TrainPlayer 6.1 - Extensions to the Trainplayer Programming Language (TPL)

The purpose of this document is to summarize the new additions to the scripting tools that have been added since the release of v6.0. It is assumed that the interested reader will already be familiar with the TPL language that accompanied v6.0 without being aware of the recent additions to the language. These powerful new tools will not work with v6.0 and you will need to update to TrainPlayer v6.1 to use them; all these tools are fully documented under the Reference Tab of the Script Central dialog.

New Commands

EXIT	**Exit** is a new command that immediately terminates a script, equivalent to inserting a label at the end and using Goto Label. Caveats: -- If you insert Exit in a subroutine or procedure it terminates the calling script, not just the subroutine. -- If you use Exit in a Junction Action and the train is also running a train script, it terminates the Train Script not just the JA. Note: Exit is extremely useful during script development to prevent the processing of subsequent code when testing.
RETURN	**Return** is a new command for use within subroutines and procedures. It immediately terminates the subroutine/procedure and continues with the calling Script.
CONTINUE	**Continue** is a new command for use with conditional statements inside a While loop. Continue is used in similar manner to Break (i.e. from a conditional statement in the loop). It immediately suspends the current iteration of the loop, but instead of terminating the loop, it continues with the next iteration of the loop from the beginning and skips the processing of any remaining lines left in the loop.

Under the Hood Changes

Arguments	Jim has completely reworked the handling of arguments passed by scripts to Subroutines, Procedures, and Functions, the effect of this change has been to make the passing of arguments more robust.
Input Dialog	The format of the Input Dialog is changed, so the prompt is now displayed above the text box. The Input Dialog can now be resized in the same way as a Note window.
On Key	Keystrokes entered when the Note window has focus are now passed on to the underlying view. It is no longer necessary to click the layout background before On Key will detect a keypress with a Note window open.

Local Variables

LOCAL	**Local** is a new command for use within subroutines and procedures to declare variable names as "Local" to prevent multiple scripts which simultaneously call the same subroutine from interfering with each other.
Example:	LOCAL i, j let i=10 <== does not affect any global i declared elsewhere * If the subroutine is called by two different scripts the value of i will be kept separate in each script. Several examples of defining Local Variables within subroutines can be found in the new Subroutines library.

New $Scenery Function

$SCENERY(id,<argument>)	**id** is the ID number of the scenery item as seen under Show Numbers. **argument** can be any one of IsVisible, Show, Hide, or Toggle to allow scriptable control of scenery visibility.
$SCENERY(id, IsVisible) **$SCENERY(id, Show)** **$SCENERY(id, Hide)** **$SCENERY(id, Toggle)**	-- Returns 1 if the scenery object is visible, else returns 0. -- Makes the designated scenery visible if hidden, if already visible does nothing. -- Hides the designated scenery item if visible, if already hidden does nothing. -- Toggles the visibility of the designated scenery object. EXAMPLE to hide a ferry under an image of the water when it sails and restore it when it returns. AT 0700; call $SCENERY(1, Hide) AT 1500; call $SCENERY(1, Show)

New $Track, $Station and $Turntable Functions

$TRACK(id, OccupiedBy)	-- id is the id number of the track segment as seen on the tooltip or in the Show Numbers view. -- The second argument "OccupiedBy) can also be used with $STATION and $TURNTABLE -- The function returns a space delimited list of labels of the cars on the designated track. -- The resulting string can be processed by $string (see below).
$STATION(id,OccupiedBy)	-- Note $station can use either the station id number or the station name as the first argument.
$TURNTABLE(id, OccupiedBy)	-- Returns a list of labels of any cars on the turntable, this can be processed with $String (see below).
$TURNTABLE(id, Track)	-- Returns the id number of the turntable bridge track.
$TURNTABLE(id, AlignedTo)	-- Returns the id number of 1 or 2 locked external tracks, to check if rotation command is needed or not. -- Returns 0 if not aligned to any tracks.

```
EXAMPLE SCRIPT to check whether or not a Turntable needs rotating to accept the train.
* Stop train on Turntable approach track
AT 189 STOP
echo TTtrack is $TURNTABLE(1,TRACK)
echo $TURNTABLE(1,Alignedto)
* Copy current alignment string into a variable
let align = $TURNTABLE(1,alignedto)
let approach = 151
let jxn = 156
* Check to see if turntable is aligned correctly
* If not rotate
IF ($findstr(@approach,@align)=-1)
    rotate 1 @jxn
    on tablestop
ENDIF
echo $TURNTABLE(1,Alignedto)
Forward
speed 5
AFTER 156
AFTER 0:0:05
STOP
rotate 1 156 ccw
echo $TURNTABLE(1,OccupiedBy)
```

$File - A complete family of functions for file management

These three functions that use the filename as their first argument

$FILE(filename, SaveVars)	-- Writes out the complete list of all global variable names and values to a named comma-delimited file. -- Filename can be a full or a relative pathname.
$FILE(filename, ReadVars)	-- Reads in the names and sets the values of the user variables from a previously saved file.
$FILE(filename, Open)	-- Use with LET to open a named text file, returns an id number file handle or 0 if file not found. -- The filename you provide is first checked as a literal path, if not found it is treated as a relative path. EXAMPLE which looks for the text file in the subs folder alongside the rrw file. let fh = $file("subs\myfile.txt", Open)

These three functions use the File Handle [obtained from $file(filename,open) function] as their first argument

$FILE(id, ReadAll)	-- Reads the complete contents of the file and returns it as a display string. -- This gives the same result as using $READ(filename) -- see below. -- First argument is the file handle obtained from the $file(filename,Open) command.
$FILE(id, ReadLine)	-- Reads the next line in the file; returns 'EOF' when done. -- First argument is the file handle obtained from the $file(filename,Open) command. -- ReadLine returns "EOF" when there are no more lines to read. -- You can insert EOF as a line in a text file and the read will terminate there. -- The Readline option can be used to extract lines from a CSV file for processing with $String.
$FILE(id, Close)	-- To close an open file. (uses the file handle id from the open file command, not the filename) -- You do not need to call Close if you use ReadAll as the file will be closed automatically. -- You do not need to call Close if your ReadLine calls go all the way to the end as the file will close anyway. -- However since you don't know if it read all the way to the end it is good practice to call it anyway.

Example Subroutine to extract a particular line from a file with multiple lines

```
** GETLINEBYNUMBER extracts a specified line from a file based by on its position in the file.
** The Call to this subroutine requires 5 arguments:
** %1 is the filename to be opened
** %2 is a variable name to hold this particular file handle
**    (to ensure no conflicts with other calls to this same subroutine)
** %3 is a variable name to use as a counter to move through the required file
** %4 is a variable name to return the extracted line to the calling script for processing
** %5 is the line number containing the data required to be extracted from the file
** Example call to extract data from a file in the same folder as the layout.
** Call getlinebynumber ".\tmp.txt" myhandle mycounter reportvariable 5
** Note Fifth argument can be a literal line number or contents of a variable (e.g. @myline)
** -----------------------------
let %2 = $file(%1, Open)
if (@%2 = 0)
   echo Error: File %1 not found
endif
let %3 = 1
while (@%2 > 0)
   let %4 = $file(@%2, ReadLine)
   if (@%4 = "EOF") break; endif
   if (@%3 = %5);break;endif
   let %3 = @%3 + 1
endwhile
call $file(@%2, close)
** -----------------------------
** End of subroutine getlinebynumber
```

Example Subroutine to find the first line that starts with matching characters supplied in argument 4

```
** GETLINEBYSTART extracts a specified line from a file
** based on the first few characters on the line.
** The Call to this subroutine requires 4 arguments:
** %1 is the filename to be opened
** %2 is a variable name to hold this particular file handle
**    (to ensure no conflicts with other calls to this subroutine)
** %3 is a variable name to return the extracted line to the calling script for processing
** %4 is the opening text which is used to identify
**    the line containing the data required to be extracted from the file
**
** Example call to extract data from a file in the same folder as the layout.
** Call getlinebystart .\tmp.txt myhandle reportvariable "Peter Prunka"
** Note Fourth argument can be a literal string, contents of a variable (e.g. @myline),
**      or a train/car name ($x_train or $x_car)
** -----------------------------
let %2 = $file(%1, Open)
if (@%2 = 0)
   echo Error: File %1 not found
endif
while (@%2 > 0)
   let %3 = $file(@%2, ReadLine)
   if (@%3 = "EOF") break; endif
   if (@%3 = %4)
      if ($string(@%3,StartsWith,%4) = 1)
      break
   endif
 endwhile
call $file(@%2, close)
** -----------------------------
** End of subroutine getlinebystart
```

NOTE: A comprehensive set of ready to run File Management subroutines are now included in the Subroutine Library.

$Read and $Write are unchanged but are included here for reference

$READ(filename)	-- Returns the complete contents of the specified text file as a display string, same as $FILE(id, ReadAll) .
$WRITE(filename,[a,]string)	-- Writes the designated string to the specified file, returns 1 if successful, 0 for failure. -- If the optional argument a is included the data is appended to the file, otherwise it is overwritten.

$String is a new family of functions for manipulating string data

$STRING(string, Length)	-- Returns the length of the specified string as a number of characters same as $StrLen(string).
$STRING(string, Contains, substring)	-- Returns 1 if the string contains the substring (anywhere in the string), returns 0 if it does not.
$STRING(string, StartsWith, substring)	-- Returns 1 if the string starts with the specified substring, returns 0 if it does not.
$STRING(string, EndsWith, substring)	-- Returns 1 if the string ends with the specified substring, returns 0 if it does not.
$STRING(string, NextToken[, delims])	-- NextToken returns the next substring up to the first delimiter. -- Then it removes the extracted data (and the delimiter) from the original string. -- This leaves the modified ready for the next call which will extract the next substring. -- Default delimiter if unspecified is blank or comma. Adjacent blanks and commas count as one. -- You can specify different delimiters in the call: `let t = $string(s, NextToken, ";-/")` -- The use of @ is optional in a NextToken call. You can say (s, NextToken) or (@s, NextToken). -- NextToken returns "EOL" if there are no more tokens, but you don't have to check for this. -- You can just look at whether the string has been reduced to an empty string (see examples). -- If you use a literal instead of a variable, nothing is modified, but you can only get the 1st token.

```
EXAMPLE
let string = "token 1, token2, token3, and so on"
echo $string(@string,EndsWith,"so on")    ** Should return 1, @ before string is optional.
echo $string(@string,StartsWith,token)    ** Should return 1, @ before string is optional.
echo $string(@string,contains,token2)     ** Should return 1, @ before string is optional.

EXAMPLE
let string = "Jim, Bruno, Peter, Richard, and so on"
while (1=1)
   let token = $string(@string, NextToken,",")
   ** Note defining comma delimiter prevents spaces from separating "and so on".
   echo token => @token
   if (string = "");break; endif
endwhile

EXAMPLE (using / as delimiter)
let string = "token 1/token2/token3/and so on"
while (1=1)
let token = $string(string, NextToken,"/")
echo token=> @token
if (string = "");break; endif
endwhile
```

NOTE: A comprehensive set of ready to run String Management subroutines are now included in the Subroutine Library.

$SubStr, $FindStr and $StrLen are unchanged but included here for reference

$SUBSTR(i1,n,s)	-- Returns the substring of s starting at i1, n chars long; if n = -1, go to end,
$FINDSTR(sub,s)	-- Returns the zero-based position of sub within s, or -1 if not found.
$STRLEN(s)	-- Returns the length of string s, same as $STRING(string, Length) above.

New $System Function

$SYSTEM(command)	-- Command is any command you can type into the Windows system Command Prompt box. -- Code is executed within a command window. -- Returns an undocumented code indicating success or failure. -- If you want time to read the output from the command window you should append "& pause" EXAMPLE(s) call $system("type myfile.txt") call $system(type @file) let retcode = $system("rename abc.txt def.txt") EXAMPLE let prompt = "c:test.xml" let code = $system(@prompt) if (code = 0) echo file exists! Else echo file not found endif

New $Train Function

$TRAIN(id,Flip)	-- Call $train(es40, Flip) - -- Picks the train up off the tracks and turns it around to face in the other direction.

New Read/Write Functions

While most functions remain "read only" because they are being continually updated by the system there are now many functions that can be set to new values from within a script. The functions that can be reset in this way are all designated as r/w in the ref tab of Script Central.

To use a r/w function to set a value, just call it with SET or LET and follow with a value.

FUNCTION	EXAMPLE
$KEY	**Let $key = 0** -- Ensures $key is empty so it can be continually monitored for another value while a script is looping. -- This prevents loss of train monitoring control which would occur if the script were stopped with ON KEY
$LAYOUT **$LAYOUT(Name)**	**Set $layout "My Layout"** -- Assigns a new name to the currently selected layout. **Let $Layout(Name) = "New Layout" -** Assigns a new name to the currently selected layout.
$SWITCH(id)	**Set $switch(56) 1** -- Is the same as throw 56 1
$SETTING(regvar)	**Let $setting(maxMPH) = 40** -
$TRAIN(id,Name)	**Let $train("train1", Name) = "Broadway Limited"** -- First argument can be a user assigned name or the train<id>
$TRAIN(id,Speed)	**Set $train(@tname, Speed) 20** -- Set or change the speed of a named train, this need not necessarily the train running the script
$TRAIN(id,Direction)	**Let $train(name, Direction) = F** -- Value should be F (Forward), R (Reverse) or T (Toggle).
$CAR(id,Label)	**Let $car(X14,Label) = X188** -- Change the label on car X14 to read X188 **Let $car(@car,Label) = A** -- Change the label on the car whose name is in variable car to read A.
$CAR(id,Note)	**Let $car(X188,Note) = "This car is heading for Chicago"** -- Places the designated text in the Car Note field.
$CAR(id,ExcludeOps)	**Let $car(RA15,ExcludeOps) = 1** -- Exclude the designated car from being selected by the Ops Generator. **Let $car(RA15,ExcludeOps) = 0** -- Allow the designated car to be selected by the Ops Generator.
$CAR(id,Loadname)	**Let $car(HT12,Loadname) = Aggregates** -- Change the default load for car HT1 to Aggregates.
$CAR(id,Loaded)	**Let $car(HT12,Loaded) = 0** -- Unload the specified car -- same action as Unload(HT12) **Let $car(HT12,Loaded) = 1** -- Load the specified car with its default load -- same as Load(HT12)
$CAR(id,RevEngine)	**Let $car(ED2,RevEngine) = 1** -- Set the Reverse Engine flag on on engine ED2 **Let $car(ED2,RevEngine) = 0** -- Cancel the Reverse Engine flag on engine ED2

User Defined Train and Car Properties

You can now define new fields (or properties) for both cars and trains, these can have any user defined name and be used for a variety of purposes, such as carrying a train timetable, or recording the use of water as an engine progresses on its journey. The information stored in these user defined fields cannot be seen in Train/Car props but they can be identified and extracted for processing using additional Script commands. These properties stay with the car until removed, they are saved with the layout and will travel with a car across linked layouts.

FUNCTION	EXAMPLE
$TRAIN(id,anyname)	**Let $train(@train,timetable) = "0800,0900,thru,1100"** Creates a new property/field on the train whose name is stored in the specified variable and allocates a timetable to the train which can be processed by scripts as the train progresses on its journey. Use **Let $train(@train,timetable) = ""** to delete this property from the train.
$CAR(id,anyname)	**Let $car(X14,ReturnMTto) = "Moth Lake Yard"** Creates a new property/field on the car with the name "ReturnMTto" and stores the value "Moth Lake Yard" in it. Use **Let $car(X14,ReturnMTto) = ""** to delete this property from the train.

Notes compiled from information supplied by Jim Dill during the development phase of Trainplayer v6.1

Richard Fletcher, November 2014