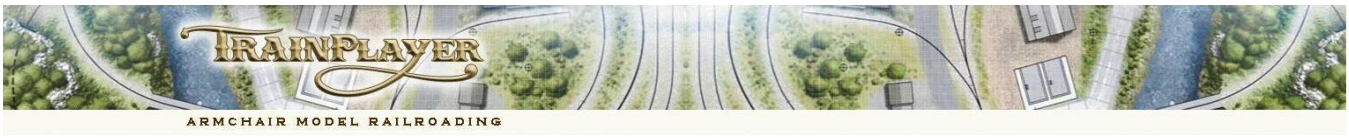


An introduction to the TrainPlayer Programming Language (TPL)

History and Overview

The concept of scripting was first introduced to TrainPlayer version 2.3 in 2006 with the addition of a text based language designed specifically for the running of the trains. This original scripting language made good use of two specific statement types, Commands to set the route and drive the trains, and Wait Conditions to control when, where or under what circumstances these commands would be executed. Each script was attached to a specific train which, if the script were played, would instruct the train to follow the speed, route and timing instructions as applied to it by the script.

The concept of sub-routines which could be called from within a **Train Script** was introduced with version 5.1 in 2012 . A **Subroutine** is just a series of script statements that may be needed several times and which are stored in an external file so that the same routine can be called up repetitively from within a Train Script or from another Subroutine.

TrainPlayer 6 introduces two new types of Script.

Junction Actions are scripts attached directly to junctions on the track (i.e. either to connections between two track segments, or to switches) to enable any specified train (or trains) to pick up and use the script as it crosses the junction.

Master Script which can be used to automatically execute a series of Commands when the layout is first loaded.

In addition to these new script types the TrainPlayer Programming Language (TPL) now uses a substantially enhanced set of statements, including new conditional statements to control the flow of the program as it operates the trains. There are some additional train commands and many new functions to access and use the internal data. There are also new functions which allow customized Wait Conditions to be designed and a variety of tools to allow the writer of the script to communicate directly with the user who is running the script.

The Different Types of Script

The four different types of Script can each be used in isolation to control a few simple tasks, or in combination with the other types of script to provide a high degree of layout automation and interaction between several trains.

Users with experience of the earlier scripting language may prefer to start with the familiar Train Scripts, whereas a novice may find it easier to start with a simple Junction Action script. This could be used to perform a simple task such as loading cars under a tipple and unloading them again on a trestle; this would require only two two scripts each containing a single line of code. Whereas a user who already has a few basic programming skills should find it relatively easy to completely automate a small layout with several trains running at the same time, and to produce code to interact with the user in the form of providing instructions, and then monitoring the actions taken by the user before providing further instructions.

The aim of this document is to introduce you to the concepts of the TrainPlayer Programming Language (which will now be referred to as TPL) by providing an explanation of the functioning of the different script types, and how to find and use the appropriate dialogue boxes to write and edit these scripts.

Train Scripts are scripts attached to a particular train or engine, they are initiated from the Train Script editor and can be edited by that same editor, or from the Scripts Tab of the new Script Central dialogue.

Subroutines are scripts stored as separate text files, they need to be written with Windows Notepad or another external editor, but they can be subsequently accessed and edited from the Subroutines tab of the Script Central dialogue. When you call a subroutine you can also pass additional arguments to it which are represented by placeholders in the script.

Junction Actions are scripts placed on the track where they can be picked up and executed by any train that passes over that track connection (even non scripted trains). They are written using the Junction Action editor, but can be subsequently edited from the Junction Actions tab of the Script Central dialogue.

Master Script is a script attached to a layout for executing only once as the layout is loaded. A Master Script is not attached to any train. It can be used to set variables, define new procedures and start the sequence of trains. The Master Script is initiated and edited from the Scripts tab of the Script Central dialogue.

Each of these script types will be explained in more detail once we have dispensed with the preliminaries of ensuring TrainPlayer is properly set up for scripting work to begin.

All four types of script will accept any of the TPL instructions and any script type can call up a subroutine (including a subroutine calling another subroutine). There is no limit to the number of commands that can be used in any of these script types, although for some simple Junction Actions only one or two lines of code may be needed.

The Scripting Language Enhancements

In this document we will confine ourselves to providing only an overview of the various statement types, layout commands, train commands, flow commands, user interface commands, wait conditions, system variables and system functions.

All four of the Script types described above can use the full extended set of TPL statements. These include:

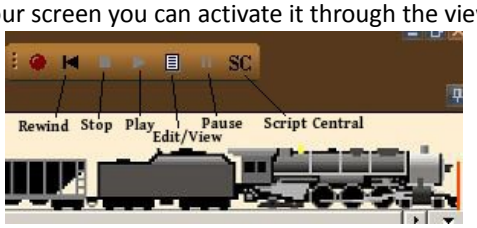
- Layout Commands - To set routes, control turntables, play sounds and load and unload cars at set points on the layout.
- Train Commands - To start, stop and drive the trains, set the direction, control the speed, and initiate other train scripts.
- Flow Commands - To control the flow or order of events based on different conditions encountered while running trains.
- User Interface Commands - To pass messages, process responses, record key presses, and set values for processing.
- Wait Conditions - To ensure subsequent commands are not processed until certain events have already occurred.
- System Variables - To access rapidly changing data which is automatically updated by the program as the trains run.
- System Functions - Functions to process the data held in system variables and manipulate the information found.

The best way for a novice to learn about writing scripts is to study the code used in existing well commented scripts and to experiment with short batches of their own code on a small test layout.

A full Glossary of Terms for all the TPL control statements is provided under the reference tab of the Script Central dialogue (to be described shortly) and these are also covered in more detail in a separate document with examples of their use for ease of reference. **See TPL_Ref_ Doc_2.pdf**.

The Script Toolbar

For writing or playing any TPL script the Script Toolbar is essential, if this is not already displayed on your screen you can activate it through the view menu.



The purpose of this document is to explain the new features of TPL and we will only refer very briefly to the Train Script recorder (red light) which only makes use of the original restricted command set. The purpose of the other icons on the Toolbar are labeled in the image.

The Script Recorder

The script recorder has been part of TrainPlayer since the introduction of scripts in 2006 and it still functions in exactly the same way. However scripts written with the recorder only make use of the original Command Set, and they only use a single Wait Condition AT, which is applied to every line of the script which involves a moving train.

To use the Script Recorder, open the Train Script editor by clicking on the Edit/View icon on the script toolbar, click the Record Button on the editor, set your route and drive your train. You will see the script being written automatically as you move. When you are done click Stop, Rewind your script and click play.

This is a typical snippet of code which was probably written using this method.

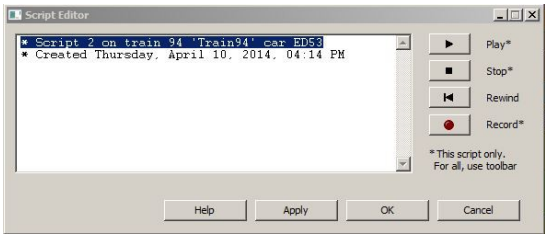
```
on stop
throw 170 1
at (103 100 40 r) throw 192 1
at (103 100 40 r) forward
at (103 102 59) speed 15
at (190 189 33) stop
on stop
throw 48 0
at (190 189 60) reverse
at (190 187 39 r) speed 15
at (35 32 72 r) speed 6
on couple
stop
```

To help you understand the meaning of this code we will translate only the third line which states: "when the train (which is set in reverse and has already stopped with the center of the lead car standing over a point 40% from junction 100 on track segment 103, then the script must change the switch no 192 to position 1". Although you could use this recorder method as a starting point for writing a Train Script, this document is about using the new tools to write you own code, so no further mention will be made of Script Recorder.

The Train Script Editor

To write your own train script, start by selecting the appropriate engine and then click on the Edit/View icon on the Script Toolbar. This will open the Train Script editor with a blank script which contains some default text in the form of comments.

You should see something like this:



The default text generated comprises two Comments. We can leave these as they are or alter them to something else.

Comments are signified by * an asterisk, // two slashes, or a # followed by at least one space.

Comments starting with ** two asterisks plus one space, instead of just * one asterisk, will not echo to the schedule window when the script is run.

Comments are lines within a script that do not perform any actions, they are there solely to aid the readability of the script and we would recommend that any script should include lots of them. In its present form this script does absolutely nothing, but the purpose of this section was to demonstrate how to open the dialogue needed to write a Train Script.

Just to demonstrate that this is a script, you could type in the following three lines of code below the comments:

```
forward
speed 10
after 0:0:05 stop
```

Now click rewind, and click play just to demonstrate to yourself how a Train Script works. your train should travel forwards at 10 mph for five seconds and then stop.

The Junction Action Editor

To position a Junction Action we first need to select an appropriate track connection point or insert a new one. In this example we are going to add a script to load the cars one by one as they pass under the loading tipple.

In this image, the track under the loading tipple is a short length of hidden track the same length as the roof cover.

We need to split the track at the point depicted by a red circle. To be able to examine the track we need to first highlight the White pointer icon on the TrackLayer Toolbar.

Next we right click at the point marked here with the red circle and select "Create Action Here". You will only see this option if the TrackLayer toolbar is active. This opens the Junction Action dialogue box depicted in the image.

If you right click on an existing junction rather than a section of track you will see the option "Action..." rather than "Create Action Here", this fulfills the same task and opens a new Junction Action dialogue.

When the Junction Action dialogue first opens the drop down boxes are preset to "Any Train". In this case we need to modify this using the drop down arrows and select AAR in the first box, HM for hoppers in the second box, and W>E (West to East) in the right hand box. We then add our comment and our single line script to "load $x_car".

load $x_car is a TPL Script instruction to identify the car crossing the junction and to load it with its default load.

Once this information is entered we click Apply at the bottom of the dialogue to add the script to the layout (Note: clicking OK also adds the script to the layout and simultaneously closes the Junction Action dialogue).

After splitting the track for the new JA script you will find that one half of the formerly hidden track section is now visible and you will need to right click it and set it to Hidden again. If you are testing this sequence on a layout of your own, this might be a good time to click Rewind on the Scripts Toolbar, and Save your layout.

You can now test your Junction Action by driving a train under the tippler, each empty car entering the tippler should come out the other side carrying its default load. The train will load car by car, irrespective of whether it has a train script or not.

Any number of TPL script commands can be used in a Junction Action but here we only required two lines, a comment and a single line Command instruction to identify the car and load it with its default load. In our example we also had to use the same process to add Junction Action Scripts to the other three loading tracks.

To summarize, a Junction Action reads the details of the train crossing its Junction to check if the instructions relate to the particular train or train type. If appropriate the script is temporarily attached to the train, if the train doesn't meet the criteria then the script is ignored or a further check is made to see if a different Junction Action instruction has been stored for this particular train (or train type).

Other typical uses for Junction Actions might be to to stop briefly in a station, to set down and/or pick up a cut of cars at an industry, or to route a train through a switch depending on the train type, train name or composition.

Subroutines

Subroutines are a useful tool preferred by several experienced scripters but they are by no means essential and the novice would probably be better not to concern themselves with this feature until more proficient with the other script types.

Subroutines are simply standard TPL scripts that are written up in separate text files using Windows Notepad (or some other text editor). They are stored in a separate Scripts folder in your Apps data, underneath the TrainPlayer folder and at the same level as your Layouts folder and Carsets folder. Their main advantage is that they can be used to code repetitive tasks so that only a single line CALL is needed, rather than needing to insert the same code into every script that needs it.

To call a subroutine, you use the new CALL command and supply its filename (without the extension).

```
CALL mysubroutine
```

You can also pass additional values, or arguments (separated by spaces) to a subroutine (if an argument consists of more than one word, enclose it in quotes). These arguments are represented within the subroutine script by placeholders which are replaced by the actual values of the arguments when the subroutine is called.

Insert placeholders where your routine will take variables from the calling script. A placeholder is of the form %n, where n is the position in the argument list from left to right. For example, if your routine takes two arguments, use %1 in the code to stand for the first argument, %2 for the second. For example you could use:

```
CALL mysubroutine 123 1
```

To pass the id number of a switch and the position you want it to be in. The appropriate line within the subroutine that used this information would then be simply THROW %1 %2 which would substitute 123 for %1 and 1 for %2.

A subroutine can even be used as a complete Train Script in which case the script stored in the Train Script or Junction Action script would only need to be a single line instruction to CALL the subroutine.

When a script encounters a subroutine CALL it opens the file, copies it into the current script at the position of the CALL command and closes the file. The subroutine itself does not become attached to any train nor stay open while it is being executed. Once the loaded subroutine is finished executing control returns to the remainder of the calling script.

The Script Central Dialogue

Before explaining how to create a **Master Script** it is necessary to introduce the new comprehensive **Script Central** dialogue as this is where most of the subsequent work on your scripting will be done. To open the Script Central dialogue just click on the SC icon of the Script Toolbar. You will see that this has several separate tabs which cover different Script Types, plus some Reference and Settings information.

The Scripts Tab is the active tab when the dialogue opens. Here you should see a copy of the Train Script if you prepared one when we described this process above. If you are viewing a different scripted layout you should see all of the Train Scripts, clicking on any one of these will show the relevant code in the right hand pane. Now that these scripts exist they can be edited directly in this pane, or alternatively you can double click the script in the list and reopen it in the original Train Script edit window. You will also use this Tab to create your Master Script and we will explain how to do this shortly.

The Junction Actions Tab is next, click on this to see a list of any Junction Action Scripts on the layout. Again clicking on a Script in the list will show its contents in the right hand pane. Existing Junction Action scripts can also be edited directly in this pane, but if you need to change the trigger conditions you will have to double click the item in the list and make the changes to the drop down boxes using the original Junction Action editor.

The Subroutines Tab presents a similar list of external routines that may or may not already exist on your own TrainPlayer installation. Subroutines, as described above, are scripts that are stored as standard windows text files in a special Scripts folder, these can be called up from within any embedded script type using the CALL command.

The Reference Tab provides a detailed list of all the commands available to you for writing scripts in TPL. Just take a quick look for now but don't worry at this stage about trying to understand them. TPL is very powerful but it can also function well at a lower level, with a small Command set, and there is no need to learn to use all of these commands to get started. When a script is running the reference tab also shows the current values of any control variables being used in the scripts.

The Settings Tab is for saving the positions of your trains and switches, overriding the defaults which are already set to the positions as they were when the file was first loaded. There is also a facility to save your train positions and switch positions to a separate file, and to reload them.

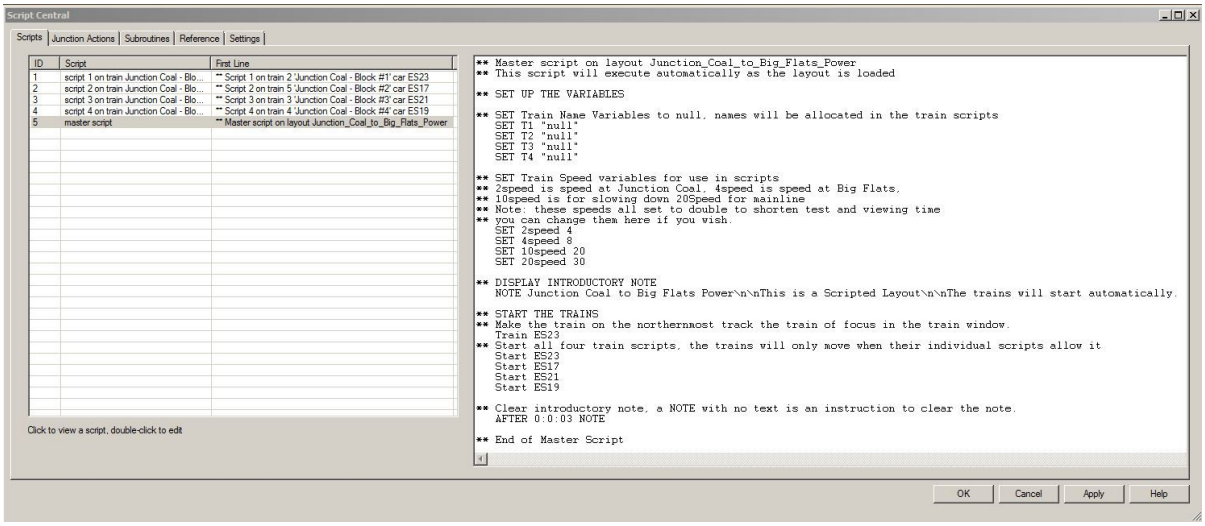
The Master Script

Now you have seen Script Central it is time to consider the Master Script. This is different from any other type of script in that it is not attached to any train or to any junction. What makes it special is that it will execute automatically as soon as the layout is opened. A layout does not have to include a Master Script, but if it does then it can only have one of them.

To insert a Master Script reopen Script Central by clicking the SC button on the Script Toolbar, select the **Scripts** tab (if not already selected). Right click on the Scripts list and select the option to "Add Master Script".

This will immediately add a new script to the list, including a default opening comment which you can leave in place or edit

as you wish. Remember if you edit, to leave the * asterisk in place to signify this is just a comment, or add an additional * so that you have ** two of them if you don't want your comments sent to the Schedule window or Status Bar. If you adjust the text you should click Apply before moving to any other tab, although the program will issue a warning if you do not.



The above image depicts a short but complete Master Script for the automated version of the Junction Coal to Big Flats Power layout. While you can use any TPL instruction in a Master Script, this would normally be used for tasks that you only want running once as the layout opens. These might include:

1. Setting up or initializing variables needed for adjusting and testing as the script progresses.
2. Displaying an Introductory Note or sequence of Notes, and testing for user responses.
3. Starting the individual Train Scripts running, or just starting an unscripted train (Using the Drive Block sequence).
4. Defining Procedures for evaluating subsequent conditional actions (we will include a Procedure example next).

Procedures

Procedures can be defined in any script but cannot be CALLed until the script that defines them has been run. It therefore makes sense to define them in your Master Script as this runs automatically when the layout loads.

A procedure fulfills exactly the same function as a Subroutine in that it can be used for any repetitive sequence of Script Commands. The only difference between a Subroutine and a Procedure is that the Subroutine is stored in an external file, whereas a Procedure is embedded in the layout, usually in the Master Script.

One particularly valuable use for Procedures is to design new customized Wait Conditions to hold up the processing of your script until the required conditions are met. This is an example which checks to see if a train s moving or stopped, whether it is set to a forward or reverse direction, and to check if the correct number of cars are in the train.

```
** HoldUntil, holds up the execution of a script until specific train conditions have been met.
** This can be used during user interaction to check if instructions have been complied with.

** Current version requires four parameters
** %1 Name of train for testing. (Can be passed literally, as contents of a variable, or as $x_train).
** %2 S if instructions require train to Stop, or M if instructions require train to start Moving again.
** %3 F or R, i.e. the direction setting needed on the Train Controller before allowing the script to continue.
** %4 Train length (car count) needed before allowing script to continue, to check couple and uncouple events.
**
PROC HoldUntil
   ** If train is not the required length, hold up the script.
   While ($train(%1,Ncars)<>%4); endwhile

   ** If train direction is not correctly set, hold up the script.
   While ($train(%1,Direction)<>%3); endwhile

   IF (%2="S")
      ** If train is required to stop, hold up script until it stops.
      While ($train(%1,Speed)>0); endwhile
   ELSE
      ** If train is required to move, hold up script until it starts moving.
      While ($train(%1,Speed)<1); endwhile
   ENDIF
ENDPROC
```

This procedure would then be called over and over at various places in the Train Script to check if the last set of NOTE

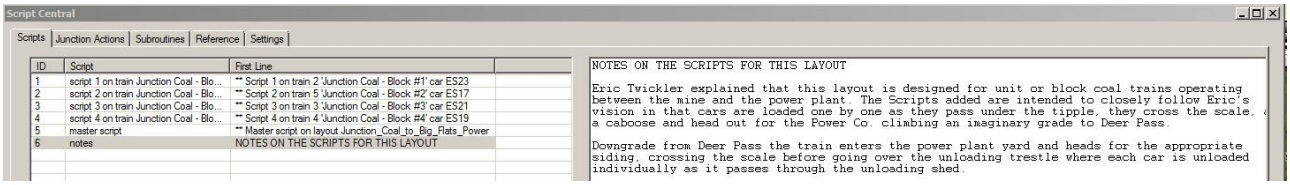
instructions had been complied with. e.g To check if the operator stops after Junction 513 facing forward with 12 cars.

```
AFTER J513
CALL HoldUntil $x_train S F 12
```

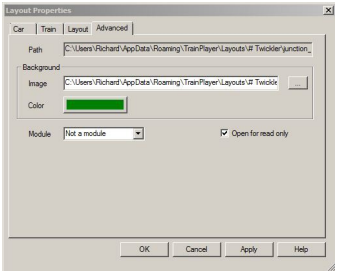
Adding a Note

This is an optional step that can be undertaken to take advantage of the fact that Script Central allows Notes of any kind to be stored under the Scripts Tab of the Script Central dialogue. It is important not to confuse this type of Note with a Note Window (see below).

Adding a Note to the Scripts Tab uses exactly the same procedure as adding a Master Script. i.e. Select the Scripts Tab in Script Central, right click on the list of scripts, and select "Add Note" from the context menu. Here is an example note from one of our scripted versions of Eric Twickler's "junction_coal_to_big_flats_power" layout. The note was used to provide a quick summary about the scripts we had added to this layout.



Protecting your Scripts from accidental damage by enthusiastic users



Once you are satisfied that the scripting of your layout is complete you can select Layout Properties from the context menu after right clicking on the layout. Go to the Advanced Tab and put a tick in the box which states "Open for Read Only" and Save your file.

When this file is reloaded it will be set for Read Only so that if a user accidentally clicks "Yes" when prompted to Save on closing the file, this will not overwrite the original, but instead it will offer an incremental file name in the Save As dialogue.

Adjusting the size and position of the NOTE Window

The NOTE window can be used by the Scripter to interface with his user, there can only be one NOTE window open at any one time and attempting to deliver a second NOTE while another is still open will cause the new text to overwrite the existing text.

This can be used to advantage in an interactive situation where instructions can be issued, checked for with a procedure (as above) and once the instructions are obeyed the next instructions can be delivered to the NOTE window.

The size and position of the NOTE window is not set from within the Script, so it is necessary to display a scripted NOTE that will not clear itself, and resize and reposition this before Saving. Saving the layout also saves the current size and position of the Note. There is also an advanced feature to script the note size.

Glossary of Terms for the TrainPlayer Programming Language (TPL) Statements

A full glossary of the statements used in TPL can be found under the Reference Tab of Script Central.

There is also a separate PDF version of this information which includes several examples of how to make best use of the new features. **See TPL_Ref_ Doc_2.pdf**.

Scripting a layout can be a lot of fun, if you look at another user's scripts they may appear daunting but it is important to remember that this is just a list of short sequences to perform specific tasks, and if you take a step by step approach scripting is no where near as difficult as it might first appear.

Richard Fletcher, April 2014