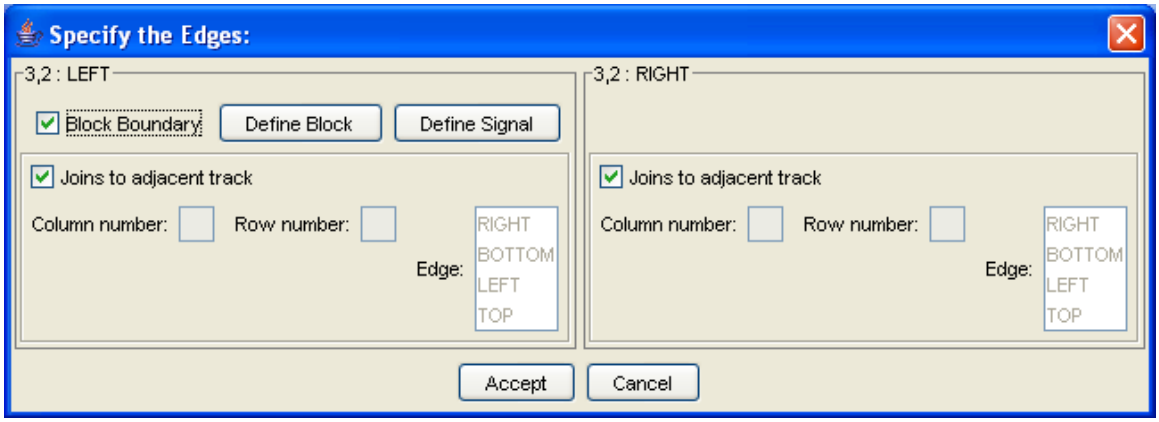


Figure 3: Grid Edge Window

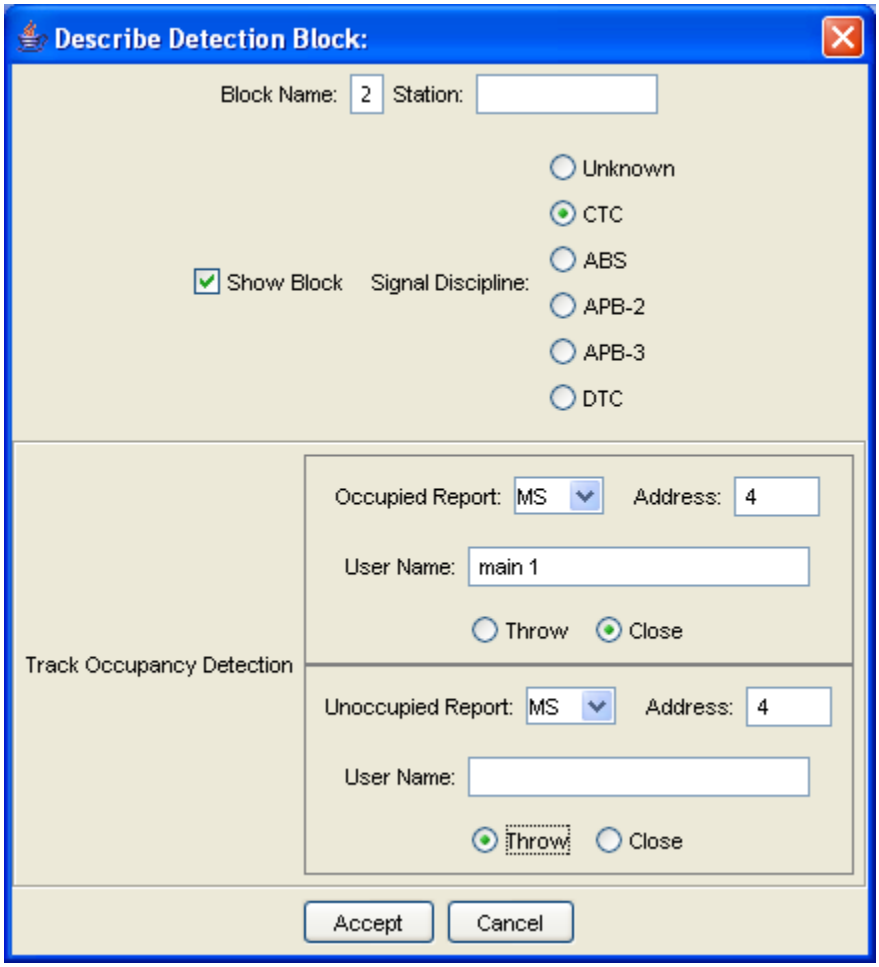


Figure 4: Detection Block Definition

So, the meaning of the track colors is:

- Red – a problem that needs to be corrected. It could be because the track is not included in a defined block or because a normal route has not been defined through switch points (see Section 9.3).
- Grey – the Block is defined, but does not have an occupancy detector defined ("dark territory")
- White – the Block is defined and has occupancy detectors.

These colors can be changed (see Section 14.1.1).

6.4 How **CATS** *Draws the Layout*

So, you name your block, save your work, and fire up **CATS**, but something doesn't look right. **CATS** did not preserve the same proportions! There are several things going on. First, **CATS** will attempt to put as much of your track plan on the viewing area as it can. This means you can resize the window and more (or less) of your track plan will show. In addition, **CATS** will try to "wrap" your diagram. Suppose your diagram is wider than the view area. **CATS** will start at the right hand side of the screen and work towards the left, looking for the column on the right half of the area with the minimum number of horizontal tracks. It will break the track plan there and try to place it below what is shown, at the left hand side. The concept is like a word processor breaking what you write into lines on the page. If you resize the window, you will see **CATS** making the "wraps" at different places[4]. This means that when you place track in **designer**, make it all linear – do not format the drawing area into bands. However, if you want to force a particular geometry you can do so and later on, this document will describe how to tell **CATS** that two track ends that do not touch on the drawing are logically adjacent.

Finally, you may notice that some columns are not as wide as others. This is by design. In order to fit as much layout on the view area as can fit, **CATS** will test if a column from **designer** can be displayed half as wide. It can be if it does not contain any track that is on a diagonal or has a label. So if you want the wide spacing, you may have to pad by putting two horizontal tracks in adjacent columns[5].

Some people do not like CATS changing what was created in designer. Under the Appearance menu item are two checkboxes that affect how **CATS** shows the track. The *Compress Screen* checkbox controls shrinking the width of track. If the box is checked, then **CATS** attempts to shrink the columns, as described above. The *Automatic Wrapping* checkbox controls how **CATS** wraps track. If the box is checked, then **CATS** tries to break a row on a column with the least number of tracks. Neither one of these checkboxes have any affect in **designer**. They only affect how **CATS** draws the track plan. They are options in **designer** so that they can be saved with the layout description. They remove the need to set those options whenever **CATS** runs.

[4] This "auto-wrap" feature can be overridden – see Section 14.1.13.
[5] This compression feature can also be overridden – see Section 14.1.12

7 File Operations

If you want to preserve your work when you end **designer**, you will need to save it. When you want to resume your work, you will want to load it back into **designer**. Both of these general operations require a file name and a place where the file is stored. The first time in a session when you save or load a file, **designer** will look in the same place that it is run for a file, using a standard navigation window. **Designer** remembers when you navigate to a different location and uses that location as the starting point, the next time a file is needed.

7.1 Save

Use File->Save to save your work. The first time in a session that you do this, the program will ask you for the file name to save your work under. It presents a standard browser window so you can place the file anywhere you have permission to save files. By default, it will choose the folder from which **designer** is run as the place to save your work.

designer has an inconsistency in its file naming convention. The file names it shows are folders and files ending in **.xml**. The filter is intended to reduce the number of file names you have to look through. The inconsistency is that **designer** does not force the .xml suffix on any file it creates. You have to add it when you create the file name. Consequently, if you save a file without that suffix, it will not appear on the file selection list by default. You can select "show all file types" (the specific procedure is operating system dependent) to see all the files in a folder.

The program tries to be safe in that it remembers if the track plan has been changed in any way. If it has and you try to do something that could lose those changes, it will ask you if you want to save the changes. It tends to be a little too safe and may ask you to save changes, even if you made none (another bug to locate and fix).

It is a good idea to save your work often because there are some bugs that can crash the program and I have not isolated the causes, yet. The program does attempt to keep one old copy of your work around. When you save a layout, it looks to see if a file with the same name already exists. If it does, it changes the existing file's name by appending ".bak" to it. This will replace any existing backup files. If the existing file is empty (0 length), it will not be renamed. If the program crashes while saving a file, the crashed file will not be backed up.

Because of the bugs, it is possible to create a zero length (empty) file. Though it cannot correct the problem, **designer** will warn you when it has done so. So that you do not lose everything when you quit, try removing some of the changes since the last save, try another save, remove changes, etc.

7.2 Save As

Suppose you have been saving your work and now want to save it under a different name, use the File->Save As menu option. If you use File->Save after File->Save As, it will be saved under the name provided to any previous File->Save. It will not let you Save As to the same file name twice in a session. I have not figured out why, yet.

7.3 Open

File->Open is the negative of File->Save and File->Save As. It is used to read in an existing layout file. This is the operation used to resume working on a layout. When this operation is selected, **designer** restores everything to as though it was just started. Since there may be some unsaved work, **designer** will ask if you want to save it before clearing it out. File->Open combines two other operations: File->New, followed by File->Import (see below).

7.4 New

File->New erases everything that has been done since **designer** started up (except anything saved will remain saved). It is used to start with a clean canvas without ending and launching, again.

7.5 Import

File->Import reads in a saved layout (a library) without erasing any existing work. It is a way to merge multiple layouts together, add some pre-canned design elements to the existing layout, insert existing signal definitions, etc. When a file is selected, **designer** will grab the track plan from the file and insert the upper grid corner of the trackplan at the grid cursor location. It will expand the layout in the horizontal and vertical directions as needed. Note that the library is not inserted, but replaces existing track; thus, preserving any track not overlaid. Selecting *Edit->Undo* will remove the tracks (but may not exactly recreate the replaced track). Selecting *Edit->Redo* will add the tracks back **at the current cursor location**. So, if you insert a library at the wrong place, you can undo, move the cursor to the right location, and redo.

Tracks, information associated with tracks (e.g. Block definitions), Stations, Signals, etc. will be added to the existing work. File->Import will also merge any *Devices* (Section 8) defined in the file, but not any *Appearances* (Section 14.1), *Trains* (Section 10), *Crew* (Section 12), or *Jobs* (Section 11). "Merging" is defined as "if something in the file does not exist in the current trackplan, it is added". This means that things in the library file will **not** replace things with the same name in the trackplan.

Here are three ways (by no means the only ways) to use libraries.

7.5.1 Design Elements

Over time, I hope a library of simple track structures (e.g. a passing siding or a wye) will be created. These structures (or design elements) can provide building blocks for assembling a layout, rather than having to build them by hand. To be effective, they should be saved without any layout specific information (e.g. decoder addresses, block names, etc.). They can be considered larger building blocks for a layout.

7.5.2 Modules

A module group might create a library of module descriptions. Each module description would be completely "wired" – decoder addresses, names, etc. When the modules are set up, a CTC panel can be created by importing the appropriate module libraries, in the order they are assembled.

7.5.3 Prototype Signals

Because signal definitions (Section 8.1) are imported and merged, it is possible to create libraries that do not have tracks, but only signal indications and aspects. These signal indications and aspects could be tailored for specific prototype railroads. They would allow **CATS** users to share the signal rules for their modeled railroads.

8 Defining Devices

Before diving into applying more details, it is useful to define things that will be used in other places. Pull down the Devices tab and see that signals, decoder chains, and JMRI devices can be defined.

8.1 Defining Signals

First, we will define the signal types that will be used on the railroad to protect the blocks. A signal template defines a class of signals. All signals on the layout are members of a signal template and the template describes what aspects the signal may show[6]. So begin by Devices->Add Signal Template. You will see a new pop-up window.

[6] An aspect is the appearance of a signal – number of heads (arms), color of each head (arm), color position, semaphore position, etc.

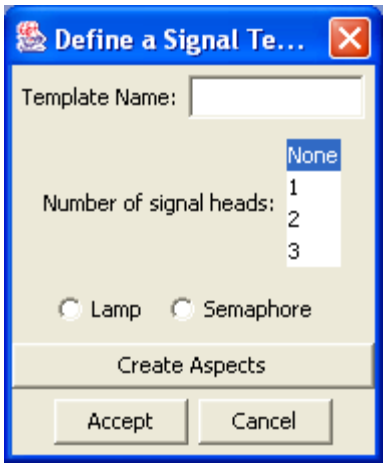


Figure 5: Signal Template Pop-Up

Through it, you name the template (the top field) so that you can refer to it later; the number of heads (some of the literature calls these *arms*) the signal has; and if the signal uses semaphores or lights. The later is really not important in how the layout operates, but is used for selecting which icons appear on the dispatcher panel. There is one other button in the pop-up, Create Aspects. When you click on it, you will see yet another pop-up window (Figure 6: Signal Aspect Template). This window is a table. The speeds of the block(s) being protected by the signals form the rows. The speeds of the following block(s) form the columns. There are three cells below the table for "stopping" indications. Each cell has a selection for the aspect for each head for the indication. If **designer** finds no track that uses a particular speed, it makes the column and row for that speed grey and those cells cannot be edited. This reduces the number of aspects that you define. See below for a discussion of what indications are supported. This window is used to tell **CATS** what aspect to use for each indication. If you have signals with similar numbers of heads, but different aspects, then you should create multiple signal templates.

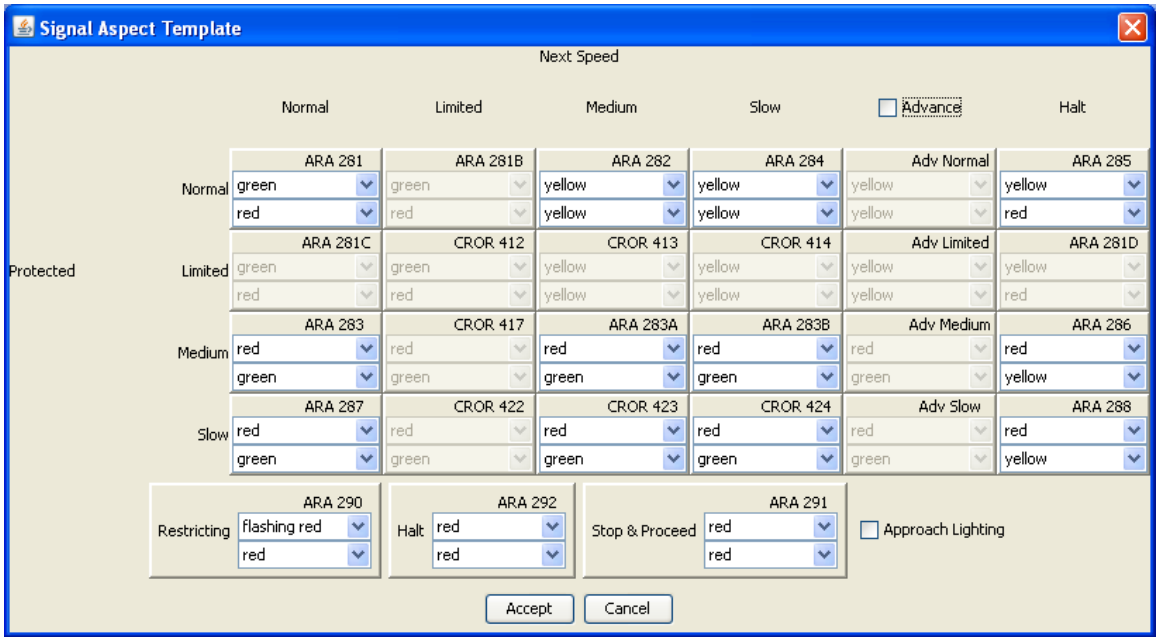


Figure 6: Signal Aspect Template

CATS implements 4 speed, 2 block[7] indication rules. This means that each signal shows the speed through the block(s) that it is protecting. The speed is either *normal*, *limited*, *medium*, or *slow* (if the signal is stop, then the train should not pass the signal, unless it is showing *stop and proceed* or the engineer is given permission to enter the block, and then the train should move at a *restricting* speed, defined as meaning the train must be able to come to a complete stop in ½ the visible range – usually under 15 mph). Furthermore, **CATS** looks at not only the block(s) protected by the signal, but the block(s) after it, protected by the next signal. To illustrate, consider single track mainline, signaled in only one direction, with travel from left to right.

[7] To be precise, it is a 2 signal look ahead (rather than block) because a signal can protect multiple blocks

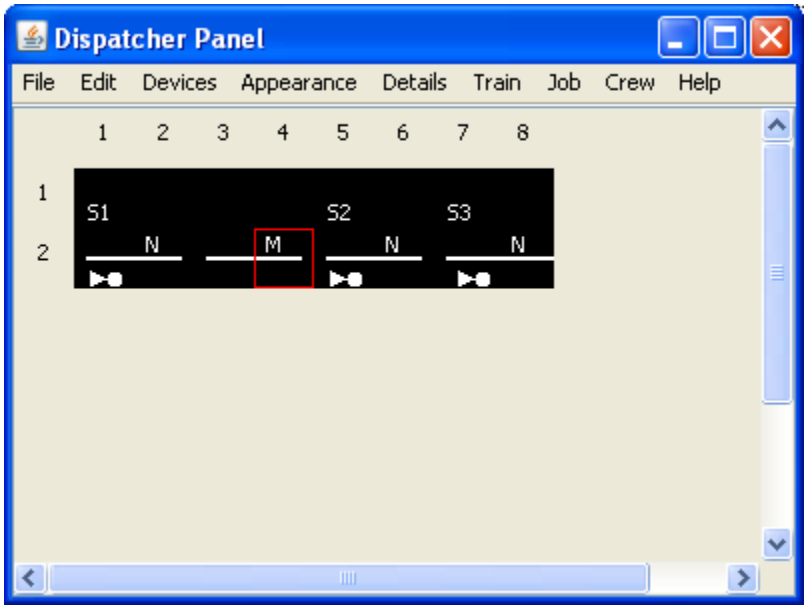


Figure 7: Example of Signal Coverage

S1 is the signal on the left. S2 is the signal in the center and S3 is the signal on the right. S1 protects two blocks. The first has a normal speed and the second is medium speed (assume it goes through a populated area with several crossings). The other two signals protect blocks with normal speed. The protected speed for S1 will be medium because that is the slowest speed protected by the signal. The protected speed for S2 and S3 will be normal. Thus, the "clear" indication presented by S1 will be the cell at the Medium row, and the Normal column, ARA 283 of Figure 6. When S2 is showing Halt, S1 will show the cell at row Medium, column Halt (ARA 286). Suppose Advance is checked and S3 is showing Halt. S2 will show row Normal, column Halt (ARA 285), S1 will show row Medium, column Advance (Advance Medium).

This may seem inflexible, but is actually more complex than is needed on most model railroads and the ability to define multiple signal templates with different aspects provides you with ways to work around the limitations. For example, to simulate route signaling (where the aspect shows the route the train will take rather than the speed), define the straight path through an OS section as *normal* speed and a track section on the diverging route as *medium* speed. There are subtle differences in the indications, depending upon which track section(s) are set to *medium*. If the siding is *medium* speed, then when the switch at either end is set for the siding, the protecting signal will show *Medium Clear* or *Medium Approach* (depending upon the signal protecting the next block). However, if the diverging route at only one end is set to *medium*, then only its protecting signals will present a *Medium* indication. The signals at the other end will not. This difference is because a signal takes on the most restrictive speed of any block it protects. The signal protecting entry into the siding from the main protects the

diverging route through the switch and the siding. Since the siding is protected by both end signals (the signals overlap), making the siding speed *medium* affects both end signals (but does not affect the exit signals from the siding). Making the diverging route of one turnout *medium*, affects the entry signal into the siding and the exit signal, but neither signal at the other end of the siding.

The 4 speed, 2 block analysis may seem like overkill, but it lets you show off those multi-head signals with solid and flashing lights. Furthermore, you do not have to use all the capability. If you set all the main track speeds to *normal* and all siding speeds to *medium*, the signals will probably work like you want them to. Alternatively, do not set any speeds and **CATS** will consider all diverging routes to be *medium*; thus, providing route signaling.

Notice that some of the cells are in grey and cannot be changed. With the exception of the "Advance" column, these indications are not needed because no track has been defined with that speed. You should be aware of a potential problem. The aspects are created based on the speeds used at the time the aspects are created. If you later assign a track a speed not used before, the aspects will be out of date. However, every indication has a default aspect, so if you do not replace the aspect, **CATS** will use the default.

The "Advance" column is special. **CATS** actually looks more than two signals ahead. If the third signal is in a "stop" state, the second signal will show "Approach". If you check the "Advance" column heading, rows in the column can be edited and the indication will support "Advance Approach". This provides the engineer with possibly two signals before seeing the "stop". **CATS** does this based on the following:

- A stop condition cancels out the speed.
- A stop condition in the next block forces an Approach speed in the protected block, which cancels the speed.
- Thus, the "Advance" indication combines the speed of the protected block with the "Approach" of the subsequent block to yield "Advance Approach" (normal followed by Approach), "Advance Limited Approach" (limited speed, followed by Approach), "Advance Medium Approach" (medium speed followed by Approach), and "Advance Slow Approach" (slow speed followed by Approach).

Advance indications should be used to protect short blocks (which is probably more typical in model railroads, than on the prototype), to give the engineer some advance notice that the train needs to be slowed down.

On the Create Aspects window (Figure 6: Signal Aspect Template), there is a pull down selection for each head for each indication. These are the common colors or blade positions. Some of them require flashing. If your hardware supports flashing, then use your hardware. If your hardware does not support flashing, then **CATS** will do it in software. These selections are only labels (much like the name of the signal template is a label), so if you have a blue light and no white,

then pretend that the white selection actually selects the blue and use white where you want blue. **CATS** does not read the color on the layout and knows no difference. You can use red as green and vice versa, if you want, but that could get confusing quickly.

The top selection always applies to the top head. The lower selection always applies to the head under it. The bottom selection always applies to the lowest head.

Though not exactly correct for 4 speed, 2 block signaling, the following chart shows how **CATS** decides what indication to give to a signal. To use it, read down the left column, locating the speed of the block(s) being protected by the signal. Then read across the row for the column with the speed of the following block(s). The unqualified numbers refer to the AAR rule. The CROR numbers are the Canadian Rail Operating Rule. The latter are included to fill in gaps in the AAR rules. You should not get stuck on these rule numbers. They are used to provide a common reference for indications. Every railroad seems to use different numbers. For example, on the BNSF, AAR rule 281 (Clear) translates to BNSF Rule 9.1.3 (also named Clear).

Next / Protected	**Normal**	**Limited**	**Medium**	**Slow**	**Halt**
Normal	281	281-B	282	284	285
Limited	281-C	CROR 412	CROR 413	CROR 413	281-D
Medium	283	CROR 417	283-A	283-B	286
Slow	287	CROR 422	CROR 423	CROR 424	288
Restricting	290	290	290	290	290
Halt	292	292	292	292	292

Table 2: Indication Rules

Acknowledgment: James P. G. Sterbenz and Mark D. Bej, M.D.

The following example shows the aspects the Crandic uses for each indication. G means green, Y means yellow, R means red, and r means flashing red. An * signifies that there is no situation on the Crandic where it can happen, but it is included for completeness[8].

Signal / Indication	**1 head light**	**2 head light**	**3 head light**
281: Clear	G	G/R	G/R/R
281-B Approach Limited	Not Used	Not Used	Not Used

[8] Someday, I would like to give the dispatcher the ability to place "slow orders" on a segment of track and the signals could reflect the "slow orders", so the dispatcher would have the ability to change the effective speed of any segment of track and some of these situations could happen.

282: Approach Medium	G	Y/Y*	Y/Y/R*
284: Approach Slow	G	Y/R*	Y/R/R*
285: Approach	Y	Y/R	Y/R/R
281- Limited Clear	Not Used	Not Used	Not Used
CROR 412 Limited Approach Limited	Not Used	Not Used	Not Used
CROR 413 Limited Approach Medium	Not Used	Not Used	Not Used
CROR 414 Limited Approach Slow	Not Used	Not Used	Not Used
281-D Limited Approach	Not Used	Not Used	Not Used
283: Medium Clear (Diverging Clear)	Y	R/G	R/G/R
CROR 417 Medium Approach Limited	Not Used	Not Used	Not Used
283-A Medium Approach Medium	Not Used	Not Used	Not Used
283 – B Medium Approach Slow	Y*	R/Y*	R/Y/R*
286 Medium Approach (Diverging Approach)	Y*	R/Y	R/Y/R
287 Slow Clear (Diverging Clear Slow)	Y*	R/Y*	R/R/G
CROR 422 Slow Approach Limited	Not Used	Not Used	Not Used

CROR 423 Slow Approach Medium	Y*	R/Y*	R/R/Y*
CROR 424 Slow Approach Slow	Y*	R/Y*	R/R/Y*
288: Slow Approach (Diverging Approach Slow)	Y*	R/Y*	R/R/Y
290 Restricting	r	r/R	r/R/R
292: Stop	R	R/R	R/R/R

Table 3: Crandic Aspects

Finally, the signal aspect has an "Approach Lighting" checkbox. Set the checkmark if you want **CATS** to turn on a signal only when the block in front of it is occupied. This means that your signals must support "off" as well as the aspects you define. If you select approach lighting, be sure your hardware can turn a light off.

8.2 Defining Decoder Chains

Sometimes you want multiple commands to be sent to the layout when something happens. For example, in a staging yard, you select a track and want the ladder to align itself for that track (see also Section 14.6). A chain associates a sequence of decoder commands with a single pseudo-decoder. The *Decoder Chain* pull down is used to add, delete, or edit a chain.

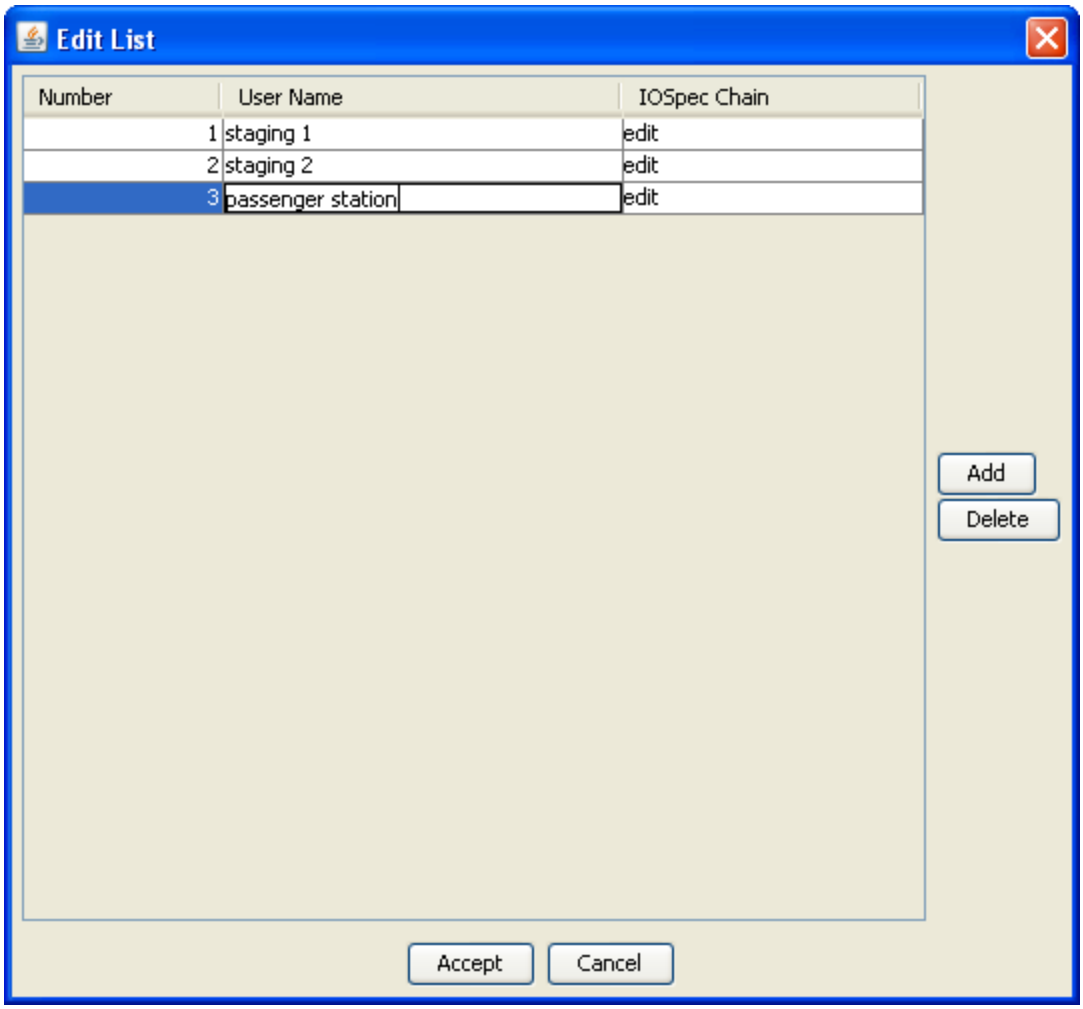


Figure 8: List of Decoder Chains

The number is used in constructing the JMRI system name and the User Name is the JMRI user name (see 13.1). The number cannot be 0. A JMRI system name is a 2 letter prefix, followed by a number, which is the "address" of a device of the type indicated by the prefix. For a decoder chain, the prefix is "IC" (Internal Chain) by default, but this can be changed (see the next section). The "edit" column is a button, when pushed launches the chain editing window.

A chain is edited (defined) with the following window.

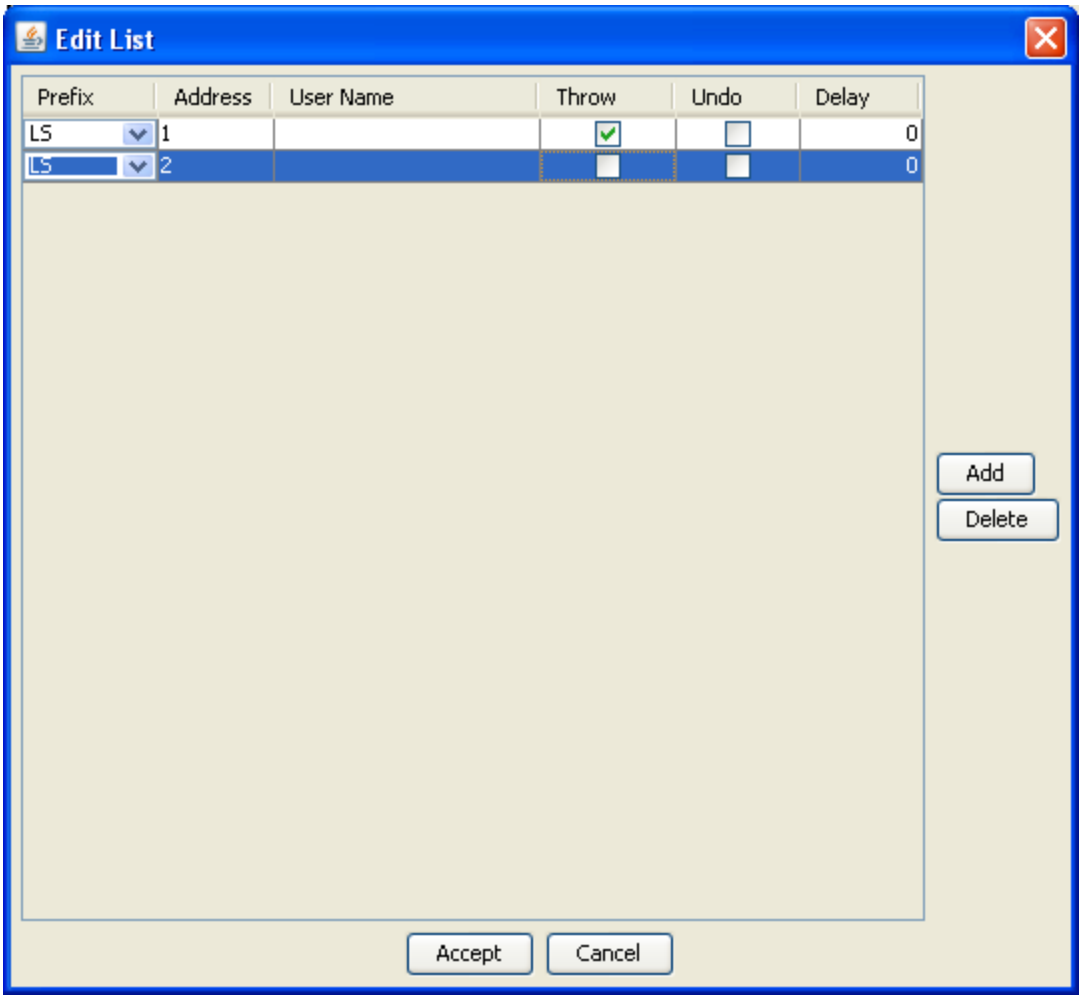


Figure 10 Chain Definition

This window looks different from others that are used for specifying a decoder address (13.1) because there can be an unlimited number of decoders in a chain. A chain can be in a chain. However, **CATS** will run out of memory quickly if one chain includes another which includes the first. The delay column deserves special notice. It specifies the number of milliseconds after sending the decoder command before the next is sent. This is useful for doing something like creating a pulse. The other columns will be discussed later.

In general, when invoking a Chain, you should select the "Close" box. When the Chain is executed, it will execute the defined setting for each of its elements. If the "Throw" polarity is checked, then the opposite setting will be requested of each element. If "Undo"is checked, then the opposite request will be honored; otherwise, the decoder will ignore the opposite request.

You can think of a decoder chain as a little brother to the JMRI route device type. Chains can be used for sending commands to a layout, but not for receiving status from a layout. **Designer** will not prevent the latter, but **CATS** will.

8.3 Defining JMRI Devices

CATS uses JMRI libraries for communicating with the layout. **CATS** builds on the work of the JMRI developers. This means it has to be told how to use JMRI code. This panel is used to tell **CATS** which JMRI interfaces to use.

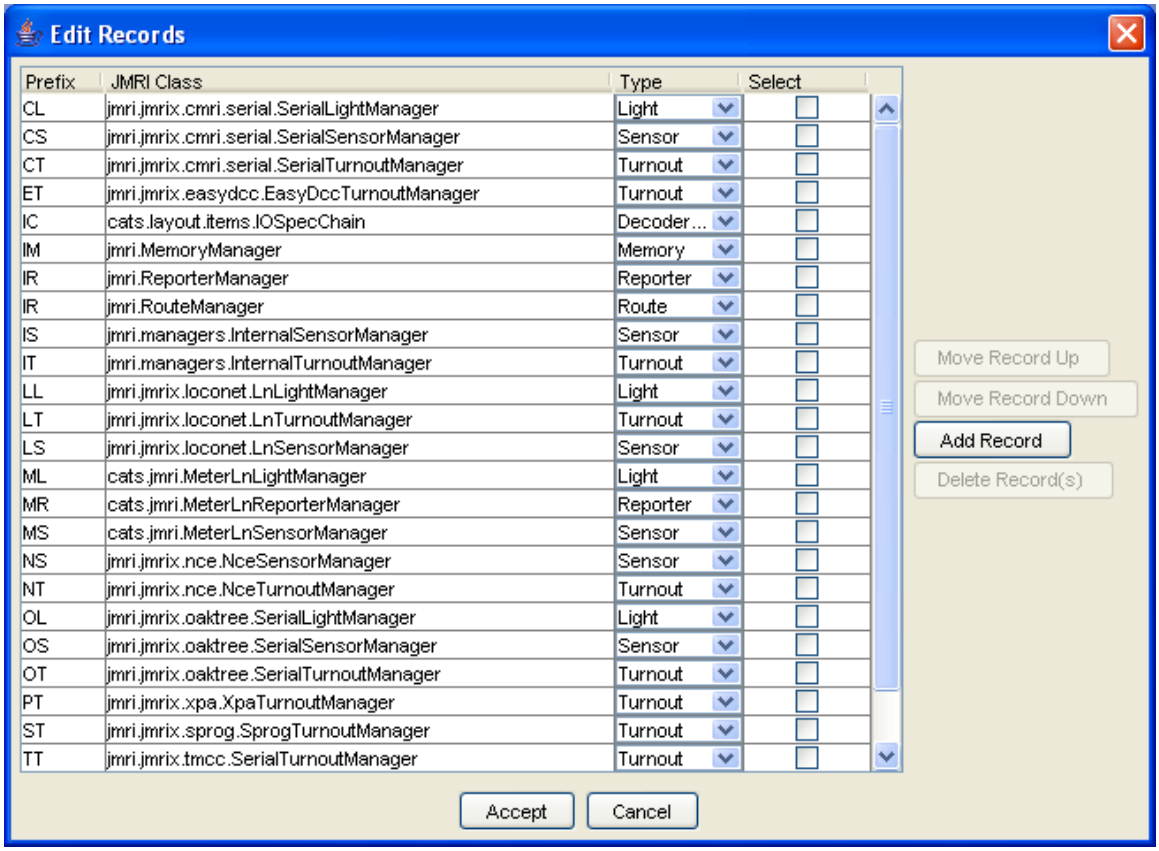


Figure 11: JMRI Device Prefixes

Each row corresponds to a JMRI device. The Prefix column is the two letter designator for the JMRI device and should follow the JMRI naming convention. The first letter is the interface to the layout and the second is the type of device. You can change them, but many JMRI device managers will reject the different designator. The JMRI Class column is for advanced users. It refers to internal pieces of the JMRI code. You should probably not change any of these. The Type column is the type of JMRI device. As of this writing, the JMRI interfaces (first letter) are:

Letter	Connection system
A	CTI Acela
C	C/MRI serial
D	Direct packet drive (SRCP)
E	Easy DCC
G	Grapevine (ProTrak)

I	Internal
K	Maple Systems
L	Loconet
M	Metered[9] and MRRLCB
N	North Coast Engineering
O	Oak Tree
P	XPA
Q	QSI programmer interface
R	RPS
S	SPROG
T	Lionel TMCC
U	ESU ECoS
V	TracTronics SECSI
X	Xpressnet via LI100
Z	Zimo MX-1

Table 4: JMRI Prefixes

The JMRI device types (second letter) are

Letter	Device
C	Chain[10]
H	Signal head
L	Light
M	Memory
P	Power manager
R	Reporter
R	Route
S	Sensor
T	Turnout
X	Logix

Table 5: JMRI Device Types

The final column is a check box. It should be checked if your layout will use those kinds of devices and not checked if it will not. Most people will need to only change the check boxes in this window. However, as the JMRI developers add more interfaces, **CATS** can adapt by simply adding new rows (or changing columns) in this table.

[9] This prefix is not part of JMRI. It was created to solve some problems we were having with Loconet devices and is the default prefix. A "metered light" waits between sending Loconet commands. The wait is to give the device time to react to the command. A "metered sensor" holds an active report for a few seconds before reporting it to **CATS**. It is an attempt to filter transient occupation reports. If the sensor does not stay active for a few seconds, it is simply dropped. Our experience has been that detectors usually report active immediately, but hold inactives for up to 4 seconds. Thus, an occupied report should never be less than four seconds.

[10] ditto

Only the JMRI designators checked will appear in the decoder selection windows (8.2 and 13.1). Though you could check them all, you will save yourself some work when defining decoders if you check only the ones you want to use. In addition, **CATS** will not work as hard because it will not create unused device managers.

With a big exception, in general, it is sufficient to select the "Sensor" devices for your interface (for receiving status and for sending commands). JMRI provides additional logic for the other device types, which could interfere with **CATS** (or vice versa). JMRI has created the "Memory", "Recorder", and "Reporter" devices for the use of supplemental programs. **CATS** recognizes them so **CATS** actions can stimulate other scripts and **CATS** can respond to the actions of other scripts.

The exception is Loconet. Loconet users should be aware of a couple of things. The JMRI "LH" decoder type is missing. This is because JMRI associates "LH" decoders with the SE8C and "LH" does not fit into the JMRI decoder manager scheme. Second, there are three message types exchanged with Loconet decoders:

- OPC_INPUT_REP. This is the message generated by Loconet sensors (e.g. BDL block detectors and the SE8C DSx inputs). The "MS" decoder type listens for and sends this message type.
- OPC_SW_REP. This is the message generated by push button switches (e.g. the SE8C SWx inputs). The "MR" decoder type listens for and sends this message type.
- OPC_SW_REQ. This is the message sent to stationary decoders (e.g. SE8C or DS-54) requesting that they change an output. It is also the message sent by a throttle to perform the same function. The "ML" decoder type sends and listens for this message type.

So for connecting to a Loconet network, you will need to determine which of the three messages is accepted or sent by each of your stationary decoders. For controlling decoders, you will usually select "ML". For receiving status from the layout, it will usually be "MS". For receiving push buttons, it will usually be "MR".

If you are not familiar with JMRI naming schemes, the "L" designator is for "Lights". Do not confuse a "Light" with a signal light. A "Light" is intended for controlling ambient lighting, building lights, and other illumination devices that have the characteristic of "brightness". A single bulb on a signal is usually controlled by a "Turnout" ("T") (as is the position of a turnout). In other words, a "T" controls a binary output.

9 Working with Tracks – Detection Blocks, Switch Points, and Crossings

Let's assume that you have enough track drawn that you can see that it fits on **CATS** and has what you want the dispatcher or computer to know about (the two

are not the same, as will be explained in the discussion about hidden track). This might be a good time to begin adding details to the detection blocks (or breaking blocks into smaller detection blocks). A detection block is represented on the track plan as logically contiguous cells, sharing a common detector and being protected as a group by signals. Thus, each detection block has a name[11] (**remember that the block will not show up on CATS unless it has a name**) and an optional detector. If the end of one track section is a block boundary, then the end of its neighbor that it shares an edge with must also be a block boundary. So, let's look at Details->Track Ends.

Begin by placing the cell cursor over a cell that has at least one track section in it, and then select Details->Track Ends. A pop up window will appear that contains a box describing each edge of the cell on which a track terminates.

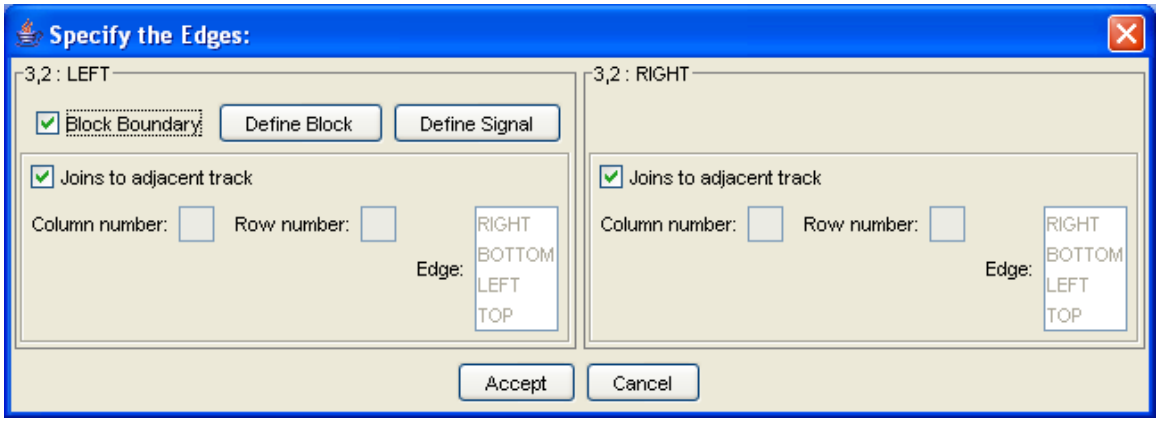


Figure 12: Edge Specification

For reference, each box has the coordinates of the cell (column, row) and the name of the edge. There are four kinds of boxes (2 are shown above):

- If two track ends touch on an edge and the edge in the adjacent section also has two tracks that are continuations, then the edge could be either switch points or a crossing (see Section 9.3.3).
- If more than one track section ends on the edge, then the edge has switch points and there is a button for providing more information about the points.
- If the shared edge on the adjacent cell does not have switch points, then there are three buttons for defining a block boundary.
- If the shared edge on the adjacent cell has switch points, then there are no additional buttons – the edge cannot be a block boundary because its neighbor cannot be a block boundary[12].

[11] A track in a Block that does not have a name will be drawn in **designer** in red (Section 6)
[12] The program does not permit points to be on block boundaries because drawing the block boundary gap and points in **CATS** does not look good.

However, all edges have a check box for "Joins to adjacent track". This checkbox is used to avoid drawing loops of track (and to keep drawings clean, as illustrated in a "flyover" – See Figure 23: Flyover). On the prototype, there are very few track plans equivalent to what is an oval (a continuous) loop in the model railroad world. **Designer** will let you lay out a loop, but **CATS** will crash trying to handle it. So, the way to lay out your diagram is linear, running left to right. Position the cell cursor over the cell on the far left or far right, that you want to connect to the other side of the diagram and uncheck Details->Track Ends -> Joins to adjacent track on the edge that does not have a neighbor. This enables the "Column number", "Row number", and "edge" fields. Fill them in with the column number, row number, and edge of the logically adjacent track section. For example, to connect the right side of the cell at column 92, row 4 with the left side of the cell at column 3, row 10, place the cursor over the cell at column 92, row 4 and select Details->Track Ends -> Joins to adjacent track on the box titled RIGHT and remove the checkmark. After removing the checkmark, click on "Column number" and type 3 in the blank field. Then click on "Row number" and type 10 in the blank field. Move the cursor over to the "Edge" box and select "Left". Finally, click on "Accept". The two cells are joined and you can insert or delete cells between them and they remain joined. If you decide to disconnect them, you can place a checkmark in Joins to adjacent track or remove either of the cells.

If a track does not join to an adjacent track, **designer** tries to highlight the disconnect by putting a "tick" mark (short, perpendicular line) at the end that "jumps" somewhere else. See Figure 23: Flyover.

But what about those other buttons, the ones that change depending on the number of tracks terminating on the edge or the shared edge? The discussion on switch points will be deferred while we look at detection blocks. If "Block boundary" has a checkmark, then the end of the track is also the end of a detection block. This allows you to define the characteristics of the block and, optionally, add signals protecting the block. Let's begin by defining the characteristics of a detection block.

9.1 Defining Blocks

Select Details->Track Ends->Define Block and you will see another pop up window (Figure 13: Detection Block Definition). If the "Block Name" field is blank, be sure to name the block.

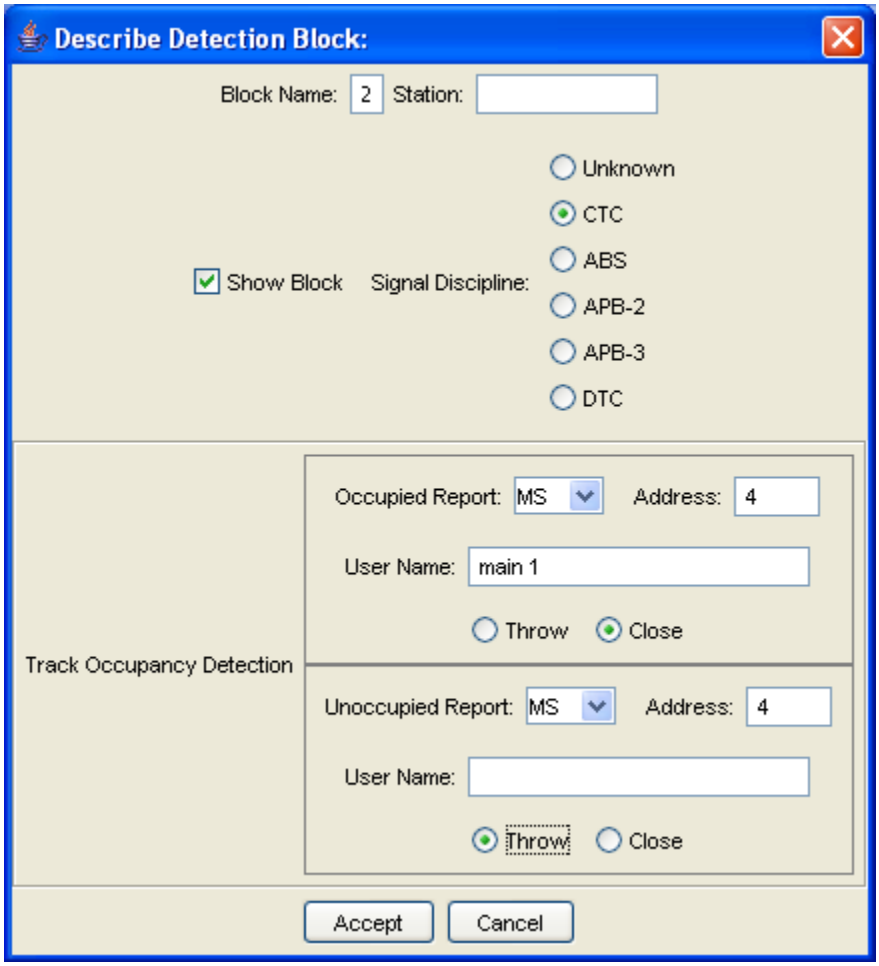


Figure 13: Detection Block Definition

9.1.1 Specifying the Control Discipline

Select one of the options for how signals protect the block:

- "Unknown" allows **CATS** to figure something out, which usually means it complains and does not show any track sections in the block, so it is best to pick another option.
- "CTC" selects "Centralized Traffic Control" (also called "Train Control System" or "TCS"). This means that signals protecting the block by default show "stop". If the block is safe to enter (meaning that it is unoccupied and there are no turnouts aligned against traffic entering the block from the end a signal is on and the dispatcher has not granted local switching or taken the block out of service) and the dispatcher sets a signal, then one signal will go "non-stop". All other signals protecting the block remain "stop" because the dispatcher is allowing a train to move through the block in one direction only.

- "ABS" selects "Automatic Block Signaling". This is a simple protection scheme which works best when all traffic is in one direction, to prevent a train from running into the back of a train in front of it. The basic concept is to keep traffic moving, providing a "stop" indication only if an unsafe condition exists. Thus, all signals protecting a block[13] that is occupied are "stop"; otherwise, an effort is made to show "non-stop". The signals protecting traffic approaching the block from adjacent blocks show "approach" (assuming any is unoccupied) and the signal protecting the next block away (in the direction of travel) is "clear". Thus, there is a block buffer around any occupied block with a warning that a train should slow down because the next signal it sees may be stop (giving the engineer some time to stop)[14]. Note that ABS does not consider the direction of travel, only the location of occupied blocks (and turnout alignment, local switching, maintenance, etc.).
- "APB-2" selects "Absolute Permissive Blocks, with a two block buffer". This is like ABS, but takes into consideration which direction the train occupying a block has been moving. APB is better suited than ABS is for single track operation with trains traveling both directions. The reason is that if there are several ABS detection blocks between passing sidings, the signals will allow two trains to enter the single track from opposite ends, simultaneously, with the result that they will eventually end up nose to nose and no where for either to go. Under ABS rules, an "approach" signal just says that the protected block can be entered, but the next cannot. It does not say that there is a train approaching. So APB works by noting that when a train enters single track, then all signals protecting the single track in the opposite direction turn to "stop". The signals in the direction of travel behave as ABS, but the signals in the opposite direction behave as though a dispatcher set a direction of travel against them.

  The "two block buffer" means that a yellow signal precedes the red in front of the train, in the opposing direction and a green precedes the yellow, so there are two blocks of warning in front of the train for opposing trains to stop. However, this is often not enough and an opposing train may still see a green signal followed by a red.
- "APB-3" selects "Absolute Permissive Blocks, with a three block buffer". This variation on APB extends the buffer in front of a train from 2 blocks to three blocks and ensures that an opposing train will see at least one yellow between the green and red signals. APB-3 has not been implemented and behaves the same as APB-2.
- "DTC" selects "Direct Train Control". This is actually a misnomer because the prototype uses DTC without signals. This selection can eliminate some paperwork on sections of layouts without visible signals. Often the dispatcher will directly control trains by telling the train crew

[13] Here, again, a signal can protect more than one detection block. It protects all blocks encountered between it and the next signal.
[14] However, see the discussion on "Advance" indications.

between which points on the layout the train is authorized to operate within. For example, "Train 100, you are cleared from station A to station B". If the track from station A to station B is defined with multiple blocks and defined to use DTC, the blocks behave like CTC – the dispatcher reserves the route from A to B. As the train traverses the blocks, the detectors fire (assuming detectors have been defined and are working) and the occupied blocks light up. However, unlike CTC, when the detectors release, the blocks do not return to "idle" but turn yellow. This "bread crumb trail" is to remind the dispatcher that those blocks cannot be given to another train until they have been taken away from the train they were granted to.

Thus, the largest difference in "Control discipline" is how much control the dispatcher has over signals. With CTC and DTC, no signal is "non-stop" without the dispatcher doing something. With ABS or APB, the signals set themselves in response to the trains.

Which discipline should you use and where? We usually set up the layout for APB during open houses, when the dispatcher (if we have one) cannot devote as much attention controlling trains. We use the same track plan, but use CTC for operating sessions. We have a few sections of ABS (on the transition from classification yard to main). There is a subtle distinction between CTC and APB. CTC allows trains to run only if the dispatcher has created a route. APB tries to let trains run unless it would not be safe. Thus, CTC signals tend to be "stop". APB signals tend to be "clear". Prototype signaling tends to use "automatic" signals between interlocking plants (of which an OS section is the most basic). You make these CTC or APB. With CTC the "intermediate" signals will tend to be "stop" until the dispatcher sets a route. With APB, the "intermediate" signals will reflect track conditions. When the dispatcher sets a route, it will appear on the panel and the opposing signals will tumble down to stop.

9.1.2 Hidden, Dark, and Normal Blocks

CATS recognizes three kinds of track: normal, hidden, and dark. Dark track shows up on the dispatcher panel, but does not have an occupancy detector associated with it; thus, it does not change colors as trains pass through it. Hidden track is defined in **designer**, but is not painted on the **CATS** dispatcher panel. Normal track has both detection and is painted. The "Track Occupancy Detection" area of the pop up window (Figure 12) is used for telling **CATS** what detectors are associated with which blocks. If that information is not filled in, then the track is treated as "dark". The "Show Block" checkbox is used for hiding track, so it is not painted by **CATS**. Use hidden track for track that is controlled by the computer, but not to be seen by the dispatcher (e.g. staging). An option is to simply not include hidden track on the diagram. However, if you want the computer to control turnouts on track the dispatcher does not see (for example) then define the track in **designer** and uncheck the show block checkbox. Another case to include hidden track is when you have track that you want to affect visible

track. For example, you want to put a signal on the staging track lead. So, you put a detector and signal on the lead, then define the lead as using "ABS" signaling and mark it as hidden. Something you probably do not want to do is hide CTC track because the dispatcher could never set the signals.

Designer colors the track based upon if detectors are defined or not. If the track is in a named Block and has both detectors defined, **designer** will draw it in the *Block Empty* color. If the track is in a named Block and does not have both detectors defined, **designer** will draw it in the *Dark* color. Finally, if the track is not in a named Block, **designer** will draw it in the *ERROR* color (see Section 14.1.1 for color definitions).

9.1.3 Defining Track Detectors (Occupancy)

The bottom two panes on the block definition pane are used for defining the occupancy detectors. As, mentioned above, if these are not specified, the block is treated as "dark territory". There is one pane for defining the "occupied" event and one for defining the "vacated" event. Each pane has three components. See 13.1 for what how these three components are used.

9.1.4 Station

The Station field is used to tell CATS that several Blocks form some location on the railroad. For example, the tracks in staging yard 1 may represent Argentine Yard, so the "Station" field in all Blocks (each staging track) could have the same entry, such as "A Yard". This field is completely optional; however, if a **Train Status Client** runs during an operating session, the location of trains will only be reported when they are in blocks with the "Station" field defined. Similarly, if you want CATS to interwork with JMRI Operations, then the "Station" field should be the same as the corresponding "Location" in Operations (see Section 16 for more details).

9.2 Adding Decorations

This document uses the term "decorations" to mean optional things (such as signals and depots) added to a section of track.

9.2.1 Adding, Changing, and Removing Signals

All changes to signals are made through Details->Define Signals. It generates a pop up window.

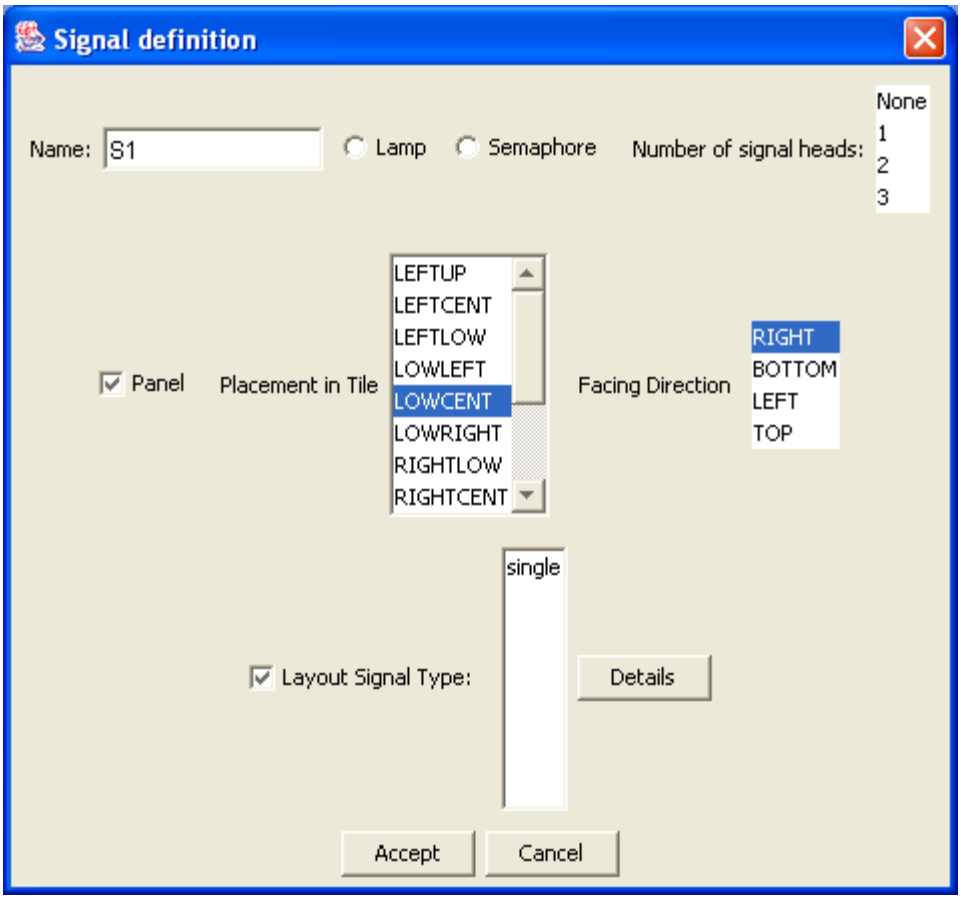


Figure 14: Signal Definition Pop-up

At the bottom of the window is a list of known signal templates (see section 8). Select the template that describes the signal protecting access to the track segment from the associated grid edge. If there is a signal on the layout, check Layout Signal Type. This will allow you to select one of the templates. Use the Details button to pop up another window for defining the decoder addresses for the signal (see below). The upper half of the window is used to tell **CATS** how to draw the icon for the signal on the dispatcher panel. To place a signal icon on the dispatcher panel, check Panel. If a signal template is selected, then the program makes the icon conform to the kind of signal and number of heads. If a signal template is not selected, then you can fill in the kind and number of heads manually. You can name the signal in the Name field. This is optional, though I recommend that you name each signal with a unique name. Eventually, I would like to use the contents of this field to label the signal on the dispatcher panel. **CATS** will use the signal name in generating the JMRI system name of each SignalHead. It will also appear in traces. So, the signal name is useful in troubleshooting.

There are four possible settings:

Layout Signal Type	Panel	Signal Type
checked	checked	Control Point

checked	unchecked	Intermediate
unchecked	checked	Place Holder
unchecked	unchecked	Virtual

Table 6: Signal Types

Control Point A Control Point is an icon on the dispatcher panel associated with a signal on the layout. It exists to provide the dispatcher with a place to stop and hold a train. Conversely, it provides the dispatcher with a place to indicate that a train is permitted to proceed. A Control Point corresponds to an absolute signal. When a dispatcher creates a route, the route propagates from one Control Point to the next and no farther. In APB, a route (created when a train enters a block) also propagates from one Control Point to the next. In addition, **CATS** automatically promotes any signal protecting an APB interlocking plant (e.g. OS section) to a Control Point.

Intermediate An Intermediate signal exists on the layout, but not on the dispatcher panel. It is used to provide spacing between trains traveling in the same direction. It is usually a permissive signal (i.e. a train may stop then go or proceed slowly through a halt indication). An intermediate signal will propagate a route, if conditions are safe to do so. If there is an unsafe condition, the signal will show halt, but change to something less restrictive when the unsafe condition is removed. If an Intermediate signal has a route leading into it, it will propagate the route as soon as any blocking conditions are resolved.

Place Holder A Place Holder does not exist on the prototype because if there is no signal in the field, there should be no signal on the dispatcher panel. However, for completeness, it is allowed in **CATS**. It is actually useful on a model railroad to provide a way to break up a staging track into multiple sections to accommodate multiple trains.

Virtual All block boundaries have signals associated with them, even if neither Layout Signal Type nor Panel is checked. These formless signals record the status of the detection block that they protect and relay that status to the signals that precede them (the signal a train encounters prior to the Virtual signal). Thus, a "real" signal merges the status of the detection block it is protecting with the status of any Virtual signal that follows it. The "real" signal may protect multiple blocks.

If Layout Signal Type is not checked, then the signal symbols will be in a different color on **CATS**, to remind the dispatcher that the engineer cannot see a signal.

The middle fields are used for positioning the icon on the dispatcher panel. The Panel checkbox must be checked for the icon to appear on the panel. After it is checked, the lists next to it are enabled and an entry can be selected from each. Use Placement in tile to control where the icon is placed in its grid. The entries beginning with "Left" are used to position it along the left edge. The entries

beginning with “Low” are used to position it along the bottom, etc. Use Facing Direction to control the rotation of the base of the icon. So, make some selections. Look at the icon on the **designer** screen, then change them, as you desire. It is best to be consistent. Getting the orientation correct takes a little experience. It is best to think of a signal as a gatekeeper. It does not control a train from leaving a block, but controls a train from entering a block. So, it should be “facing” out from the block.

The base of a signal can be drawn in two different ways. The default is a triangle. The apex of the triangle (the corner touching the circles) is a handy reference for reminding dispatchers that the signal is for a train moving in that direction. It can be considered the tip of an arrow head. Some prototype displays use an inverted “tee’ to represent the base. You can select either. Under the Appearance menu is a checkbox labeled *Tee Base*. When checked, the signal bases will use the latter representation. The option selection takes affect immediately in **CATS** (i.e. all signals are changed when the option changes), but only affects signal icons created after it is changed in **designer**. This difference in behavior is due to how the icons are created in the two programs.

The first screen capture shows triangle bases and the second shows the inverted “tee” base.

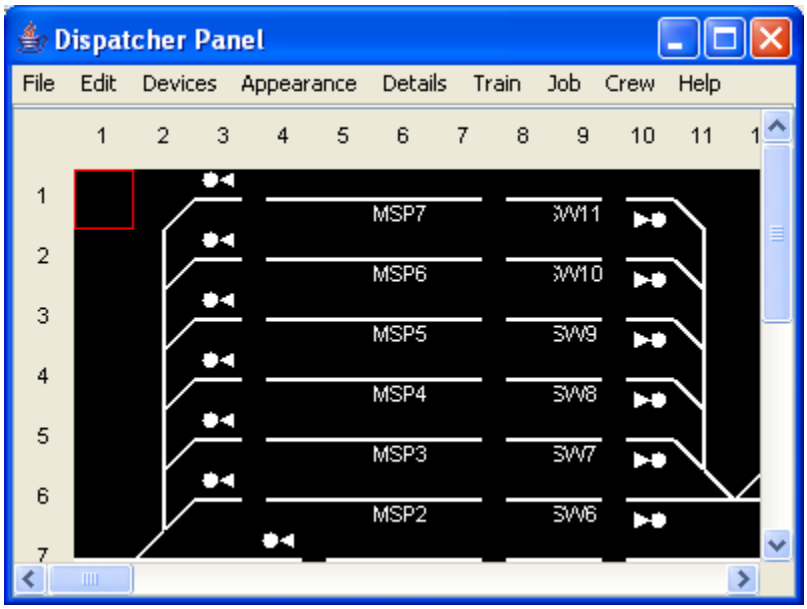


Figure 15: Signals with Triangular Bases

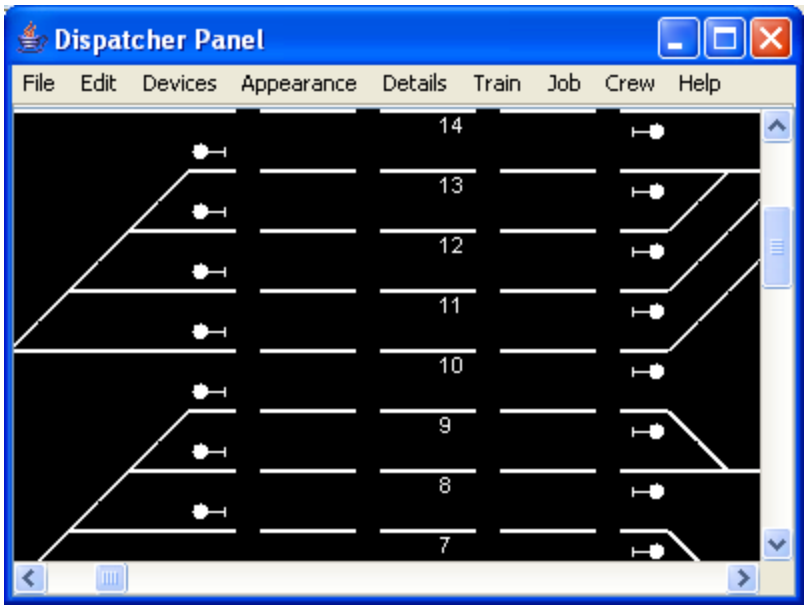


Figure 16: Signals with Tee Bases

Finally, intermediate signals appear as a filled in circle on the end of the track protected by the signal.

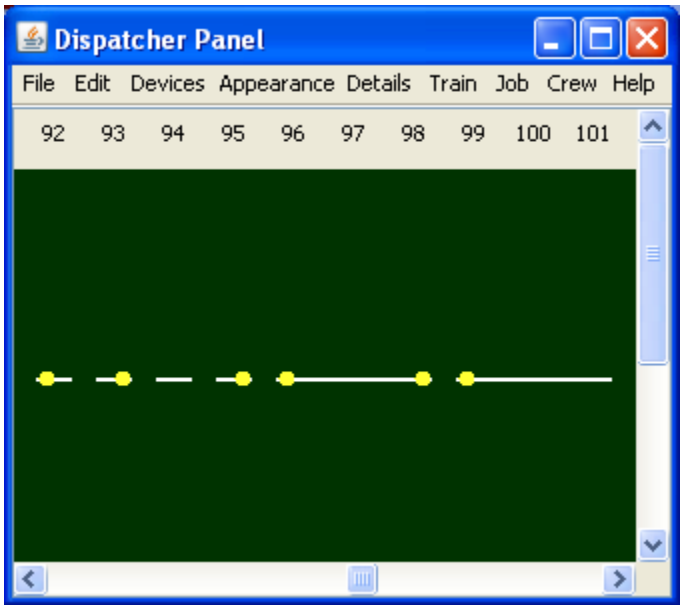


Figure 17: Representation of Intermediate Signals

Let's go back to the Details button. When selected, you will see another pop up window. This one has a tab for each head.

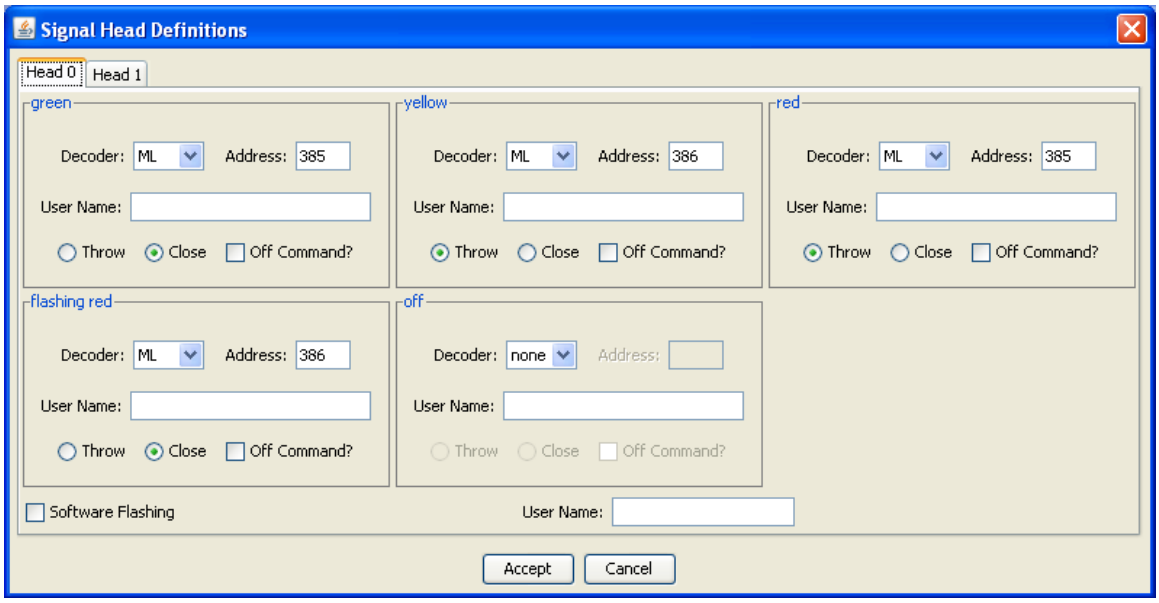


Figure 18: Signal Head Decoder Association

Each head has a list of colors or positions for each of the portions of an aspect that it is required to present (see 8.1). If the layout electronics can handle that color or position, pick the JMRI name that describes the device driving the signal. Then, you can fill in the decoder address that causes the electronics to show that color or position. The program assumes that the decoder has two positions – close and throw (or open). If the light is turned off by sending a command with the opposite position, you should check the "Off Command?" box. If that box is checked, **CATS** will send the "other" command to turn the light off before sending a new command to set the head to a different color. If the box is not checked, **CATS** assumes that the next command changes the color automatically. For example, the "Off Command?" box probably should not be checked when working with 2 lead, bi-color LEDs. A general rule is that if the colors have unique addresses, check the "Off Command?" box. The Digitrax SE8C can show four presentations with two decoder addresses. The "Off Command?" box should not be checked for an SE8C.

The "off" pane appears if an aspect is defined as "off" (the light is dark) or the signal has approach lighting or an aspect is "flashing". If there is a flashing aspect and your hardware does the flashing, leave the "Software Flashing" box unchecked (e.g. SE8C). If you want **CATS** to control the on and off for flashing, check the "Software Flashing" box. In addition, either the decoder must have an "off" address (defined in the "off" box) or the light can be turned off by sending the opposite command as the one that turns the light on (the "Off Command?" box is checked).

If you do not define a decoder for a particular color or position, then **CATS** will not send anything to the layout.

See also Section 14.7 for tips on using JMRI SignalHeads and exposing **CATS** SignalHead definitions to JMRI.

To change an icon or decoder definition, just make the changes in the windows. Be aware that large changes (such as changing the template) may not retain all the information from the previous setting. So, it is a good idea to have the decoder information at hand when making changes.

To remove a signal on the CTC panel, uncheck Panel. To remove a signal on the layout, uncheck Layout Signal Type or change the JMRI name to "none".

9.2.2 Defining Stations and Depots

Selecting Details->Station brings up a pop up window for placing a depot or station icon on the dispatcher panel. There is a checkbox labeled Station, a selection list for where to place the station icon in the grid cell, and a selection list of color names (see 14.1.1). A station has no affect on operating the dispatcher panel. It is intended to remind the dispatcher where the depot is located. The icon appears as a solid box on the dispatcher panel, in the color indicated by the selection list on the right.

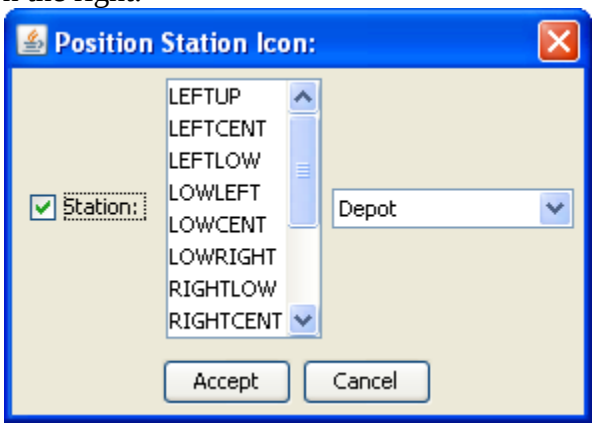


Figure 19: Station Icon

To remove a station, uncheck the Station checkbox.

9.2.3 Giving the Track Section a Name

If you select the Details->Name item, you will see a pop up window that lets you add a label to the grid cell, which appears in that cell on the dispatcher panel. This provides the ability to place a name on a siding or station, or territory. The pop up window has a selection list for where to locate the name in the cell (do not be concerned if it is clipped by the sides of the cell – it will not be clipped by **CATS**), a text field for the name, and a selection list of font names (see 14.1.2).

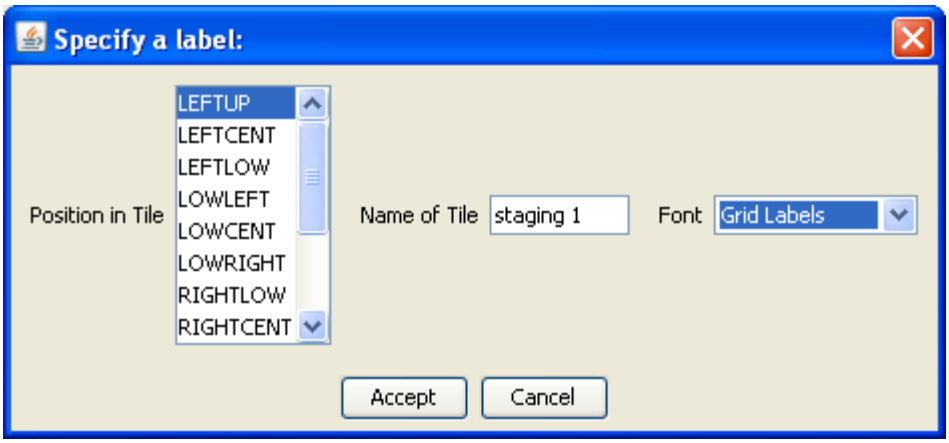


Figure 21 Cell Name

To remove a name, erase all characters out of the text field.

9.2.4 Adding a Picture

The Details->Picture option brings up a pop up window, which lets you select a gif image to appear on the dispatcher panel. Check the Select Files check box to indicate that you want to add a picture (uncheck it to remove a picture). You can use the browse button to bring up a standard file search window. So, select the file you want or enter its name directly into the text field. If you use the browse option, then the full path to the file will be saved in the XML file, which makes moving the XML file to a different computer a little tricky.

As with track section names, do not be concerned that the image does not fit in a cell. It can spill over to other cells in **CATS**.

Why would you want to add an image to the dispatcher panel? You might want to put the railroad logo on it.

9.3 Defining Switch Points and Turnouts

Details->Define Switch Points pops up the most complicated window in the **designer** program. It is used for describing switch points.

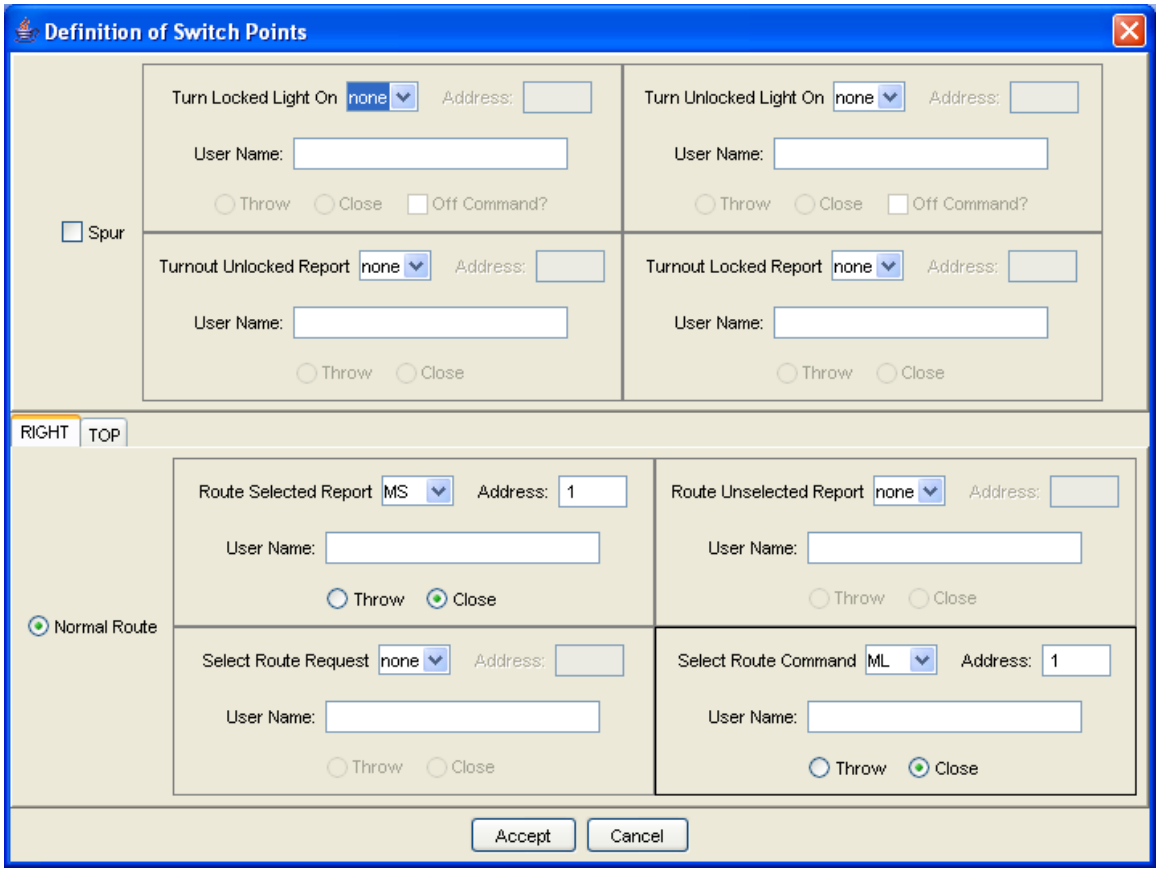


Figure 22: Defining Switch Points

Note that the window is split horizontally into a two halves. The top half contains information that applies to the switch points as a single entity. The bottom half is tabbed, with a tab window for each route. Even if there is nothing on the layout corresponding to the switch points, be sure to pay attention to two things. First, on the top half, there is a *spur* checkbox. It is used to tell **CATS** that the dispatcher has control over the turnout or not. If checked, then the turnout is not under dispatcher control. If not checked, then the turnout is an OS section and the dispatcher can close or throw the points. **Second, be sure to check the *normal route* box in one of the tabbed windows**. It should be the window which describes the "main" route through the turnout. **CATS** does not work right (it crashes) if a switch points does not have a normal route checked. **Designer** assists in telling you that some switch points are missing a normal route designation. It will draw the tracks composing the points in the *ERROR* color (Section 14.1.1).

9.3.1 Definitions in Common to All Routes

As noted, the top half of the window contains things not specific to a route. There are four decoder definitions:

1. *Turn Locked Light On* If there is something on the layout to tell the switching crew that they do not have local control over the turnout, then fill in the decoder information. When the dispatcher "locks" the turnout from local control, then **CATS** will send a message to the decoder using the information. If "Off Command?" is checked, **CATS** will send the "other" command when the dispatcher unlocks the turnout.
2. *Turn Unlocked Light On* If there is something on the layout to tell the switching crew that they have local control over the turnout, then fill in the decoder information. When the dispatcher "unlocks" the turnout, **CATS** will send a message to the decoder using the information. This may appear to be redundant with the item for turning off the locked light, and in many cases it is; however, if the locked and unlocked lights are not controlled by the same decoder, then two commands are needed. **CATS** will send both, if they are defined.
3. *Turnout Unlocked Report* Some model railroads imitate the prototype by requiring that the local crew unlock the turnout before they can move the points. If your railroad does this, then this is the message sent by the decoder to tell the computer that the crew has unlocked the turnout. This does not do anything in this version of the program.
4. *Turnout Locked Report* This is used for telling **CATS** how to identify the turnout control being restored to the locked position.

9.3.2 Defining Routes

There is one tabbed pane in the bottom of the window for each track segment selected in *Details->Track*. It has the *Normal Route* checkbox and decoder information about the route. **Designer** will guarantee that no more than one route has the *Normal Route* box checked, but **it will not guarantee that one has it checked, so be sure to check one**. **Designer** will draw any switch points without a *Normal Route* box checked in color *ERROR*.

Let's digress for a moment into the realm of Turnout feedback. "Turnout feedback" is the layout reporting to **CATS** how the points are aligned. There tend to be three common schemes on model railroads for turnout feedback:

- None. **CATS** sends a command to the switch machine and assumes that it works.
- Positive. One alignment has a sensor on it. When the turnout is set to that route, the sensor reports contact. When the points move out of that route, the sensor reports loss of contact. **CATS** will assume that when contact is reported, the points are in the indicated alignment. **CATS** will assume that when contact is not reported, the points are set to the non-instrumented route. Note that this may not be completely accurate

because if the points could get stuck between routes, **CATS** will assume they are in the non-instrumented position.

- Exact. Each route has a sensor on it.

The decoder information is for both sensing the turnout and controlling it.

1. *Select Route Request* is sensed by **CATS**. It should be the decoder address of the switch that the crew uses to request that the computer move the turnout points to the enclosing route. So, when **CATS** receives the request indicated, **CATS** will look at the layout and determine if the turnout can be moved to the enclosing route. **CATS** will look at the lock condition of the turnout, the occupancy of the block containing the turnout, traffic reservations through the turnout, the signal discipline on the block, etc. and if it is safe, **CATS** will send the commands to move the points. The source of the request can be a fascia switch wired to a decoder or from a throttle or either.

   Some model railroads are wired with a momentary (non-latching) push button. The procedure is that the crew moves a knob (which changes the power routing), then pushes the push button. Thus, the push button is used in selecting any route and always sends the same decoder address. **CATS** understands this situation. If your layout is wired this way, put the same address in each route. **CATS** will recognize that the addresses are the same and set a different route every time it sees one of the decoder messages.
2. *Select Route Command* is the message(s) **CATS** sends to the layout to move the points to the enclosing route. The Decoder Chain is useful here for throwing multiple turnouts.
3. *Route Selected Report* is the decoder information sent by the layout when the turnout has been moved to the enclosing route. This applies only if your layout is wired for either "positive or exact position reporting" and you want **CATS** to know when the turnout has reached the requested alignment.
4. *Route Unselected Report* is the decoder information sent by the layout when the turnout has moved away from the enclosing route. This applies only if your layout is wired for "exact position reporting" and you want **CATS** to know when the turnout is not in the alignment of the enclosing route.

   You do not need to have decoders for all of this. If you leave out decoder information **CATS** will simply ignore it. Here is how the above is applied:
   - No feedback. Leave *Route Selected Report* and *Route Unselected Report* blank.
   - Positive feedback. Populate *Route Selected Report* on each of the routes. They will probably be the same decoder address, but one entry will be thrown and the other closed.

- Exact feedback. Populate *Route Selected Report* and *Route Unselected Report* on all routes. The decoder addresses will be probably be different.

To summarize how turnout position feedback is specified, if only one of *Route Selected Report* or *Route Unselected Report* is defined, then the turnout position on the dispatcher panel will not change until the report is received. If both are defined, then the turnout will be in a transition state when *Route Unselected Report* is received and completed when *Route Selected Report* is received. If neither is defined, then **CATS** treats the request to move the turnouts as always happening instantly.

If the layout uses ABS or APB, then at least one of *Route Selected Report* or *Route Unselected Report* should be defined (and the turnout should be wired for feedback). Without a dispatcher, this is the only way for **CATS** to know which way a turnout is set (which is a factor in determining signal indications).

There is an item under Appearance that affects turnouts. If the *Lock Turnout Decoders* checkbox is checked, **CATS** (this option only applies to **CATS**) will remember which turnouts are locked (e.g. occupied or are involved in a route reservation). It will disallow any turnout to be thrown that will send commands that will move a locked turnout. For example, it is common for the turnouts on a crossover to share the same decoder addresses, so that when one moves, the other moves. This checkbox will prevent that from happening if one of the turnouts is locked. If your layout does not share decoder addresses, then leave this box unchecked because **CATS** will run a little faster.

9.3.2.1 Enforcing Dispatcher Control of Turnouts

As you work out the details of operating the railroad, you will encounter the question of protocol for who controls a turnout – the dispatcher or the local crew – and when do they relinquish control. Some situations are easy. Turnouts defined in **designer** as spurs are always under the control of the local crew. The dispatcher has no way to tell **CATS** to move them. On the prototype, these are often called manual switches. **CATS** allows the dispatcher to move points that are not defined as spurs. On the prototype, these are often called automatic switches. However, by design or not, the local crew may also have the capability to move the points as well. For example, **CATS** sends a message down the command bus to move the points. The local crew may have the ability to send the same command via the throttle (cab). Alternatively, the layout may have a button on the fascia for moving the points under local control, which can be triggered at any time. Thus, the turnout can have two "owners", which can lead to confusion, and the layout will require a protocol for deciding who "owns" the turnout at any time.

The general rule for automatic switches on the prototype is that "the dispatcher owns the turnout until granting control to a local crew". It is often possible for a local crew to usurp control of a turnout, but such actions are detected and the crew disciplined. Many layout owners want to emulate the prototype – "the dispatcher owns the turnout until granting control to a local crew". However, the consequences are not as dire if the local crew takes control of a turnout and it is often much easier to do so. Thus, it is tempting for a local crew to think, "I can finish my work if I can just cross the main and since nothing is coming, I will set the points, make my move, and return the points without getting permission from the dispatcher". Some layout owners see this as not worth preventing; other owners see it as not at all in the spirit of simulating the prototype. The rest of this section is for the benefit of the latter.

There are multiple ways to set up a layout and **CATS** to prevent local crews from unauthorized movement of turnouts. They all hinge upon the dispatcher granting track authority on a block of track.

- Mechanical Lock. When the dispatcher grants track authority, **CATS** will turn the *Unlocked Light On*. In doing so, **CATS** can send one or more commands. The recipients of those commands can unlatch a mechanical lock on the turnout; thus, activating the fascia button. This method relies on additional hardware to provide the mechanical lock and a decoder that can be set from the computer to activate and release the lock. It may also require a fall back mechanism so that the lock can be released when **CATS** is not running.
- Software Lock. The layout can be wired so that the fascia button does not actually move the turnout points, but sends a message to **CATS** (*Select Route Request*). **CATS** will look at the situation and move the points on behalf of the local crew, if the dispatcher has granted track authority. This method requires that the push button be tied into a decoder that can send a message to **CATS**. It also mandates that **CATS** be running to move the turnout locally.
- Countermand Lock. **CATS** can be set up (see Section 14.1.15) so that when it sees a turnout move without authorization to send the command to move it back. This method requires both feedback (*Select Route Report*) and a command (*Select Route Command*), but nothing additional. It works on both spur and non-spur turnouts. Depending on the control system and feedback, the points may move, then immediately move back, which is disconcerting, but eventually the crew will understand that they really do not have control of the turnout.

9.3.3 Tracks Revisited – Puzzle Tracks

Puzzle tracks are complicated sets of points that baffle novices (and some veterans). Double and single slips are the primary examples. This section discusses how to create crossings, flyovers, double slips and single slips.

The following diagrams illustrate each. The first example is a fly over. The top track logically connects to the bottom track, even if the drawing does not. It uses the "not connected to adjacent track" feature described earlier to join the vertical segments (Section 9). This configuration would be used for the intersection of a figure 8 or any place a track tunnels under another.

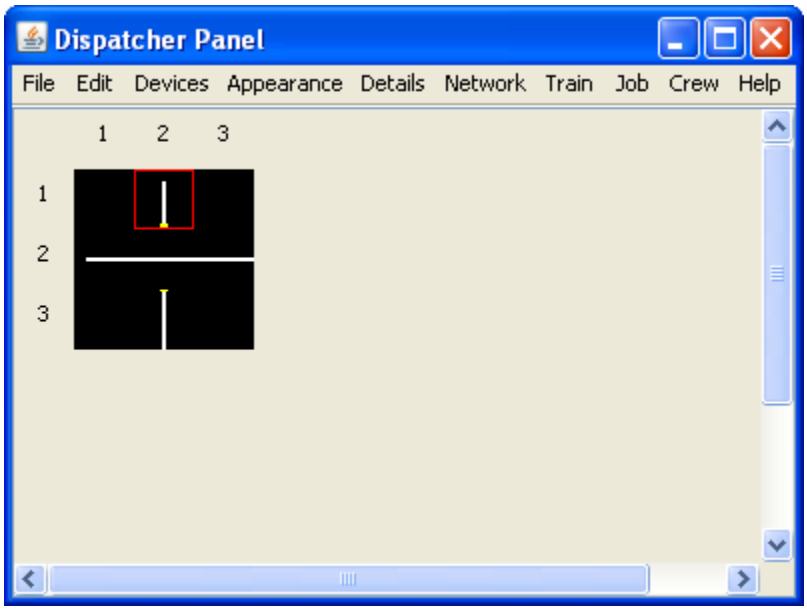


Figure 23: Flyover

The next diagram illustrates one form of a crossing. The tracks cross each other at grade, but there is no route from the vertical track to the horizontal and vice versa.

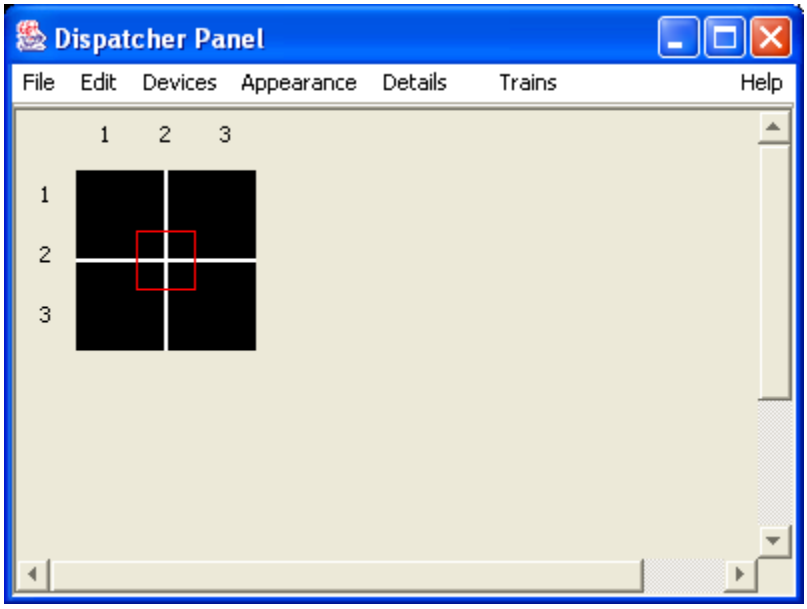


Figure 24: Crossing

Prior to version 2.01, all the tracks shown are in the same detection block. After version 2.01, the tracks are in separate detection blocks. Thus, one track could be CTC and the perpendicular track could be ABS. Furthermore, when a route is set on one of the tracks, signals protecting the crossing track drop to *stop*. This is similar to a turnout being thrown to a fouling position.

Other forms of crossings are the following (grid lines are included to show Section edges):

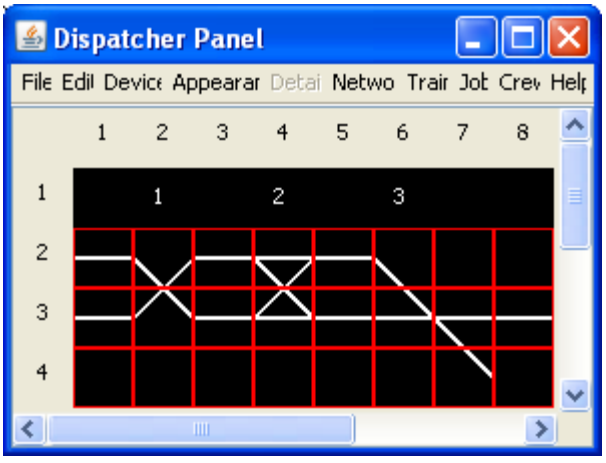


Figure 25 Crossings on Section Edges

Column 1 illustrates a 90° crossing. This is more compact when two parallel tracks cross than bending tracks as for Figure 24: Crossing. Column 2 illustrates

a “scissors” or double cross over[15]. It is column 1 with the parallel tracks continued. Column 3 illustrates a crossing on an oblique angle.

Just as the first form of crossing is ambiguous, so are crossings on an edge. Specifically, they look like switch points. To distinguish between crossings and switch points, **designer** considers tracks that meet on an edge to form a possible crossing only if

- There are only two tracks touching on the edge
- Both Sections that touch (halves of the crossing) have only two tracks
- The tracks that touch the edges are complements (UpperSlash/LowerSlash, UpperBackSlash/LowerBackSlash, Horizontal/Horizontal, Vertical/Vertical)

Any construction that does not meet those criteria cannot be a crossing. Nonetheless, the constructs that meet the above criteria could also be switch points that form a double slip. When **designer** detects this ambiguity it requests clarification on the Edge Specification panel (Figure 26 Crossing or Switch Points Selector Pane).

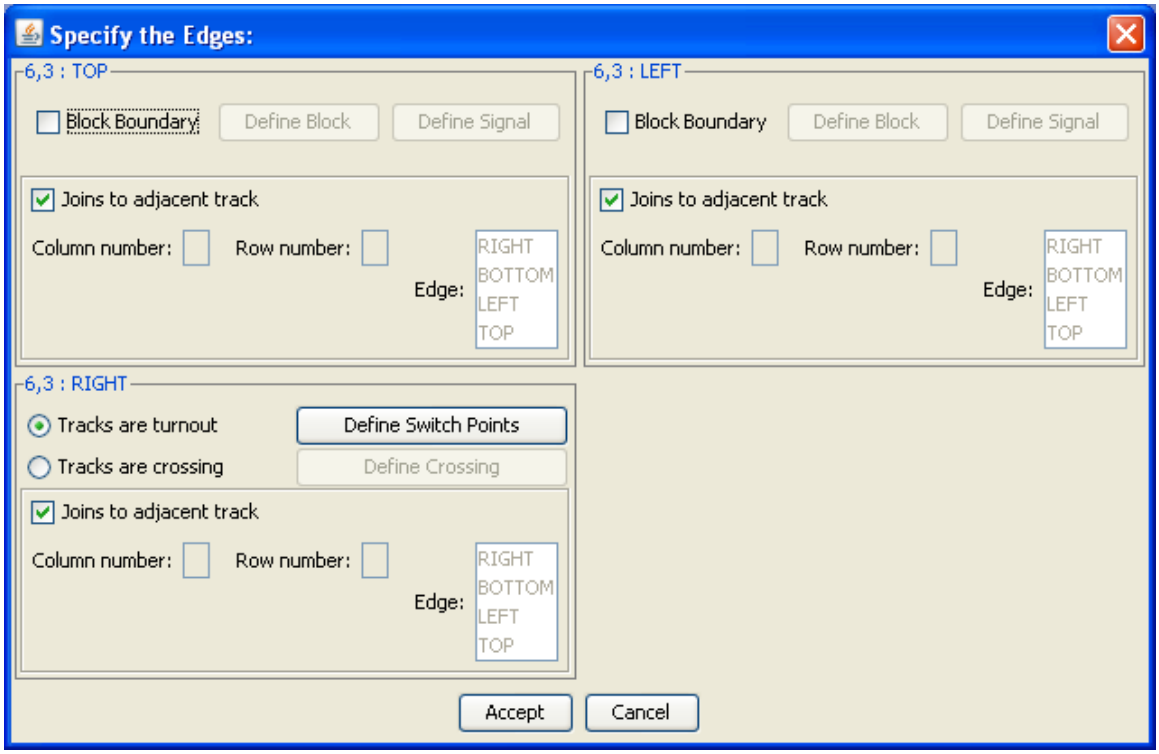


Figure 26 Crossing or Switch Points Selector Pane

The quadrant labeled RIGHT illustrates the selector. It has two radio buttons (*Tracks are turnout* and *Tracks are crossing*). Clicking on one activates the button to the right (*Define Switch Points* and *Define Crossing*) and deactivates the

[15] Note that a double slip is a very compact scissors.

one to the right of the other radio button. Selecting *Define Switch Points* (the default) pops up the switch points definition pane (Figure 22: Defining Switch Points). Selecting *Define Crossing* pops up the crossing definition pane.

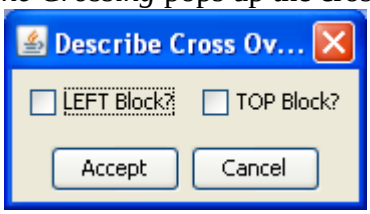


Figure 27 Crossing Definition

The Describe Cross Over window has 2 buttons. They are labeled with the Section edge that each of the tracks forming the cross over terminate on. Checking a box places a block boundary on the crossing edge for that track. Leaving it unchecked tells **designer** that the track and its continuation in the neighbor Section are in the same detection block. Notice that the tracks are independent. If they are in the same detection block they will either have to share the same detector or touch in a different Section (for example, column 2 in Figure 25 Crossings on Section Edges).

Block boundaries in crossings are treated differently than block boundaries in non-crossings:

- No gap is shown
- No signal can be placed on the panel
- No secondary pop up windows for defining the block detector or the signal information appear

For a "scissors" cross over, typically both tracks will have the *Block* boxes checked. That allows trains to pass on the parallel tracks and not interfere with each other. Some commercial section track is wired with all the tracks in the same detection block, in which case, the *Block* boxes will not be checked.

Note: watch out for adjacent crossings, such as below. In this diagram, the horizontal track crosses two tracks at an angle. The horizontal track is in one detection block, but the diagonal tracks have block ends at the crossing. The caution is to ensure that the horizontal track does not have block ends at the crossings because there is no way to define the block from the crossing. Placing a block boundary between the two will work.

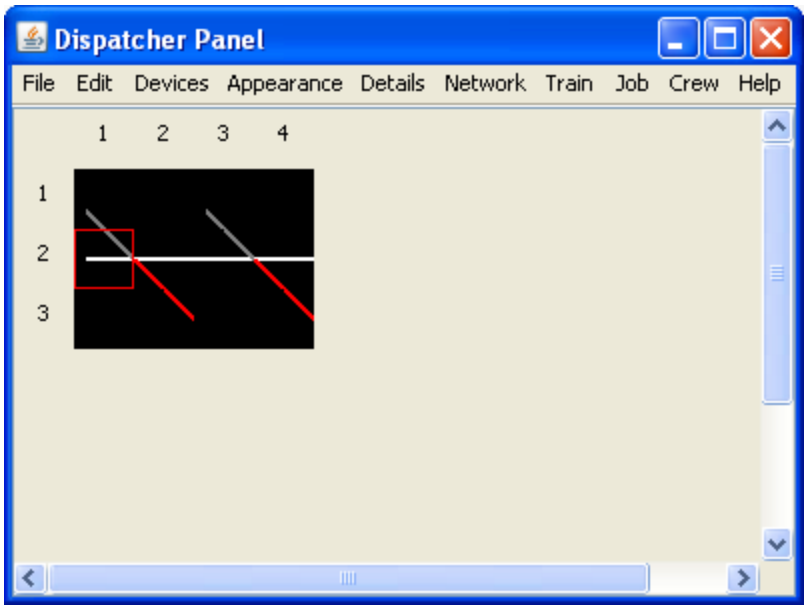


Figure 28 Multiple Crossings

A double slip is defined as a crossing with 2 parallel routes between the perpendicular legs. There are multiple ways to draw one:

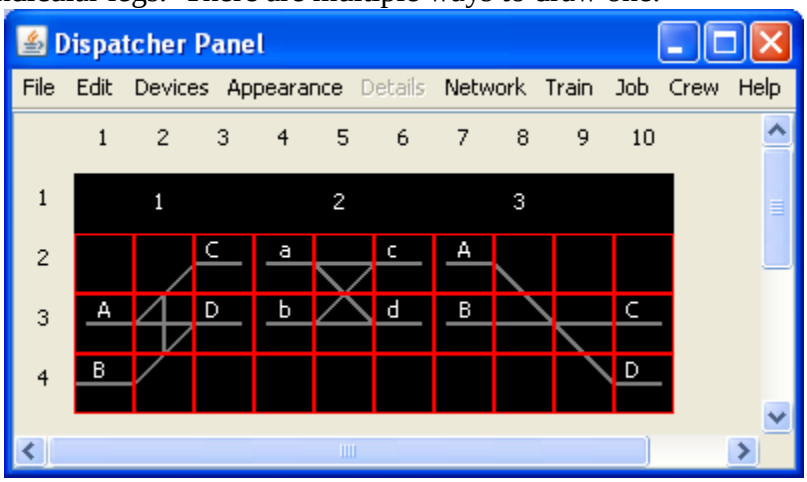


Figure 29: 3 Double Slips

There is one major problem with columns 1 and 2 in the above diagram. The dispatcher may mistakenly believe that two trains can go though the double slip on the parallel routes without colliding. In reality, that is impossible.

Column 3 is a more realistic rendition. **CATS** can pick all four possible routes, but this arrangement can easily be mistaken for a crossing, as explained above. In

this case, the *Define Switch Points* box should be checked in Figure 26 Crossing or Switch Points Selector Pane.

A single slip is a double slip that is missing one route. Here are three ways to draw one (in each case, there is no route from A to C):

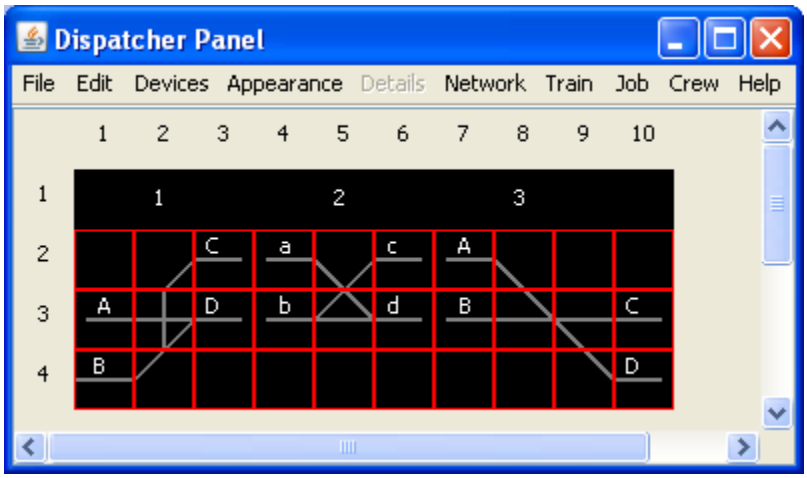


Figure 30: 3 Single Slips

Columns 1 and 2 are the straight forward construction – a double slip with one missing route. Column 3 is not an obvious single slip, but with the judicious selection of commands for setting the points, it can represent one.

Let's arbitrarily say that route A-C is not supported (consistent with the first two examples). That means the valid routes are:

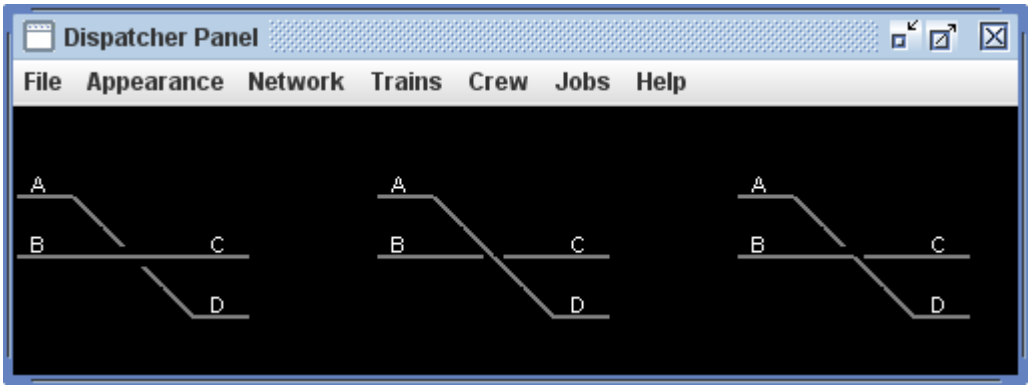


Figure 32: Possible Routes Through Single Slip

A is involved in only one route: A-D. C is involved in only one route: B-C. Thus, if the dispatcher throws the left hand turnout for A, the right hand turnout should be thrown for D. Likewise, if the dispatcher throws the right hand turnout for C, the left hand turnout should be thrown for B. By using a **CATS** chain or JMRI Route in the Select Route Command box (Section 9.3.2), this restriction can be enforced.

There are more combinations (a diamond, all six possible tracks), but given the complexity of a double slip, it is doubtful that an example exists in the prototype in a compact space.

10 Creating Trains

The Train menu item is for creating trains. You do not need to define any trains using **designer** because trains can be defined in **CATS**. However, if you have a train that is a regular in operating sessions (is not "spur of the moment"), you will save yourself some repetitive work by defining them in **designer**.

10.1 Edit Trains

This default menu is used to describe a train.

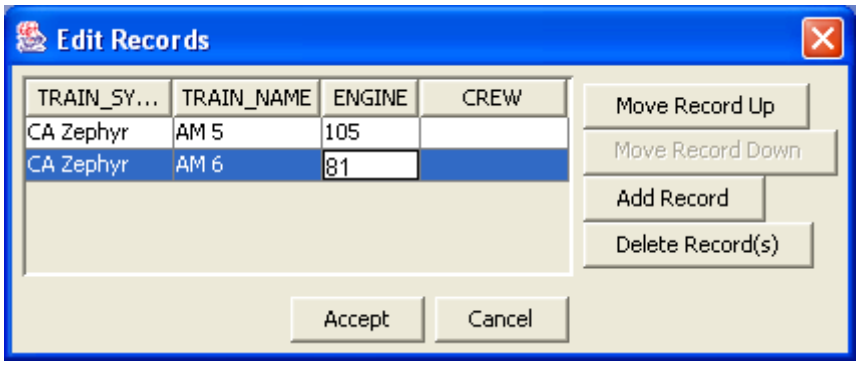


Figure 33: Train Edit Pane

The Train window can be the lineup for your operating session. Along the right are four buttons. You use *Add Record* to create a new train in the lineup. It will add a new row to the table above the currently selected row. Conversely, *Delete Record(s)* will remove the highlighted trains from the table. If you delete a train, it is not gone (but you cannot do anything with it) until you push the *Accept* button. So, if you push *Cancel*, you will close the window and leave your line up as it was prior to selecting Train. The other two buttons are used to move the selected trains up or down in the lineup. If you arrange the rows in the order the trains are to run, then you have your lineup with the dispatcher panel. The *Move Up* and *Move Down* buttons can assist you in arranging the order to run the trains in.

The Train table is where the information about the trains is stored. There are 9 columns, containing a field of information about each train.

1. *TRAIN_NAME* is the common name of the train (for example, California Zephyr).

2. *TRAIN_SYMBOL* is an abbreviation for the train. It should be short (only a few characters) and unique. It is what appears on **CATS** for marking where a train is. There may be more than one California Zephyr, but there should be only one "5" and one "6". It is also used for requesting information on a train when used in conjunction with JMRI Operations (see Section 16 for more details).
3. *ENGINE* is the lead engine number of the train. As of this release, it only provides a little additional information about the train, but we have plans for it. Eventually, we want to tie **CATS** into the Crandic "power desk" or JMRI operations (roster), so this is one link. Some of the modern prototypes label a train on the dispatcher panel by symbol or lead engine number (see 14.1.7). This field would provide the lead engine string.
4. *TRANSPONDING* is another future item. It is intended to be checked if the locomotive supports transponding. If it does, then **CATS** will use the transponding information for locating the train on the dispatcher panel. If this is not checked, then **CATS** will make "educated" guesses as to where a train is based on occupancy reports, past history, and dispatcher route selection.
5. *CABOOSE* is also a future item. If a caboose has a transponding decoder in it, then put the decoder address in this field. **CATS** will use it and occupancy reports to "connect the dots" (fill in the blocks between the engine and caboose occupancy reports). This means that the cars in between do not have to trip detectors. **CATS** will figure out which blocks are occupied.
6. *CREW* is a place holder to fill in where **CATS** places a crew selection list.
7. *ONDUTY* is the time the crew went on the job. If you want, **CATS** will record the time that the crew went on duty on the train. This can be the time the dispatcher made the assignment or it can be the time specified here. Putting a time here can reduce the number of hours the crew has left to finish the job; thus, simulating the time it took a train to arrive from "somewhere else" to get to the layout.
8. *DEPARTURE* is another time field. If you want to remind the dispatcher when a train is supposed to depart, you put the departure time here.
9. *LENGTH* is a number used to hold the length of a train. It can be populated from JMRI Operations.
10. *WEIGHT* is a number used to hold the weight (tonnage) of a train.
11. *FONT* is the final field. It is the name of a font (see Section 14.1.2). **CATS** does not use it, but the **TrainStat** application does. If you would like the **TrainStat** application to paint some trains differently (for example, passenger trains are a different color than freights and mixed are yet a third color; or east bound trains are a different color than west bound; or through trains are a different color than locals), then this field performs the task. See section 14.8 for a detailed example of how *FONT* might be used.

Now, why did the graphic screenshot above look so different? It has only 5 fields, but 9 are supported. We will discuss how to tailor the presentation of the trains screen shortly (10.4).

10.2 Save Trains

The Save Trains menu item provides a way of saving just the train information in a file. Thus, you could create a lineup for one operating session and save it. You could start fresh (or edit the previous lineup) for a different session and save it. You could save even and odd day lineups, or AM and PM lineups.

10.3 Load Trains

The Load Trains menu item is the compliment to Save Trains. It is used to read a lineup into **designer**. The file read in must have been created by Save Trains.

10.4 Edit Train Fields (or Crew Fields or Job Fields)

This menu item lets you customize the Train screen. It looks like

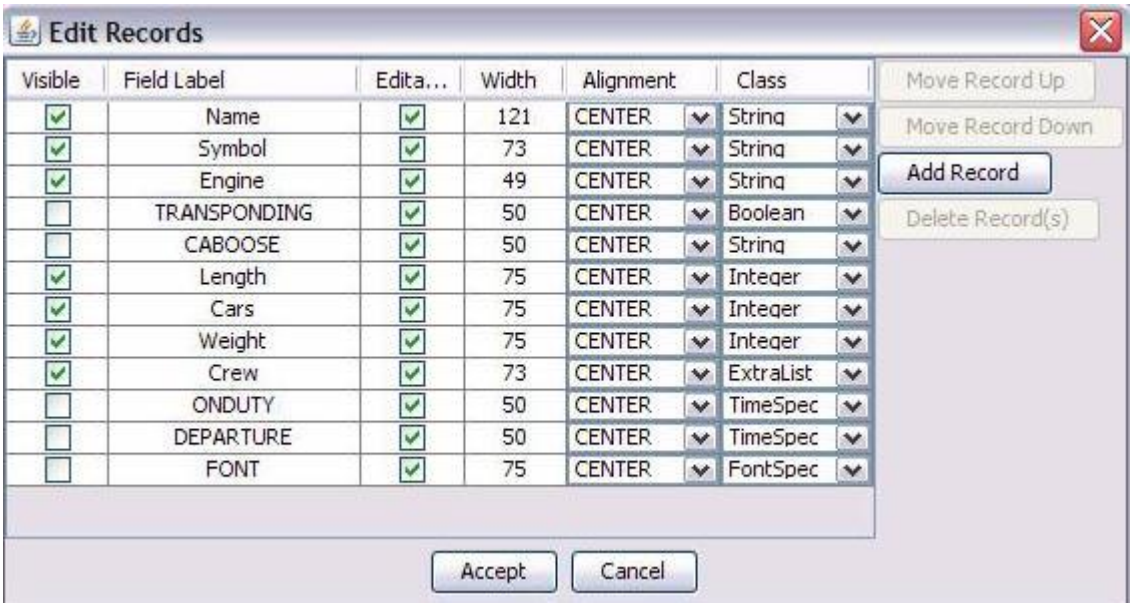


Figure 34 Train Edit Pane

This screen should look a lot like the Train screen because it uses a lot of the same code. This table shows the fields (columns in the Trains screen) a train can have as rows – one row per field. Each field can have 8 characteristics, which are the columns of this screen. Another way of saying this is that the train fields are records in this table and the train field characteristics are fields in this table.

Anyway, let's look at the buttons on the right, first. Notice that some are grayed out. They are active, depending upon which and how many rows have been selected. They allow you to

- Move a group of records up. This changes the order of the train fields on the Train screen, as they march across the screen.
- Move a group of records down. This pushes their columns on the Train screen to the right.
- Add a record. This lets you create your own train fields. The record is created above the top selected record, but if it does not go where you want, just use the move up and move down button to position it.
- Delete a record. **CATS** expects that the default fields exist. So, you cannot delete any of the rows shown here. You can add rows and delete the ones you create.

Next, let's look at the characteristics of each train field.

- The "Mandatory" column tells you if you can delete the train field or not. If mandatory is checked, you cannot delete it. **Designer** will not let you check or uncheck anything in this column.
- The "Field Name" column is how **CATS** identifies the contents of the field. **Designer** will not let you change it, either. Fields that **CATS** needs have some meaningful name. Fields that you create are "f" and a number. These names never appear on a screen in **CATS**, so there really isn't any need to customize the names.
- The "Visible" column is a column of checkboxes. If the box in a row is checked, then the field shows up as a column on the Train screen. If the box is not checked, then the column does not show up, so this provides a way to hide things that are not important to your operations. Compare the rows with the boxes checked above and you will see that they are the same as columns in the Train screen.
- The "Field Label" column is the string that appears on the top row of the column for the field on the Train screen. Thus, if it is not customary for your modeling to call the time at which the clock starts on your crew "ONDUTY", you can change the label to what you do call it.
- The "Editable" column tells **CATS** that the dispatcher can edit the corresponding column in the Trains screen or prevent the column from being edited. For example, suppose you do not want the dispatcher to be able to change the crew assignment on the Trains screen, check the "Visible" box and uncheck the "Editable" box. Even though the "Editable" field may be checked, **CATS** may not save the result. For example, suppose the *TIME_LEFT* field on the Crew screen is editable and the dispatcher changes the value, **CATS** will recompute the time left, anyway.

  There is a problem in **CATS**. Suppose you add a new train, how do you enter data into an editable field? The answer is that all fields in **CATS** are editable on new entries, until the changes are accepted. After accepting

the changes, fields with the "Editable" characteristic not checked, cannot be changed.

- The "Width" column tells **CATS** how wide to make each column on the Train screen. I found it tedious to put a number in the width column, look at the Train screen, adjust the width column, look at the Train screen, etc. An easier way to do things is take a guess on this screen, then look at the Train screen. If you drag a column border on the Train screen and push the "ACCEPT" button, this column will show the column width changes you made. The width of the table cannot be resized on the Train screen, so if you need a wider table, make the entries on the width column larger.
- The "Alignment" column is used to position the text string in the column of the Train screen.
- The "Class" column is used to define what kind of data can be entered into each field. You cannot change this column on the mandatory rows.

Here are the kinds of data ("Class") you can select from for the fields you define:

- "Boolean" puts a checkbox in the column.
- "Classpec" puts the same kind of pull down list that you use to select the class. There is probably no reason to select it.
- "Crewlist" puts a selection list in the column. The list contains all the names of the crew.
- "Extralist" also puts a selection list in the column, but the names are those crew members who can be assigned to trains.
- "Integer" puts a number in the column.
- "String" puts a sequence (string) of characters in the column. Be sure to make the column wide enough to hold the longest message.[16] There are no restrictions on which characters can be used, except they must be printable. "Enter" is not a printable character.
- "Timespec" puts a String that represents a time in the column. See below for more details.
- "Trainlist" puts a selection list in the column. The list contains the names of all trains that have not been tied down or terminated.
- "AlignmentList" tells **CATS** how to align the field contents (centered, left justified, or right justified).
- "FontSpec" puts up a selection list in the column. The list contains the names of the fonts defined at the time the list is created. There is probably no reason to select it because **CATS** and **TrainStat** will do something about a font only in the *FONT* field.

Another area for improvement is how data is entered into a cell. Any entries you make will not be permanent unless you touch the "enter" key. So, if you make a change and it disappears, it could be for either of two reasons. One is that it does not fit (e.g. entering "xxx" into a field expecting an integer). Another is if you did not touch "enter" before moving the cursor to another field.

[16] This is an area for improvement because I would like to be able to support more than simple phrases.

What is a “Timespec”? A “Timespec” is a way of specifying time. It can be an absolute time (number of minutes since midnight), a relative time (number of minutes since some other time), or the difference between 2 times. Here is how to indicate each:

- Absolute time: “HH:MM”. This pattern specifies the time of day in military (24 hour) format. “00:00” is midnight. “12:00” is noon. Because no day is specified, the current day is always assumed.
- Relative time: “+HH:MM” or “-HH:MM”. This pattern specifies the number of hours (less than 24) and minutes (less than 60) since another time. If either of these appears in the *DEPARTURE* field, then the time is computed as relative to the time at which the operating session began. So, “+1:30” is interpreted as an hour and half after the operating session began. If a fast clock is used, then it is an hour and a half, fast clock hours. If either of these appears in the *ONDUTY* field then it is interpreted as relative to the time in the *DEPARTURE* field. If the *DEPARTURE* field is blank, then it is relative to the time the crew is assigned to the train. If the *DEPARTURE* field has an absolute time, then it is relative to that time. If the *DEPARTURE* field has a relative time, then the *ONDUTY* time is computed by taking the time the operating session begins, adjusting it for the *DEPARTURE* time, then adjusting that for the *ONDUTY* time.

11 Jobs

The Job menu is used to define crew jobs or positions.

11.1 Edit Jobs

The Edit Jobs menu item is used to add, delete, or change a job description. A job description is any crew position on the railroad to which you would like to record a person’s name. It is intended to be used for providing a computer record of who worked what position at each operating session.

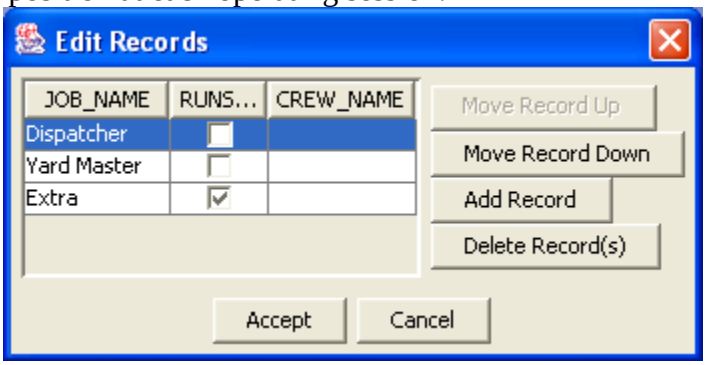


Figure 35: Job Edit Pane

CATS recognizes 5 fields:

- JOB_NAME is the name of the job.
- RUNS_TRAIN is a checkbox. If checked, then the crew member is on the "extra" board and is included on the Crew column of screens that assign crew to trains.
- CREW_NAME is a selection list from which you can pick one name from the Crew screen.
- ASSISTANT is also a selection list from which you can pick one name from the Crew screen. You can use it for things like assistant dispatcher, conductor on a two person crew, etc.
- FONT is used to create a list of defined fonts (Section 14.1.2). Currently, it is included only for consistency with the train table because nothing uses it.

11.2 Save Jobs

The Save Jobs menu item is used to save the Jobs information in a separate file.

11.3 Load Jobs

The Load Jobs menu item is used to read in a saved Jobs file. The Jobs file must have been created by Save Jobs.

11.4 Edit Job Fields

The Edit Job Fields screen is used to customize the Job screen.

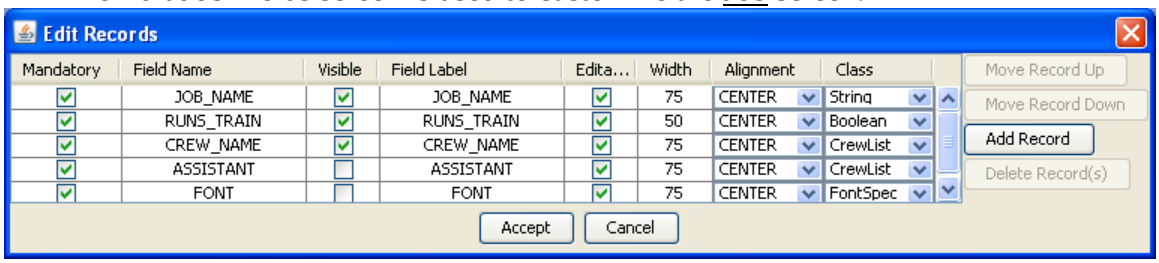


Figure 36: Edit Jobs Columns Pane

It works similarly to the Edit Train Fields screen. So, you can add fields, hide fields, re-label fields, etc.

The "Assistant" field will used in the future so that two people can be associated with the same job. For example, one crew member could be an engineer on a train and another is the conductor. So, when a train is assigned to one, both crew members would be assigned to the same train.

12 Crew

The Crew menu is used to record who is working the operating session and what they are doing.

12.1 Edit Crews

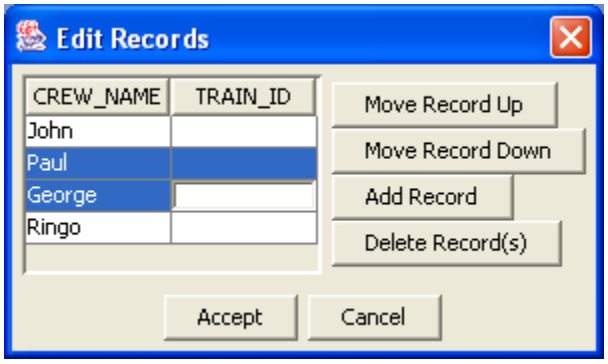


Figure 37: Crew Edit Pane

CATS recognizes 6 fields for each crew member:

- CREW_NAME is the name of the crew member.
- TIME_ON is the time the crew came on duty on the train assignment. This is usually the *ONDUTY* field from the Train screen. However, if the crew replaces someone (i.e. is a relief crew), then it is the time the new crew is assigned.
- TIME_LEFT is the amount of time left for the crew to work before they go "dead on the law". The time is measured using a fast clock (if a fast clock is being used) or the computer clock. It is computed by taking the current time, subtracting the *TIME_ON* value to get the amount of time worked so far. This is then subtracted from the *HOURS* time.
- EXPIRES is the time at which the crew has worked the legal hours. This is computed by adding *HOURS* to the *TIME_ON* value.
- TRAIN_ID is a place holder for a selection list containing the trains that have not completed their runs. These are the trains to which a crew could be assigned.
- FONT is used to create a list of defined fonts (Section 14.1.2). Currently, it is included only for consistency with the train table because nothing uses it.

12.2 Save Crews

This menu is used to save the crew list and the formatting to a file.

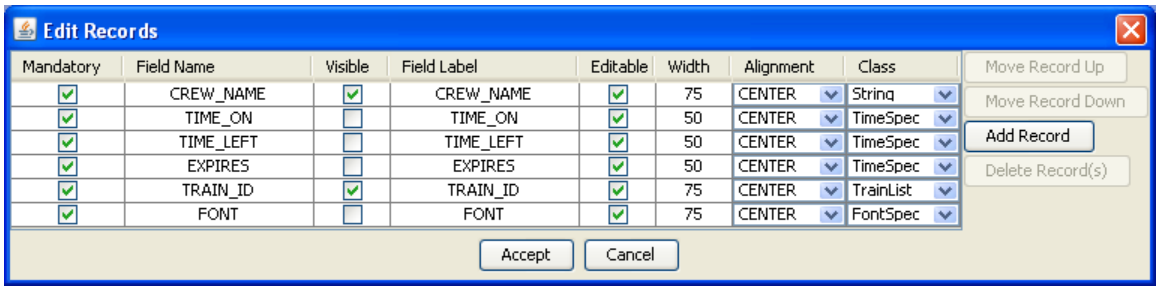
12.3 Load Crews

This menu is to read a crew list and formatting instructions. The crew list is an ASCII file (created with a text editor, such as Notepad), one crew name per line.

12.4 Edit Crew Fields

Edit Crew Fields is to customize the presentation of the Crew screen.

Figure 38: Edit Crew Columns Pane



It works similar to the Edit Train Fields screen. One note about the *TIME_ON* field. It is set to the time that the crew is assigned to a train. If the *EDITABLE* box is checked, then the dispatcher (and **TrainStat**) can change it. If the *EDITABLE* box is not checked, then the dispatcher (and **TrainStat**) cannot change it. Thus, if a crew is assigned to a train in the **CATS** *Edit Crew* pop-up window, *TIME_ON* will be set when the assignment is made.

12.5 Legal Hours

The Legal Hours screen is used to specify how long (in hours and minutes – hh:mm) a crew can legally work before they must be relieved. The hours should be in the time base (fast clock or computer time) used for the operating session.

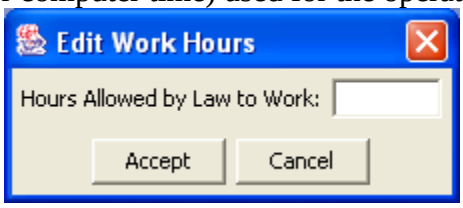


Figure 39: Legal Hours

13 Common Operations

This section discusses some checkboxes and entry fields used in multiple windows.

13.1 Defining Decoder Addresses

This can be the most difficult part of creating a dispatcher panel because it is not part of railroading. It is also one of the most difficult parts to explain because there are so many ways to connect a computer to a model railroad (Xpressnet, Loconet, C/MRI, etc); so many different kinds of connection devices (SwitchIts, SE8Cs, etc); and there are also many ways to use the devices to control things on the layout. Yet, it may be the most important part because this is where **CATS** meets the layout.

Many **designer** windows require decoder information. For example,

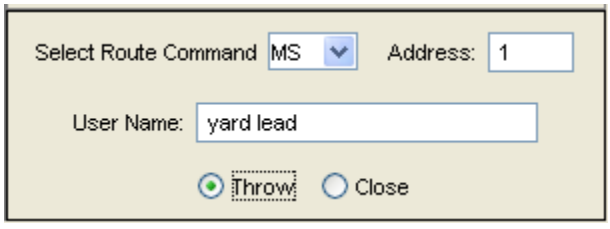


Figure 40: Decoder Specification

They have a panel containing a pull down list of JMRI names (e.g. *Select Route Command*) which is used to tell **CATS** that there is a decoder and how to use it. The pull down list is used for describing the way the detector is connected to your computer. It follows the JMRI naming convention. Your choices are the ones checked on the JMRI Device window (Section 8.3).

The *address* field is where you enter the "address" of the device. This is the address of the layout element on the connection. For example, the SE8C can control 8 security elements and has a board id. The board id is not the address. The board id is used to compute the address of each signal head, of each turnout control, of each sensor message, and of each push button message. This field, too, follows the JMRI naming convention.

Finally, the last buttons provide the "sense" or "polarity" of the report or command. This will depend upon how your decoders are wired and if you don't know how they are wired, just pick one and try it. You can always come back to this screen and change it. You can also run something like a Loconet monitor, make a change with a throttle and by looking at the monitor and the layout, deduce which radio button is correct. For defining SignalHeads only, see also section 14.7.

The discussion of naming decoders brings up a rather subtle topic. **CATS** breaks from JMRI in that **CATS** makes no assumptions about the relationships between decoder addresses. For example, JMRI defines internally, what an SE8C is and how it is wired to the layout (which addresses are used for controlling the A1 head, for the A2 head, for the B head, and for the C head). **CATS** would prefer to treat an SE8C as simply a bunch of Loconet addresses, similar to a LocoIO. In fact, **CATS** does not know what is at those addresses – SE8C, LocoIO, DS-54, and so on. This gives you flexibility in wiring your layout and you tell **CATS** how you wired it up, rather than you following a set of wiring rules. This simplified the design of **designer** and **CATS** and made them adaptable with many decoder vendors. However, the price paid is that **CATS** does not utilize some of the logic in JMRI and the user has to enter many decoder addresses. An even more subtle effect could be that you use a device that enforces a relationship and **CATS** does not know about that relationship. For example, the SE8C and DS-54 can be programmed for independent operation (when track occupancy is sensed, signals are set a particular way). This independent action could interfere with what **CATS** is doing. Watch out for those kinds of devices. You should not avoid them, just be aware that they may do things that you did not intend and you may have to change some of their option settings. Also, **CATS** conflicts with how JMRI handles turnouts. JMRI maintains the state of a turnout internally. **CATS** also needs the state of the turnout for checking for fouling points.

In general, select an entry from the selection list in 8.3 whose second letter is 'S' or 'T' (however, Loconet users need to read Section 8.3 carefully). The first letter will depend on how the computer attaches to the layout. Because Loconet uses different

messages for what the layout reports and for controlling the layout, this advice does not apply to Loconet connections.

13.2 Positioning Something in a Grid Cell

Another common menu in **designer** is a selection list of where to place something in a grid cell. There are selections like “upcent”, “leftupper”, and so on. What these do is provide some fine adjustment on locating an item. Each selection (except for “center”) is made from two directions. The first is the edge of the cell and the second is where along the edge. For example, when placing a signal in “upcent”, the signal will appear along the top edge with roughly equal empty space on its left and right. “upleft” means it will be placed on the upper edge, biased to the left. This placement is very close to “leftup”, except for “leftup”, the signal will be against the left edge, biased towards the upper edge. So, in contrast, it will be higher with “upleft” and closer to the left edge with “leftup”.

This really gets interesting with items that are wider then the cell. Things can spill over the right edge (“left” in the selection), spill over the left edge (“right” in the selection), or spill over both edges (“center” in the selection). But in any case, if it is not where you want it, try another selection.

Because train labels move above the track line, position names and signals below the track line, if convenient.

14 Fine Tuning the Presentation

This section is intended to be a set of application notes and hints on how to wire your railroad and set up **CATS** for prototypical operation.

14.1 Appearance

The Appearance Menu has some menu items that can be used to change the way the dispatcher panel looks. Any changes to these are stored in the XML file for **CATS**.

14.1.1 Colors

The Colors menu item is used to select a color from a palette of 256 colors, for the color in which things are painted in **CATS**. Actually, by using a different tab, you can select any color supported by Java.

All colors must have names because **CATS** uses the name to find the color. This sounds like it is taking the long way to get somewhere. However, it is based on the observation that with only a few exceptions, if a color is used once (for example, in a station), it is used multiple times. The things being colored are related to each other in some way. Thus, if the color of one thing is changed, most likely, all of its relatives should be changed identically.

The color table looks like the following:

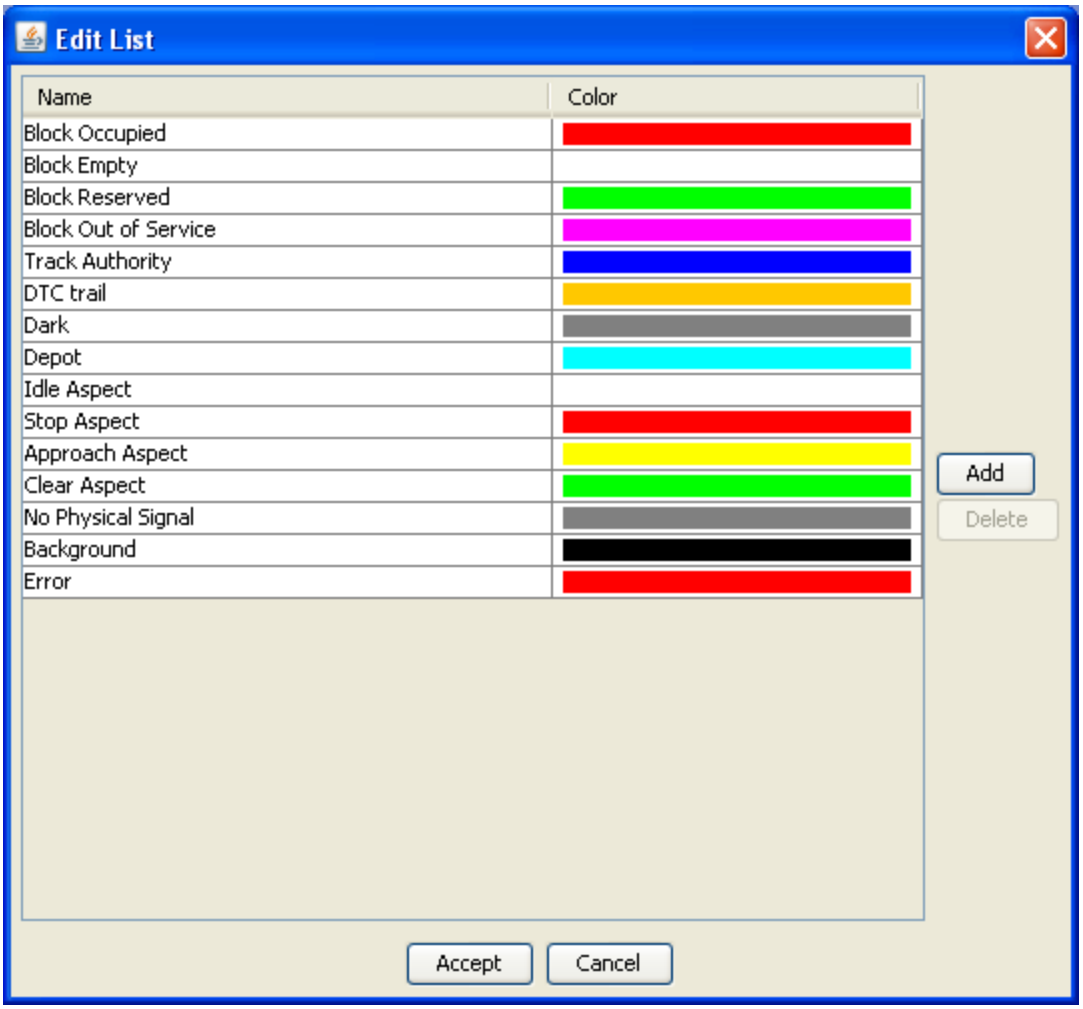


Figure 41 Color Table

There is one row for each named color. The name of the color is in the left column. The right column is a button that will present a color editor. That color editor depends on the Java user interface running on your computer.

CATS requires the following named colors. You can define more with the *Add* button, but you cannot delete any of these.

- Block Occupied is the color used to paint a section of track which is occupied.
- Block Empty is the color used to paint a section of track which is not occupied, reserved, out of service, or involved in track authority. This color is also used to paint signal icons under the dispatcher's direct control that are not involved in an authorized train movement.
- Block Reserved is the color used to paint a section of track which has been reserved for a train movement.
- Block OOS is the color used to paint a section of track which has been taken out of service by the dispatcher.

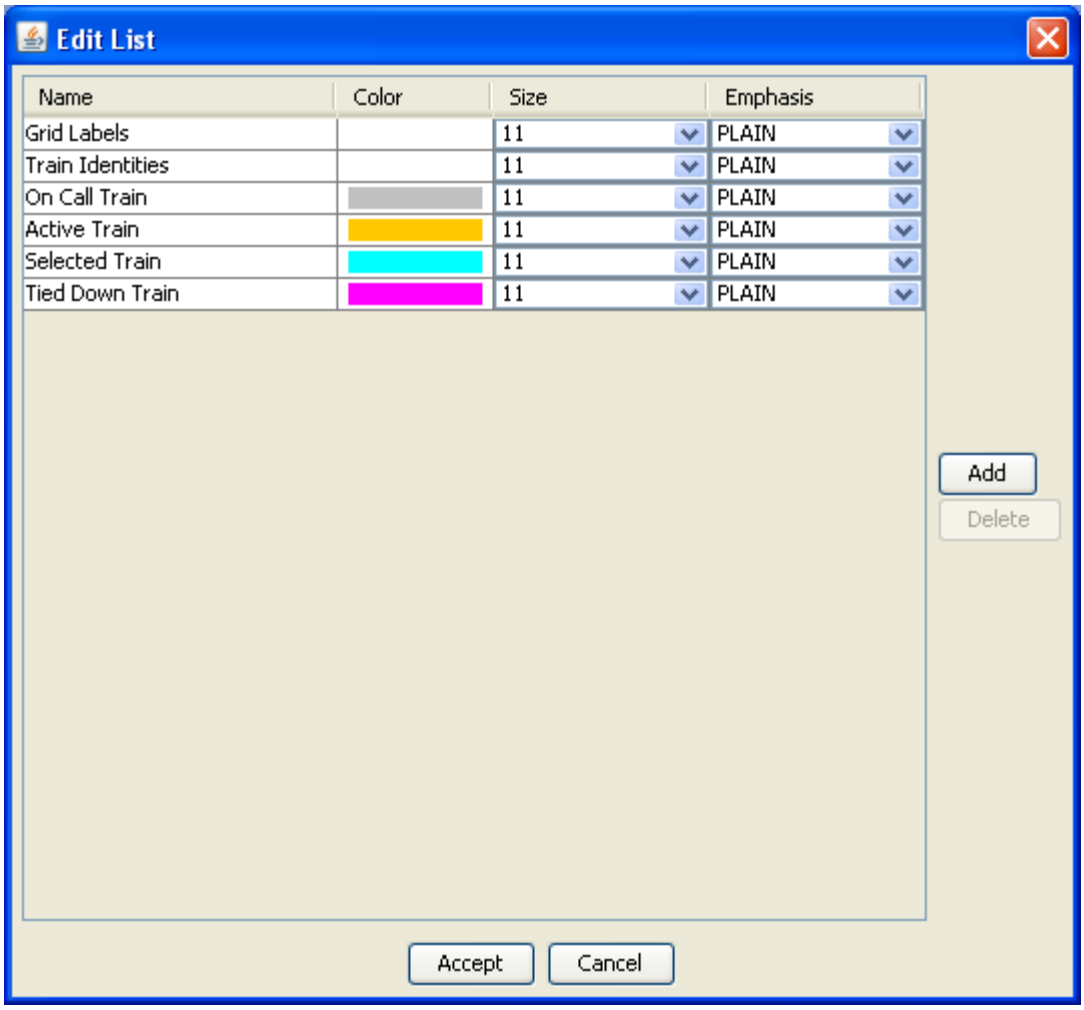
- Track Authority is the color used to paint a section of track which the dispatcher has given to a train for local work (track authority)
- DTC Trail is the color used to paint a section of track which is under DTC, was reserved for a train, was occupied, and then unoccupied. It designates the track as idle, but within the limits of some track permission.
- Dark is the color used to paint track which has no train detection.
- Depot is the color used to paint the square icons that represent the location of depots.
- Idle Aspect is the color used to paint a signal icon on the panel that is not involved in a route or protecting ABS or APB blocks. On the older Digicon panels, this was white. On newer panels, it is red.
- Stop Aspect is the color used to paint signal icons which are not under the dispatcher's direct control (i.e. ABS and APB) or are involved in an authorized train movement showing a Stop indication.
- Approach Aspect is the color used to paint signal icons which are not under the dispatcher's direct control or are involved in an authorized train movement showing that the train can proceed, but must stop at the next signal.
- Clear Aspect is the color used to paint signal icons which are not under the dispatcher's direct control or are involved in an authorized train movement showing that the train can proceed.
- No Physical Signal is the color to paint signal icons which are under the dispatcher's direct control, not involved in a train movement, but do not have a visible signal on the layout. This is a reminder to the dispatcher that the train crew does not know what aspect the signal is showing.
- Background is the color of the canvas the dispatcher panel is painted on.
- Error is the color used for drawing track that will give **CATS** problems (Sections 6 and 9.3).

14.1.2 Fonts

This menu changes the way train names and track labels are shown. The size, color, and the highlighting (normal, bold, italic) of each font can be changed.

All fonts must have names because **CATS** uses the name to find the font. Similar to colors, this sounds like it is taking the long way to get somewhere. However, it is based on the observation that with only a few exceptions, if a font is used once (for example, in identifying a station), it is used multiple times. The things being labeled are related to each other in some way. Thus, if the font of one thing is changed, most likely, all of its relatives should be changed identically.

The font table looks like



The first column is the name of the font. The second column is a button used for selecting the color of the font. Just click on it and a color edit pane (dependent on the Java user interface) will be presented. The third column sets the size of the letters. The fourth column provides emphasis to the letters – PLAIN (none), ITALICS (slanted), BOLD (wide lines), and BOLD-ITALIC (slanted, wide lines).

CATS requires the following named fonts. You can define more with the *Add* button, but you cannot delete any of these.

- Grid Labels is the default font used by the Section Name operation (Section 9.2.3)
- Train Identities is the default font. It ensures that if a name is deleted that **CATS** will have a font that it can use.
- On Call Train is the font used for trains that have not been run and are awaiting a crew.

Figure 42 Font Table

- Active Train is the font used for trains that have crew assigned and are not selected for movement through the cursor keys.
- Selected Train is the font used for the train that has assigned crew and can be moved by the cursor keys.
- Tied Down Train is the font used for trains that have completed their work and been tied down.

14.1.3 Line Widths

The Line Widths menu is used to change the width of lines. Horizontal and vertical lines have the same width. The two forms of diagonal lines have the same width.

14.1.4 Grid Size

The Grid Size menu is used to change the horizontal and vertical dimensions of the grids. Usually, the best appearance is when they are the same. The grid size option has been disabled because **CATS** assumes the default size in several places and things are not painted correctly if the grid size is changed.

14.1.5 Adjustments

Adjustments are counters and delays, used to fine tune the interface to the layout.

14.1.5.1 Occupancy Debounce

When we first installed **CATS**, we ran into some false detection issues. We saw quick, random phantom detections where a block with no train in it would flash occupied for less than a few seconds, then return unoccupied. We eventually chased these phantoms down to "silent drive" locomotive decoders tripping Digitrax BDL-162 decoders. In an effort to eliminate these phantoms, I added an occupancy filter. The way it works is that a block must be occupied for at least as long as the value specified before **CATS** accepts it as a real occupancy report. Typically, real detections would last at least four seconds (this appears to be a characteristic of the detectors), so setting it for more runs the risk of losing real detections. Setting it for less shortens the amount of time the block is marked as detected by the amount of the debounce interval. You can change this setting on the fly in **CATS**.

14.1.5.2 Refresh Delay

When **CATS** loads a layout description or the dispatcher refreshes the layout, **CATS** will send a flood of commands to the decoders on the layout. This flood may overwhelm the command bus and connection hardware. Setting this value to non-zero means that **CATS** will send 10 commands, then wait for the number of milliseconds specified before sending the next 10. You can change this setting on the fly in **CATS**.

14.1.5.3 Loconet Governor

This adjustment applies to only devices from the MS, ML, and MR (see 8.3) classes. We found that it is easy to saturate Loconet with commands. By setting this value, **CATS** will wait for the number of milliseconds specified between commands. You can change this setting on the fly in **CATS**.

14.1.6 Fast Clock

Fast Clock is a check box. When it is checked, the time stamp on all train movements is taken from the fast clock (if the layout does not provide one, JMRI will). When it is not checked, all time stamps are taken from the computer clock.

14.1.7 Engine Label

Engine Label is a checkbox. When checked, **CATS** will use the lead engine number as the train label. If there is no lead engine number, **CATS** will use the train symbol. If not checked, **CATS** will use the train symbol. You can change this setting on the fly in **CATS**. This checkbox just sets the initial value.

14.1.8 Include File

One of the powerful features of JMRI is that Jython and other scripts can be used to sense and control decoders. For example, you can create a set of signals with Panel Pro and run scripts containing your own custom logic to set the signals. If you have decoders defined through Panel Pro, you can import those definitions into **CATS** every time **CATS** starts without having to load them by hand. To enable this, simply click on this box and select the name of the Panel Pro configuration file.

To erase a file, enter the name of a file that does not exist.

14.1.9 Flash Rate

As noted above, **CATS** can flash signal lights. This menu lets you select the flash rate (in milliseconds) for the on and off phases.

14.1.10 Tee Base

This checkbox selects the way that a base of a signal icon is drawn. The default (not checked) is as a triangle. When checked, it is an inverted tee. See section 9.2.1 for more details.

14.1.11 Direction Arrow

This checkbox selects how routes are drawn in **CATS**. The default (checked) is for each route to have an arrow head showing the direction the train is expected to exit the block. When not checked, no arrowheads are drawn. This option does not affect **designer**. It is here so that it can be included in the layout description file, saving the operator from an extra step when starting **CATS**.

14.1.12 Compress Screen

This checkbox selects how **CATS** handles columns with only horizontal track. The default (checked) is for **CATS** to try to squeeze the screen horizontally. When not checked, all columns are the same width. See section 6 for more details.

14.1.13 Automatic Wrapping

This checkbox selects how **CATS** determines when to stop painting tracks on a row and move to the next row. The default (checked) is for **CATS** to try to break the row at a place where the fewest tracks connect to the next row. When not checked, **CATS** will run the row up to the edge of the window. See section 6 for more details.

You can force a specific geometry by drawing the screen exactly as you want it in **designer** (not as one linear track), making connections using the non-adjacent track option (see Section 6), and specifying the screen dimensions (Section 14.1.16).

14.1.14 Lock Turnout Decoders

This checkbox selects how **CATS** decides if it is safe to allow the dispatcher to throw a turnout or not. The default (not checked) is that **CATS** ignores decoder commands in determining safety (because it takes time to make the extra check and many layouts do not share decoder addresses). When checked, **CATS** will remember which commands would move a locked turnout and disallows any turnout from being thrown that would send one of those commands. See section 9.3.2 for more details.

14.1.15 Reverse Local Operations

This checkbox enforces dispatcher control of turnouts (see Section 9.3.2.1). When it is checked, **CATS** watches for switch points moving, when not under track authority. When they move, **CATS** will send the command to move them back.

14.1.16 Screen Size

Selecting this menu item pops up a pane that allows you to specify where on the computer monitor the **CATS** window should be placed. It also lets you specify the size of the window. The way to use it is start up **CATS** and load your layout description. Then, position and resize the window. If you look at the Java console, you will see the position and size of the window, every time you adjust its size. You can note those numbers and enter them into the pane popped up by Screen Size.

14.1.17 Keyboard Shortcuts

Consistent with the way Windows and Microsoft programs do things, some of the drop down menu items have keyboard shortcuts. They are listed on the drop downs, next to the item title. Here is a summary of them.

Control c	**C**opy the grid tiles to the clipboard
Control e	Bring up the track **E**nds pane
Control n	Clear the canvas for a **N**ew panel
Control o	**O**pen a file

Control r	Add a new **R**ow below the current row
Control s	**S**ave the layout
Control t	Bring up the **T**rack selection pane
Control v	Paste the clipboard at the cursor location
Control x	Clear the tiles selected
Control y	Add a column to the left of the current column
Control z	Undo the last command

14.2 Where to Create Blocks

There is a temptation to scrimp on adding block detectors because they involve more wires; they are expensive; and they require cutting gaps in the tracks. The questions you should be asking yourself are "where do I want operations to go in parallel? Where do I want to allow two trains to operate independently?" As a general rule, each passing siding should have a minimum of 4 blocks: the main, the siding, and each turnout (OS section). If the siding has a continuation track to something like a spur, then the OS section is essentially a cross over. For example,

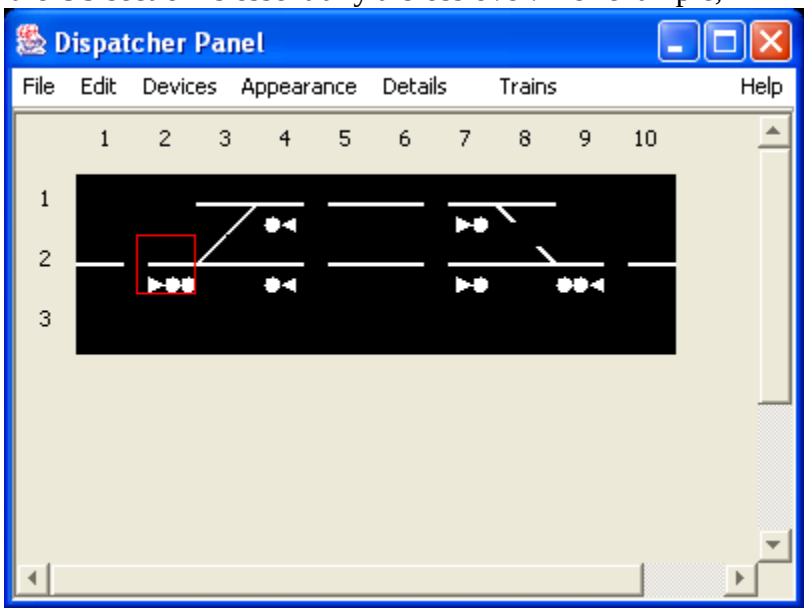


Figure 43: Example 1 - Block Placement

The diagram has seven blocks (moving from left to right): the right bound approach, the OS section on the left, the main, the siding, an upper OS on the right, a lower OS on the right, and a left bound approach. Let's compare the left and right OS sections. On the left, if a train is working the upper spur, then the OS section is occupied. Therefore, the mainline signals protecting it will show red and the main is blocked so nothing can go through, even though the working train is not actually interfering. On the right, the dispatcher can grant "track authority" on the upper OS block and as long

as the lower turnout is set for normal, the crew can work the spur without blocking the main. An alternative way of presenting the above is

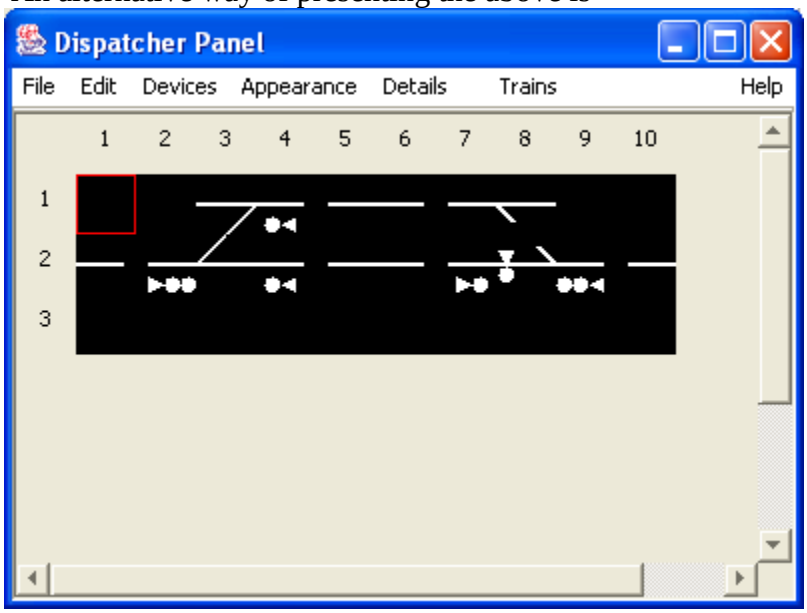


Figure 44: Example 2 – Block Placement

which is not pretty. Often what is really intended is

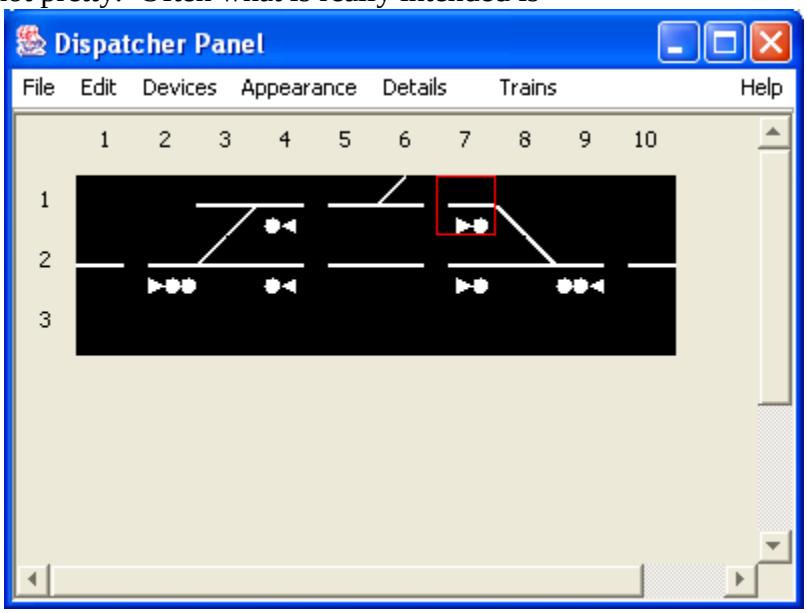


Figure 45: Example 3 – Block Placement

14.3 Yard Entrances

Often, yard entrances are a variation of the above:

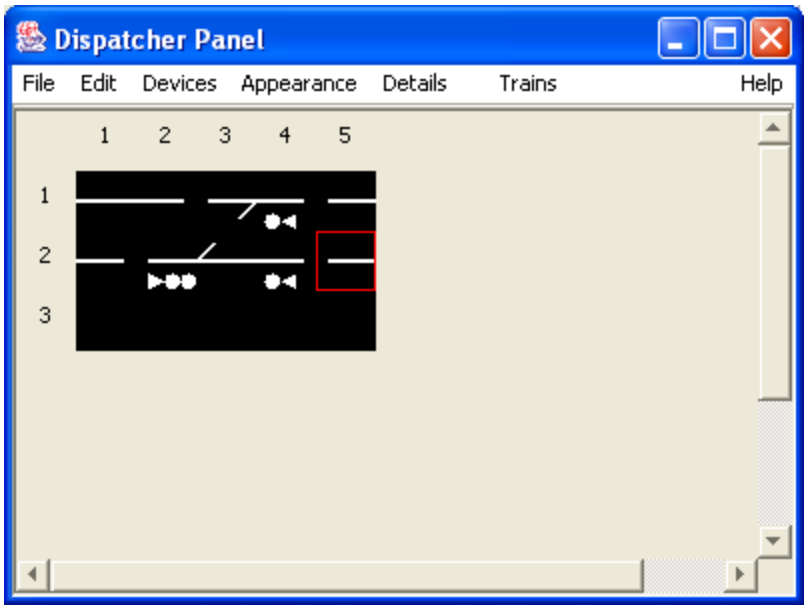


Figure 46: Yard Entrance

where the double headed signal identifies the mainline on the bottom and the yard drill track is the track on the upper left. In the above, the yard limit is the middle of the cross over track. The reason for the block boundary on the crossover is to solve the problem of who "owns" the cross over. Often the first inclination is to wire the turnouts in lock step (we did this). They are both set for the normal route or both set for the cross over. Though this arrangement provides a degree of safety, it complicates operations. If the dispatcher has control of the cross over, then the dispatcher could interfere with switching in the yard by just setting a route into the yard. Furthermore, when the yard work occupies the normal route through the upper turnout, the lower route is also occupied. If the yard controls the cross over, then the dispatcher must wait for the yard to align the crossover before setting a route into the yard. Again, this requires considerable coordination.

A workable solution is to control the turnouts independently, define the blocks on the mainline as CTC, define the blocks in the yard as ABS, add turnout position feedback in the yard turnout, and let **CATS** provide the security. The upper signal is the "exit" signal.

1. The dispatcher sets the mainline for the thru route. The mainline signal operates independently of yard activity. If the yard is also set for the through route, then the yard exit signal will reflect the activity on the left.
2. The dispatcher sets the mainline turnout for the cross over, but the yard is set for the through route. The dispatcher can set a reservation into the yard from the left, but the signal will show stop, because the route is fouled. So, the dispatcher can set things up and work on other tasks. If the intent is to send a train out of the yard, then the signal in the yard will reflect conditions to the left. But, the crew will be in contact with the yard crew and should be told when they can leave.

3. The yard aligns the yard turnout for the crossover, but the main is set for the through route. The "exit" signal will show stop because the track is fouled.
4. Both routes are set for the crossover. If the dispatcher sets a reservation into the yard, the "exit" signal will show red and the double headed signal will be "approach" because the train is leaving CTC territory. If the dispatcher is setting up an exit from the yard, the "exit" signal will show "clear" (the next block is set up) or "approach" (the next block is not set up).

Thus, an action is required by both the dispatcher and yard crew to allow a train to move into or out of the yard and the actions need not be synchronized.

14.4 Placement of Decorations

Train labels usually move along the topside of tracks. Therefore, it is more legible to place the signals below the tracks. Plus, this tends to be more consistent with the look of the prototype panels (probably for the same reason). See the above diagram (Figure 45) for an example.

14.5 Turnout Position Reports (Feedback)

CATS can handle no position reports, reports on one route (of a two leg turnout), or reports on all routes (see 9.3.2). Having sensors on all routes is best, but a sensor on one route is better than none. If the layout uses a single microswitch or feedback sensor to report turnout position, **CATS** works best by filling in the "Route Selected Report" panes on the routes on the Points window. It might seem to make more sense to fill in the "Route Selected Report" and "Route Unselected Report" panes on the same route, but the first pane tells **CATS** when the points are aligned to a particular route and the second pane tells **CATS** when they are not aligned for that route. However, not being aligned for a route does not mean the points are aligned for a different route because the points could be somewhere in between the two (which is why sensors on all routes is best).

14.6 Diode Matrices

Suppose you have a yard throat, such as a multi-track staging yard, and you want to set all the turnouts for any one track with just one click. Historically, model railroaders have used diodes to achieve this. There are several ways to set this up:

1. Use the route capabilities of a stationary decoder (such as a Digitrax DS-54)
2. Set up a route in your command station (if it supports routes)
3. Use a JMRI Route object
4. Use a single **CATS** Chain
5. Use linked **CATS** Chains.

This section will illustrate the latter.

The basic concept is to define a **CATS** Chain for each track. Each Chain is composed of at least two elements (more elements may be required, depending on the turnouts). One element in each Chain aligns the turnout for the attached track. The other element can be either the Chain definition of the Chain for the preceding turnout or a dummy decoder (such as a JMRI Memory object). If the latter, in the definition

window for the turnout preceding the one being defined, set the "*Select Route Request*" pane to be the same as the dummy decoder. Thus, when the dispatcher selects a track in the yard, **CATS** sends a command to each decoder listed in the Chain. One command will be to the decoder that controls the turnout. Another command will be sent to the dummy decoder. Nothing will happen immediately except a request will be sent to the preceding turnout to set itself for the requested alignment. If it is safe to do so, **CATS** will execute the preceding Chain (or command its decoder). Note that the safety checking makes this scheme safer (e.g. if a decoder is in an occupied block, the command will be ignored) than simply throwing turnouts. However, the command ends with the first unsafe action or turnout that is aligned correctly. Consequently, it is more reliable to make the second element the Chain identity of the preceding turnout.

It is tempting to use the same technique on a crossover from one track to another (two turnouts are involved). However, you may be surprised at the results if an element in each Chain refers to a decoder that the other turnout is listening to. What will happen is that one Chain will stimulate the second, which will stimulate the first, which will stimulate the second, ad infinitum. Eventually, Java will crash. So, do not do this. Rather, create a Chain containing all the decoders and assign it to the common leg of both turnouts.

Here is an example of how to simulate a diode matrix. The tracks are numbered.

Figure 47: Example – Diode Matrix

Here are sample decoder definitions:

Track	Left Route	Top Route
1	Throw ML1	Close IC1
2	Close IC2	Close IC3
3	Close IC4	Close IC5

Table 7: Diode Matirx – Decoder Commands

Here are the Chain definitions:

Chain	Elements
IC1	Close ML1
IC2	Throw ML2, Close IC1

IC3	Close ML2, Close IC1
IC4	Throw ML3, Close IC3
IC5	Close ML3, Close IC3

Table 8: Diode Matrix – CATS Decoder Chains

Should the address of any of the decoders attached to any of the three turnouts change, only two changes need be made in the decoder definitions.

14.7 Defining Signal Heads

There are multiple ways to drive signal lights, multiple ways to wire them up, and multiple ways to define them in **designer**. **designer** leverages the JMRI SignalHead abstraction for controlling signal lights[17]. This means that SignalHeads can be defined in JMRI (Tools->Tables->Signals):

[17] Though **designer** treats signal semaphores similar to signal lights, the JMRI SignalHead abstraction for the Digitrax SE8C does not. Thus, if using an SE8C to control a semaphore, define the blade positions in **designer**.

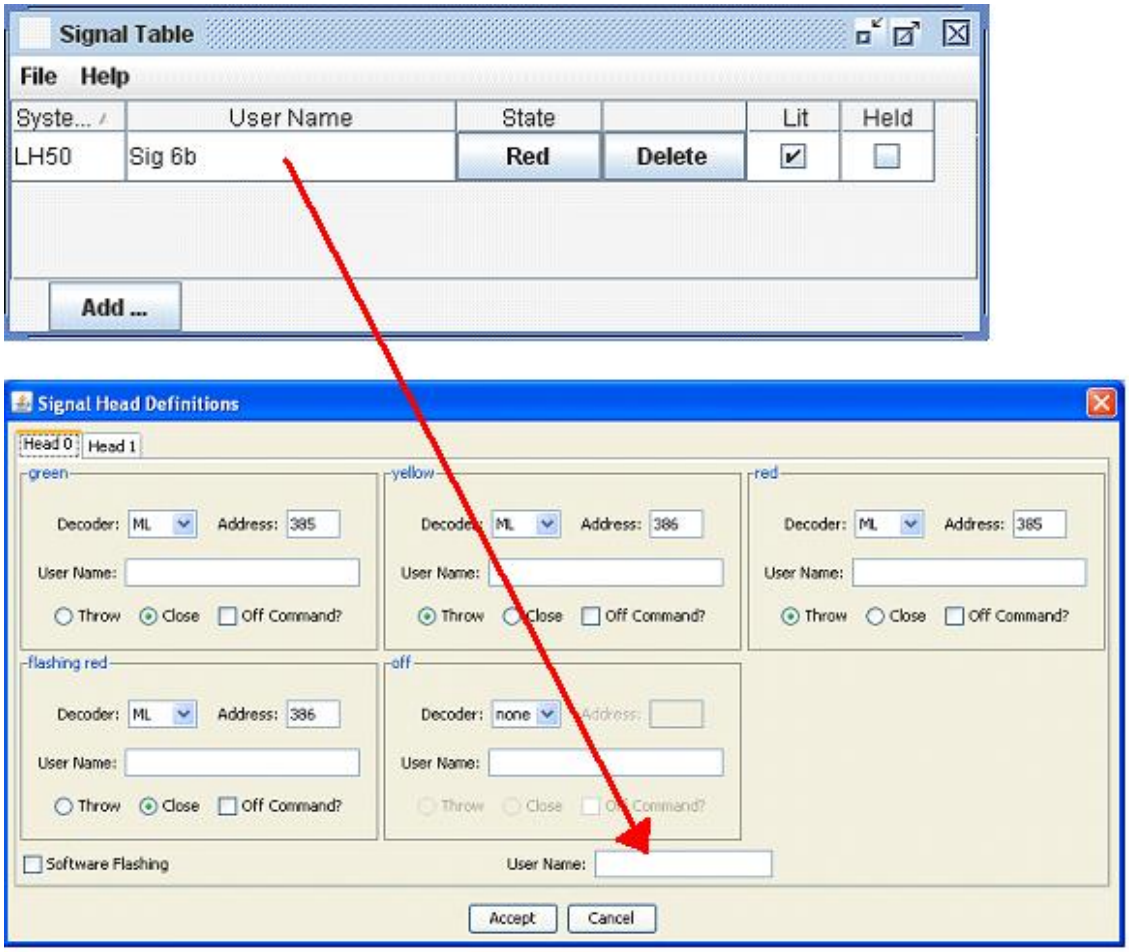


Figure 48: Using JMRI SignalHeads with CATS

The User Name field of the JMRI Signal Head can be placed in the User Name field of the **designer** definition (above the Cancel button):

Conversely, all SignalHeads defined through **designer** will appear in the JMRI Signal table when **CATS** runs and the **CATS** panel loads. Thus, JMRI scripts and Logix can access **CATS** SignalHeads. If you wish to do this, you should put the User Name that JMRI will use in the User Name field of the **designer** Signal Head definition.

Section 13.1 describes the window panes for entering decoder information. Signal color/position panes have an additional option.

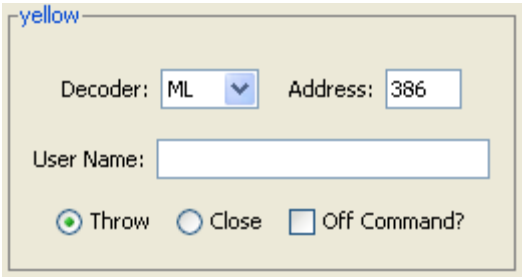


Figure 49: Signal Decoder Pane

The checkbox (*Off Command?*) is checked for decoders that need to be turned off when changing their appearance. For example, lights on an SE8C are mutually exclusive. If a head with the green light lit is turned to red, the SE8C automatically turns off the green, so the *Off Command?* box should not be checked. If each color on a head is driven by a different output line on a LocoIO, then *Off Command?* should be checked, so that **CATS** can turn off the current light before turning on the new light. When **CATS** wants to activate the output, it will send the appropriate command with the chosen polarity. When **CATS** wants to change the controlled layout element, it will first look at *Off Command?*. If checked, **CATS** will first send a command with the polarity not chosen, and then send a different command to set the new state.

This all means that SignalHeads can be defined in **designer** or in JMRI and referenced in the other. Here are some guidelines for where to define something:

- If the SignalHead is wired according to the instruction book for the decoder, as JMRI assumes, then the simplest way is to define it in Panel Pro, save the Signal table configuration, and import it into **designer** (see 14.1.8). You will save yourself some effort in filling in all the decoder definition fields because you will enter only the JMRI user name into **designer** once. **CATS** will then use the JMRI routines for controlling the signal.
- If the decoder can not be addressed as individual binary outputs (e.g. Grapevine), it will be easiest to define it in Panel Pro and use the User Name in **designer** (as above).
- Alternatively, you could define it in Panel Pro, define some internal turnouts (IT) in Panel Pro, and some Logix. The internal turnouts would appear in the **designer** color definitions and the Logix would translate the IT events to SignalHead events.
- If the decoder has addressable binary outputs, but the signal is not wired according to instructions (e.g. the red and green leads were interchanged, or you want red instead of dark), the colors should be defined in **designer**. Defining the decoder definitions in **designer** is more data entry, but means that the panel configuration file is self-contained. This is handy for backing up your panel. It also means that anyone who does not know how to create SignalHeads with Panel Pro does not need to learn (though the process is simple).
- If the decoder has addressable binary outputs (e.g. CMRI) and you are having multiple lights in a head lit simultaneously (particularly after first starting up), it is

possible to treat them as mutually exclusive. A technique to use is to leverage on JMRI Routes or **CATS** Chains. For each color in a SignalHead, define a Route or Chain. The Route or Chain will turn on one color and turn off the others. For example, the green Route or Chain would turn off yellow and red and turn on green. If you use this technique, you can uncheck the *Off Command?* box. If you let **CATS** create the flashing aspect, the dark or off aspect should turn all the colors off.

14.8 Selecting Train Font Colors or Size

Suppose you use the **TrainStat** application and you want certain trains to look different from other trains (see Section 10.1 for possible reasons), here is what you might do. For the sake of illustration, let's assume that a classification yard uses **TrainStat** so the yard master can monitor what trains are coming his way. It would be nice if the trains that stop are shown differently in **TrainStat** than the trains that do not stop. The steps for configuring the trains are:

- First, you will need to define some fonts, so click on *Appearance->Fonts ...* That will bring up the font editing window (Section 14.1.2). You will create a new font for each class of train. For example,

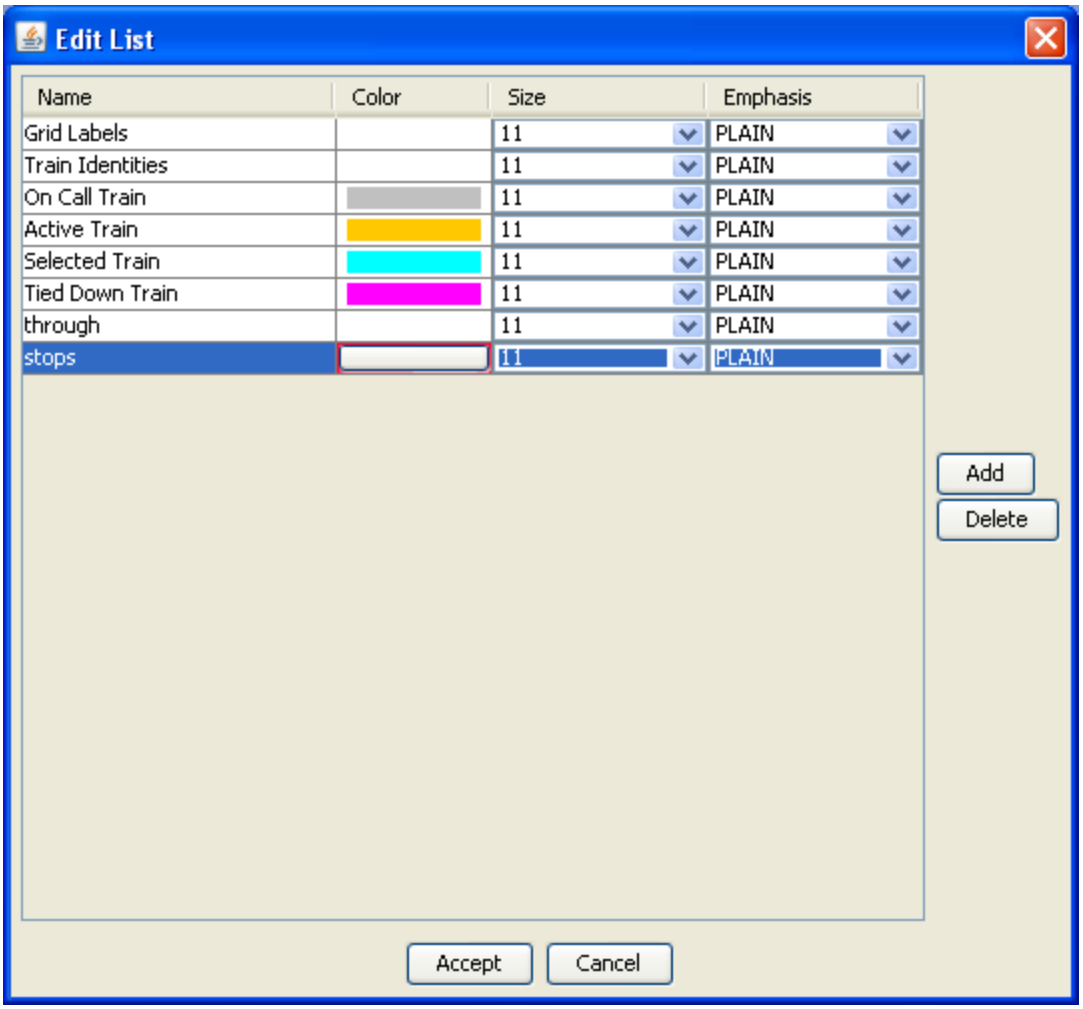


Figure 50 Creating Train Fonts

In this example, the fonts are named *through* and *stops*. The actual colors, sizes, and emphasis are unimportant because they are never used. See below for an explanation as to why they are never used.

- Selecting *Train->Edit Train Fields* will pop up a window (Section 10.4). To change the font on a train, it is necessary to make the *FONT* checkbox visible, so check the *Visible* column in the row containing *FONT* and click on *ACCEPT*.
- Select *Train->Edit Trains* to pop up the window that describes each train. For example, the following defines two through trains and one that stops. The through trains use the font named *through* and the one that stops uses the font named *stops*.
- This step is optional. If trains are not added during the operating session, again select *Train->Edit Train Fields* and uncheck the *Visible* box on *FONT* so that it does not appear. Hiding the column on the **CATS** train screen removes some clutter from the screen (plus if it is hidden, it cannot be changed).

The way things work is that **CATS** will send the information describing a train to the **TrainStat** application when the network connection is made. **CATS** does not send the

font information, only the font name. The reason is that the **TrainStat** application controls the presentation of the information, including the font. Thus, there should be font definitions for *through* and *stops* entered independently into **TrainStat.** The reason behind this round about way of doing things is that the original request for **TrainStat** was for a railroad with two yards and the owner wanted the trains that stopped in each yard highlighted for the respective yard masters. In other words, the trains that stopped in yard A might be identified as a bold red and those that did not stop might be a plain white. The same would apply for yard B. The problem is that some trains (not all) stopped in both yards and some stopped in neither. Consequently, that railroad has four train fonts (based on where the trains stop) and the **TrainStat** applications have different definitions of the four fonts.

15 Networking

As originally designed, **CATS** (and **designer** for creating dispatcher panels) was a stand alone program. In addition to routing trains, the dispatcher performed many of the administrative tasks, such as crew assignments. This model works on small to medium size layouts, but falls a part on larger layouts where the dispatcher is too busy and a crew position exists for Traffic Management. With Version 2.04 of **designer**, support has been added for defining an IP network connection. Another application in the **CATS** suite (**Train Status Client**) can run on a different computer networked to the **CATS** computer (or it can run on the **CATS** computer). The **Train Status Client** application listens to **CATS** for changes in a train's status and shows them. With Version 2.32 of **designer,** support has been added for working with JMRI Operations. Operations can be running on a different computer, so the support requires the ability to configure the networking to the other computer. The *Network* pull down menu is used to configure both network connections. By default neither is configured, so if there is no **Train Status Client** application running during an operating session or **CATS** is not getting information from Operations, the *Network* menu item can be ignored.

This pull down menu is only for the **CATS**. **designer** does not use a network connection.

The drop down box has three fields:

- *Server Port*
- *Start TrainStat Server*
- *Operations*

15.1 ServerPort

The *Server Port* box provides a place for entering an IP port number. This is the port that **CATS** will listen for connections from **Train Status Clients** on. If left blank, it defaults to 54321. It should be changed if the **CATS** computer has a conflicting application using that port.

15.2 Start TrainStat Server

Start TrainStat Server is a checkbox. When checked, **CATS** will enable status reporting when the CTC panel is loaded. If no **Train Status Client** is used during an operating

session, this box should remain unchecked. Warning: if this box is checked and the **CATS** computer has network troubles (or no network), **CATS** will hesitate for a lengthy period of when the CTC panel is loaded.

15.3 Operations

Clicking on *Operations* spawns another window for filling in the information on how **CATS** contacts the computer running Operations (which can be the same **CATS**).

Figure 51 Operations Configuration

If JMRI Operations is running on another computer and you know its network name, insert the name into the top field. If JMRI Operations is running on another computer and you know its IP address, put the IP address in the second field. Finally, if Operations is running on the same computer as **CATS**, simply leave the two top fields blank.

Operations uses the Simple JMRI Server for receiving requests from **CATS**. The Simple JMRI Server normally uses port 2048; however, it can use a different port (e.g. there is a conflict with another program). If it is moved on the Operations computer, the "Operations port:" field can be used to tell **CATS** what port the Simple JMRI Server is using. Usually, you can leave this field blank.

Similarly, **CATS** will typically use port 51431 for contacting the Simple JMRI Server. If this results in a conflict with another program, you can put a different value in the "Local Port:" field. Usually, you can leave this field blank.

If you want **CATS** to contact the Simple JMRI Server on startup, check the last box.

16 Working with JMRI Operations

With JMRI Release 2.14, **CATS** has some limited capabilities for working with JMRI Operations (car routing). Specifically, **CATS** can tell Operations when a train has entered a **CATS "**Station" (Operations "Location"). Operations can then adjust the train's manifest to account for any switching scheduled for that train at that Location. Furthermore, **CATS** can query Operations for the train's length at any time. The results

of the query will update the train's *LENGTH* field in **CATS** and be forwarded to **Train Status**. Thus, the dispatcher has a real time view of how long a train is.

Operations can be running under JMRI on a different, networked, computer or it can be running on the JMRI that **CATS** is using. In the former case, you will need to supply the specifics on how **CATS** can reach the other computer (see Section 15.3, which can be provided to **CATS** during the operating session). Alternatively, the default is that **CATS** will look for Operations in the JMRI instance that **CATS** is using; thus, the network connection is to itself.

None of this capability comes for free. Some things must be configured. An earlier Section discussed how to set up the network. **CATS** and Operations must agree on how Locations and trains are named.

Both JMRI Operations nd **CATS** are evolving towards more exchange of information. In the 2.99+ releases of JMRI, **CATS** can ask for a train's weight and number of cars, as well as length. The responses are used to populate the *LENGTH*, *WEIGHT*, and *CARS* fields in the train table. The astute reader will notices that with Operations enabled, those fields should not be changed by the dispatcher or an person running **Train Status** because they will be later changed through Operations.

In the 2.14 release of JMRI every train in the **CATS** train table should be in Operations because due to a bug in Operations, Operations will crash if **CATS** asks for an unknown train.

16.1 Stations and Location

CATS can designate places on the layout by using the Station field of a Detection Block:

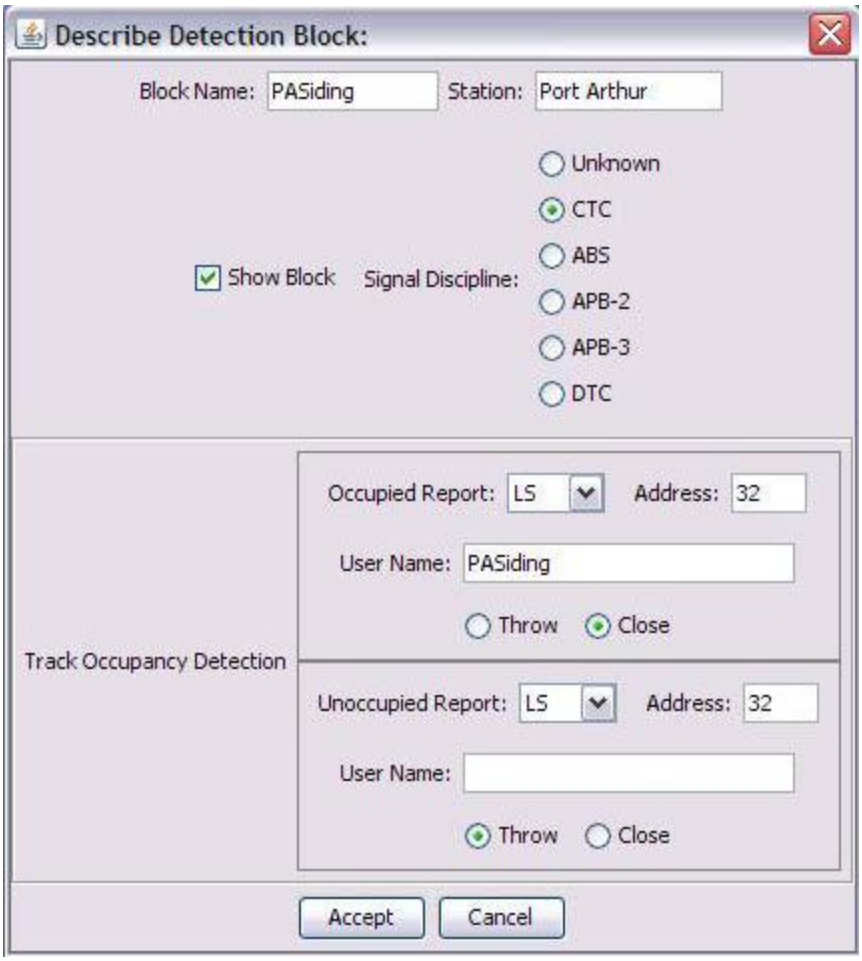


Figure 52 Station Field

If the Station has an equivalent Location in Operations, then the value of the Station field should be identical to the value of a Name in the Locations screen in Operations:

Figure 53 Operations Locations

A Station need not have a corresponding Location. If it does not, the train length will not change when a train arrives at that Station.

16.2 Train Symbols and Train Names

In **CATS**, the train Symbol is used to uniquely identify a train.

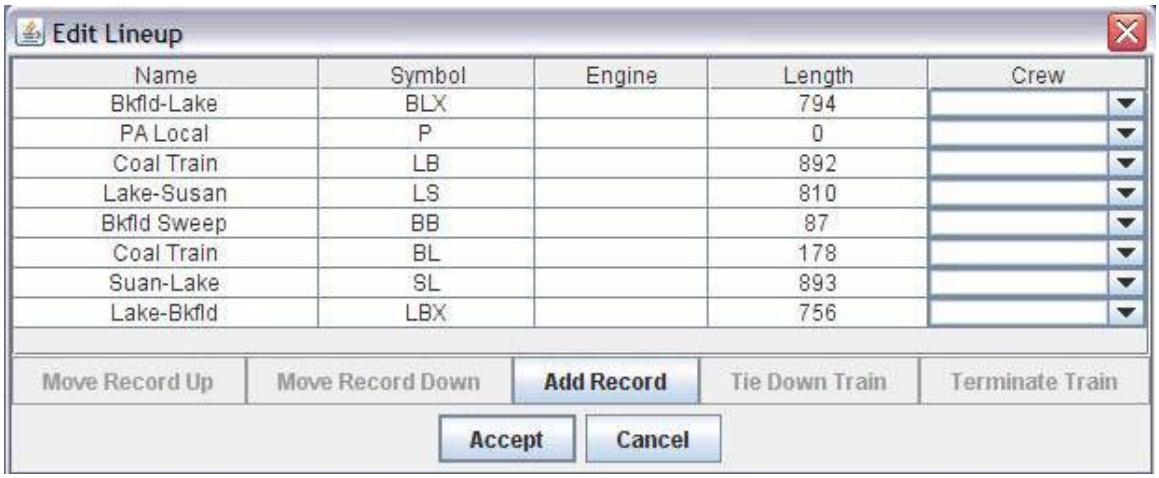


Figure 54 Train Symbol

In Operations, the train is identified by its name:

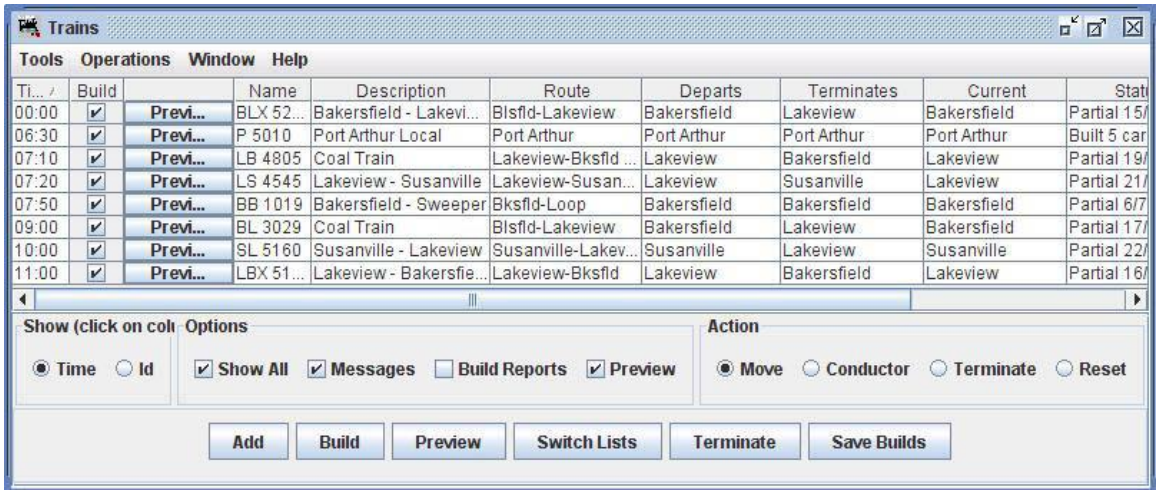


Figure 55 Operations Trains

Note that in Operations, a train's name may be composed of its symbol and a locomotive number. Use only the symbol in **CATS**.

16.3 Setting Up Operations

JMRI will need to be configured so that the Simple JMRI Server is launched at startup:

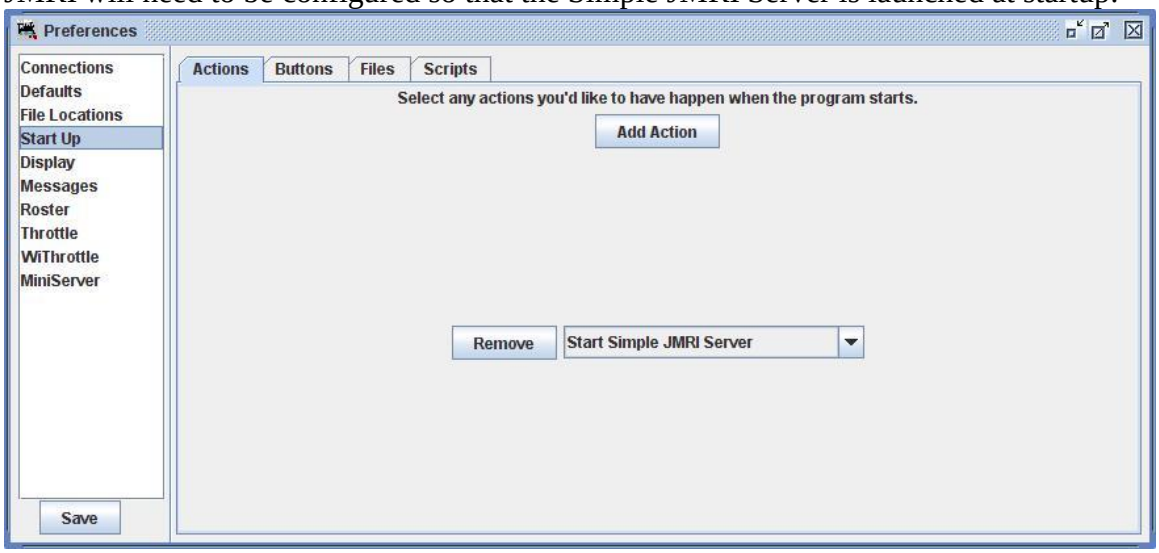


Figure 56 Simple JMRI Server

This screen is found from Edit->Preferences.

17 Troubleshooting

I wish that I could say that **designer** operated perfectly and has no bugs. I cannot. It is over 30,000 lines of Java code and not everything that can be done with it has been tested. Pasting is a particularly fruitful area to not work right because the block of tiles

being pasted has multiple edges and each edge must force a match with its neighbor in the existing layout. Nonetheless, here are a few tips on troubleshooting problems.

17.1 The Disappearing Window

Many users create a shortcut and icon to the .bat or .csh file that spawns **designer** and put it on the desktop, providing "one click launching". The problem is that the "one click" creates a command window, **designer** has trouble starting up, dumps some messages to the Java console, dies, and the command window disappears. All of this happens so quickly that you cannot see what the trouble is. The solution is to run the startup script (.bat or .csh file) by hand. Depending on your operating system, launch a command window. Under Windows, this is performed by Start->Run. Then "change directory" to where the startup script is located. Under Windows, the command is usually "cd C:\Program Files\JMRI". Then run **designer**. Again, under Windows, type "designer".

17.2 Warnings and Error Messages

Assuming the Java console window does not disappear, it could contain pleas for help from **designer**. So, if things seem to quit working, look at the Java console.

Do not be alarmed if a message pops up when loading an older layout description telling you under what version of **designer** it was created. As features are added to and changed in **designer**, efforts are made to allow the new versions to read in descriptions created by older versions. This is called backwards compatibility. However, sometimes it is not possible for a particular version to read in any description created by any earlier version. Because the amount of testing grows arithmetically with every new release, backwards compatibility is often not tested with descriptions. So, that message can provide some idea of how old the description is that is being updated.

It is usually impossible for older versions of **designer** to read in descriptions created by newer versions. If I had that kind of foresight, I would make a killing in the stock market.

17.3 Java Logging

The debug philosophy for **designer** has been evolving over time. The Java console is used for most warnings and error reports, but I am increasing the use of the log4j facilities. The file named "default.lcf" contains a filter on the kinds of messages that will be written and where they will be written. The location of this file depends on the operating system. It is usually in the same directory as the JMRI roster and configuration files.

17.4 Signals Are not Working Right

You have your layout wired. You tested all the signals with a throttle (or Panel Pro). They can display all the desired aspects with DCC commands. You launch **CATS** and some signals do not look correct. The reason could be a bug in **CATS**. But before you

fire off an angry E-mail, you should check several things. The first is that the aspects for a signal are defined correctly in the template for the signal (Section 8.1). Then check that the signal uses the template (Section 9.2.1). Finally, check that decoder instructions have been defined for each color or semaphore position (also Section 9.2.1 and Section 14.7). Something that often happens is that the conditions **designer** uses for determining colors and positions change after the decoder instructions have been defined. For example, suppose you begin by not assigning speeds to any tracks. **CATS** will automatically make them "default", which translates to "normal" on straight routes and "medium" on diverging. All of your signal templates will allow you to set aspects for indications that do not have "limited" or "slow" speeds. Your signals will ask for instructions for the aspects defined in the templates. Everything works. You go to a friend's layout and see the neat looking "Slow Approach" indication protecting entry into the classification yard. So, you change the speed on the diverging route into your yard to "Slow". However, when you try it, you do not get what you expected. If you then examine the template for the signal that protects entry into the yard, you will see some cells involving "Slow" that can be edited, which could not before. They have the **CATS** default indications. You change the aspects (say, you make the top head flashing yellow), but it still does not work. Next you should look at the decoder instructions for the signal. You will probably find that there is a box for "Flashing Yellow", but it contains no instructions. This is because the instruction window was filled out before any tracks used a "Slow" speed, so no "Slow" indication appeared in any indication, so "Flashing Yellow" was not needed when the instructions were first defined.

The change could be in the reverse direction. A speed that was not used is no longer defined. In that case, some instructions have been defined, which will never be executed.

17.5 The Tracks are Drawn in Odd Colors

Designer uses the following colors for drawing tracks (the colors can be changed in the color selection pop-up – Section 14.1.1).

- *Error* (default is red): there is something wrong with the tracks that **CATS** will not like. If just the rails forming switch points are in the *Error* color, then the points are missing a definition of the normal route. Fix this because **CATS** will crash (Section 9.3). If non-point tracks are drawn in red, then the tracks are either not in a block or the block is not named (Section **Error! Reference source not found.**). This is not fatal. **CATS** will run, but the tracks will not show up.
- *Block Empty* (default is white): the tracks are in a named block and detectors have been defined for the block.
- *Dark* (default is grey): the tracks are in a named block and at least one detector has not been defined.

18 Installing and Building from the Source Code, Under eclipse

This section is not for the faint of heart. It is for anyone interested in building the **designer** application from the source code. I built it using *eclipse* (from

www.eclipse.org, an open source development toolset). It should build under other development environments, but I had one that works, so have not experimented.

When I release a new version of **designer**, I also post the source code. I package the source and *eclipse* settings in a zip file for separate download. Here is how I set up a new build environment:

1. I create a new *eclipse* project.
2. I download the *designer.zip* file from the web site.
3. I unzip the *designer.zip* file in a separate directory (folder).
4. I select (or create) the designer project from the *eclipse* project navigation list.
5. I import the *designer* "file system" where the unzipped files reside

Designer needs very few supporting files. It needs a Java runtime, jdom.jar (for handling the XML files), and log4j.jar (for error reporting). The first comes with the Java Software Development Kit (SDK). I use the latter two from the JMRI distribution.

The launchers (*designer.bat* and *designer.csh*) are constructed by hand. They are very simple.

19 References

- "How to OPERATE Your Model Railroad", by Bruce Chubb
- "Realistic Model Railroad Operations" by Tony Koester
- "All About Signals" by John Armstrong
- "Railroad Signaling" by Brian Solomon
- NMRA Operations Special Interest Group (OP_SIG) - http://www.opsig.org/resources.shtml
- "Absolute-Permissive Block Signals", parts 1-5, by Jay Boggess, Model Railroader, Nov. 1991 – Mar 1992
- "Signal Basics", parts 1-3, by Doug Geiger, NMRA Bulletin, Aug.-Nov., 1996
- http://www.lundsten.dk/us_signaling /abs_st_sp/ p_index.html
- http://broadway.pennsyrr.com/Rail/Signal/
- http://www.ctcparts.com/aboutprint.htm
- http://deltareum.com/signal_progressions.htm