UNDER DOG SOFT WARE

STM USER GUIDE

OFFLINE DEVELOPMENT TOOL (ODT) GRAPHICAL USER INTERFACE (GUI)

FOR STM VERSION 2.4

Authority	Name	Signature	Date
Author -	A Smith		02/11/2015
Reviewed -			

UNDER DOG SOFT WARE

TABLE OF CONTENTS

1. Document Introduction

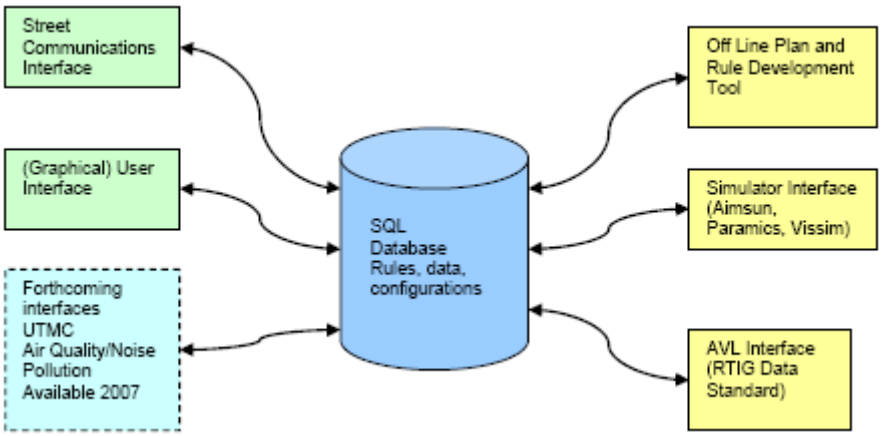
1.1 Overview of Project

STM is the Strategic Traffic Management tool. It is designed to provide a fully scalable network monitoring and control tool for small towns, major arterial roads through to city wide UTC.

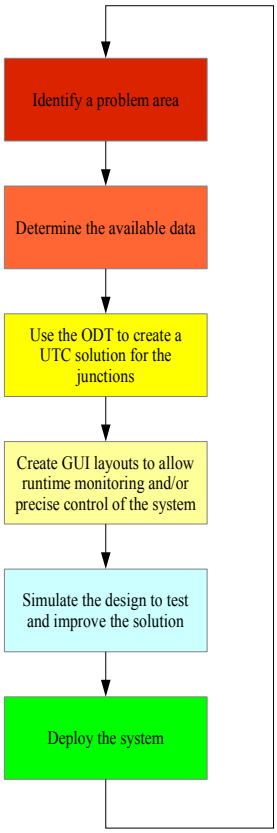
At its heart is an SQL database. This stores; timing plans for the junctions under control, all the data recovered from street, the logical rules that determine which plans are to be applied under any given set of conditions and the setup information for the user interfaces. This allows the entire DB to copied offline for security, archive and simulation purposes.

STM interfaces to the street devices via the existialso provides the user with a GUI to monitor runtime operation, an interface for AVL data, an interface to simulation tools and the ODT. The last two facilities allow new plans and control rules to be designed and tested with simulations before being transferred to the online database for real deployment.

The STM system is split into several components which fall roughly into 2 main groups, the user interface (ODT & GUI) and the core system (core, Database, UTC interface, AVL & Emulator). The system is designed to monitor and/or control any UTC system with a centralised control point. It does this by implementing a set of code sheets (known as MERV code) onto a runtime server that is connected directly to the UTC network. The MERV code can be extended by the user to solve any traffic engineering problem.



1.2 Developing a UTC Solution using STM



Once an area has been identified as needing STM the particular traffic solution is implemented in MERV code with accompanying user layouts. This is then compiled and commit into the SQL database. The core system loads the compiled MERV code from the database and processes it, which instructs it to monitor for changes to various inputs from street and make appropriate changes to outputs back to street thus implementing the defined traffic control measures. User control and monitoring of the live system is done using the GUI, which loads the layouts defined in the ODT and displays them to a user. A layout allows the user to see and control the state of the STM system. The GUI allows users to view the state of any part of the system or control it. E.g. manually extending morning peak.

2. ODT User Interface

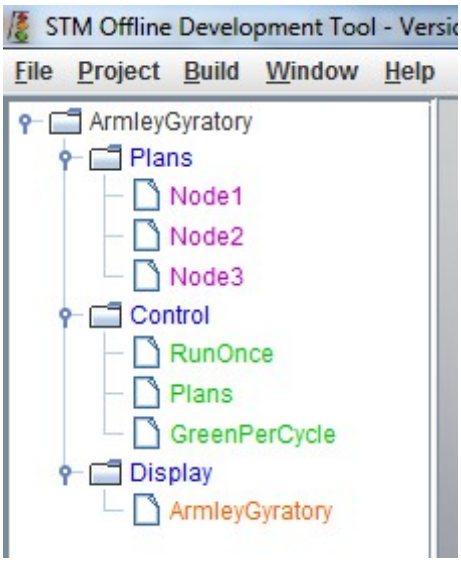
2.1 Overview

Offline Development Tool - is the tool a developer uses to create MERV code and layouts to solve traffic engineering problems. The ODT has 4 types of sheets Code, Function, Plan and Layout. The Code and Function sheets hold the MERV code instructions that define the control part of the system. Plan sheets store predefined plans that control a specific junction in a known way. The Layouts define the interface to the MERV code logic that other users (most likely UTC personnel) can use to adjust the behaviour of the system at runtime (see section 2.3 for examples of these sheets).

All text in MERV code is case insensitive such that the cell reference @a1 is equivalent to @A1, and runplan = RUNPLAN = RunPlan = RuNpLaN. It is however highly recommended to use sentence case for functions e.g. RunPlan and capitals for variables e.g. PLAN_NUMBER as this greatly improves the readability of the MERV code (For the details of MERV code see section 3.1).

The main aim of MERV code is to process data coming from street e.g.. detector bits, traffic flow rates etc and make a decision as to how to modify the street node plans so as to give the desired results. This can be with respect to anything the developer wishes to optimise e.g. bus priority, general traffic flow or air quality.

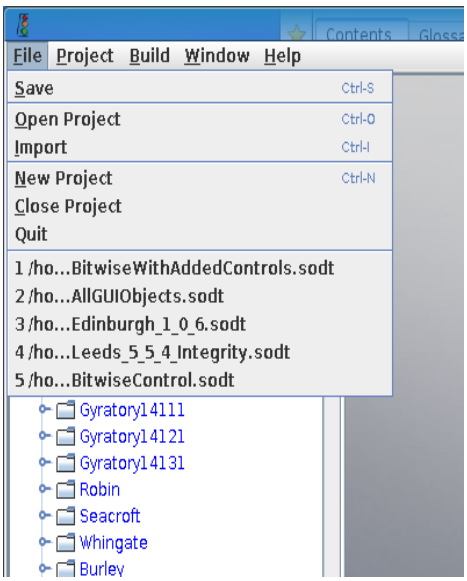
When designing an STM system it should be noted that the Namespace window can hold multiple namespaces (displayed in blue) within the project. Care should be taken to choose logical groupings of objects so that other developers can easily find sections of MERV code that either need updating or correcting.



2.2 Menus

2.2.1 File Menu

The file menu is used for normal file operations as well as creating new projects. Added are the keyboard short cuts for quick access. Below the Close Project is a list of recently opened SODT files. ODT saves the last 5 files opened. Hover over the file name to view full directory path.



Save

Saves the current project. The default name selected for the filename is the project name. The default location is the directory is the directory the ODT was launched in. All ODT files have the .sodt extension placed on them. The save process will automatically create a backup file if the filename already exists.

Open Project

Opens a project that has been previously saved by the ODT. By default this will only display .sodt files. If there is already a project loaded and unsaved the user will be warned before a new project is opened.

Import

Imports an ODT file into the current project. A project can even import itself, new Sheets will all be given unique names.

New Project

Creates a new project and calls it Project1. If a project is already loaded the user will be warned before this operation completes that they may lose their data.

Close Project

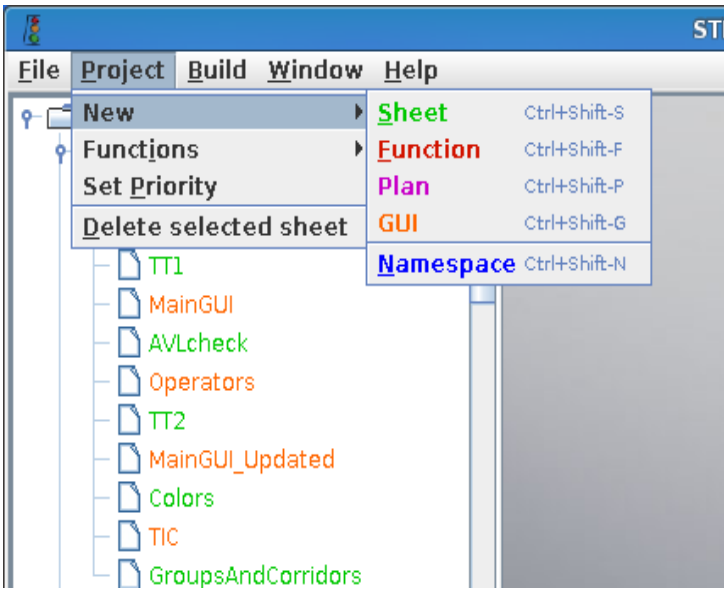
Closes the currently opened project. If the current project is not saved then the user is warned that they may lose data. Closing the ODT will automatically call this close procedure.

Quit

Closes the application. If there are unsaved changes in the project you will be prompted.

2.2.2 Project Menu

New Sub Menu



Sheet

Creates a new sheet in the currently selected namespace. Sheets contain the MERV code that controls the SCNs by calling the RunPlan function. This option is not available unless the user has selected a namespace to place the new sheet. The new sheet will be given a unique name starting with the word "sheet" and followed by the next unused number in sequence.

Function

Creates a function in the currently selected namespace. This is greyed out if the user has not selected a namespace. The new function will be given a unique name starting with the name "Function". The first row of a function is not editable as this is the input line from the calling sheet. To use a function it must be selected from the function list (see section).

Plan

Creates a new plan in the currently selected Namespace starting with the text "Plan". A plan is a 3 column table, the first column defines the change times, the second defines the bit patterns to be output at particular change times, and the third is for comment. Plans also have fields for the node they are to control and their unique plan number. Plans are used by calling them from a sheet using the runplan() function.

GUI

Creates a GUI layout in the currently selected Namespace. The layout is similar to a simple drawing package. Components for a layout are added then connected to MERV code variables (AutoVars)

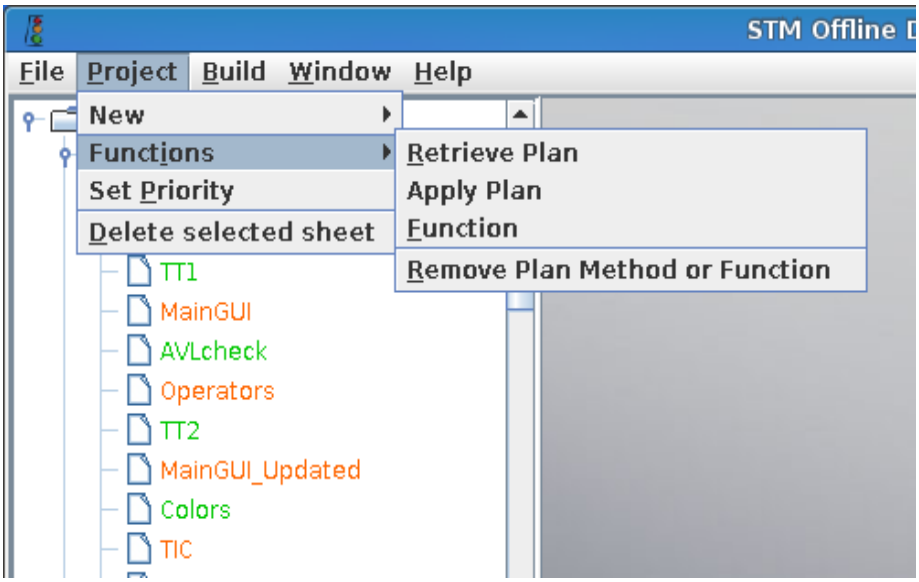
defined in the Sheets. Some GUI components can set and get values in the system others can only get values.

Namespace

A Namespace is a logical grouping for a collection of sheets and/or layouts and is always uniquely named. Layouts are always placed in the same layer of the namespace window and therefore can always be created. There is no restriction on how many or their use, though it is strongly recommended that items in a namespace share some commonality. An object in a namespace must have a unique name within that namespace, but objects in other namespaces can have the same name i.e. It is legal to have Namespace1.inputs and Namespace2.inputs, but not N1.output and N1.output.

Sheets, functions, plans, GUIs and namespaces may be renamed from the default by selecting them and pressing F2.

Functions Submenu



2.2.2.1 Retrieve Plan

This provides a means of identifying the details of a specific plan. The method can be placed in either normal sheets or function sheets. To place a retrieve plan method the user must first select a cell in the A column. When Retrieve Plan is selected the ODT inserts 2 lines horizontally across the page (overwriting the contents of the 52 cells). The first cells in each row are purple denoting that they are editable, and the rest are red denoting they are 'output only' cells i.e. can only be referenced not edited. Retrieve Plan gets the plan specified by the node defined in the first editable cell [Node] and the plan number set in the second editable cell [Plan #]. At runtime the red cells are populated with the contents of the specified plan.

[Node]	Plan Points	Time 1	Time 2	…	Time n
[Plan #]	Cycle Time	Pattern 1	Pattern 2	…	Pattern n

2.2.2.2 Apply Plan

This is similar to Retrieve Plan, in that it must be located in column A. However all the cells are editable as this function will update the specified plan with the data contained in the cells.

Node	Plan Points	Time 1	Time 2	…	Time n
Plan #	Cycle Time	Pattern 1	Pattern 2	…	Pattern n

Note that due to the dynamic nature of these plans and the fact that the plan number may contain a reference to another cell the ODT is unable to perform duplicate plan checking in the way that it does for static plans.

Care must be taken to de-conflict static and dynamic plan numbers for each node. Failure to do so will lead to undefined behaviour at run time.

2.2.2.3 Function

To make a function call to a user defined sheet you must first define the function to call (see section 2.3.3). Next a cell in column A of a sheet must be selected. To enter a function, simply select it from the list of all available functions in the system. Functions are listed by their full ‘dot separated’ name to help identify them. The first cell has the name of the function selected. The remainder of the cells (marked in green) are the input cells to the function. At runtime the value of these cells is copied to the function, the function is executed and the processed values are placed into the Output cells (marked in red). Functions can be called as many times as required from any sheet, but they can only be called from sheets as functions cannot be nested.

Function Name	Input 1	Input 2	…	Input n
Output 1	Output2	Output 3	…	Output n+1

2.2.2.4 Remove Plan Method or Function

The only way to remove a function or method call is to select the first cell in the A column and use this remove method. None of the data in the cell will be preserved.

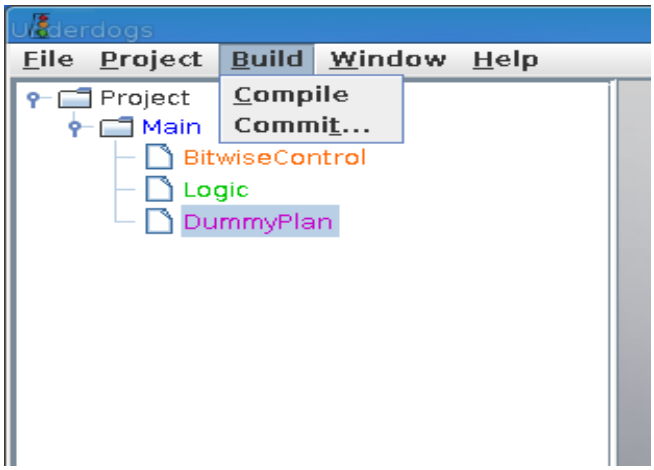
Set Priority

Each sheet has a priority assigned to it, set to 50 by default. A priority is a value between 1 and 100 inclusive where 1 is the highest priority and 100 the lowest. The sheets are executed from the highest priority to the lowest and all sheets with the same priority are executed at the same time (the system does define an order, but a developer can only control the order of execution of a series of sheets by changing their priority). Variables defined on a sheet do not have their values updated until the end of the priority level? that they are in. For more information see section .

Delete Selected Sheet (also keyboard DELETE)

Deletes the currently selected sheet or namespace from the tree view. The user is warned before this operation is executed. When deleting a namespace, all the contents of the namespace is also deleted. Highlight the sheet or namespace and press the DELETE button on the keyboard.

2.2.3 Build Menu

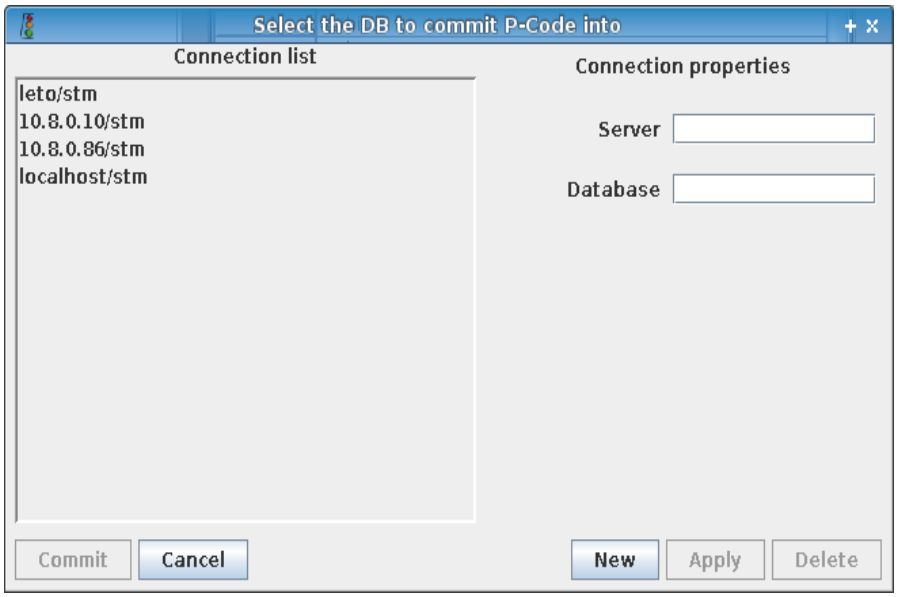


Compile

Compiles all the sheets in the current project. This process verifies the code to check for basic syntactical errors then internally generates byte code from the MERV code. If there are any errors in the current sheet then the compile process is stopped after the remainder of the sheet has been processed. (For a full list of errors see section 3.3.1).

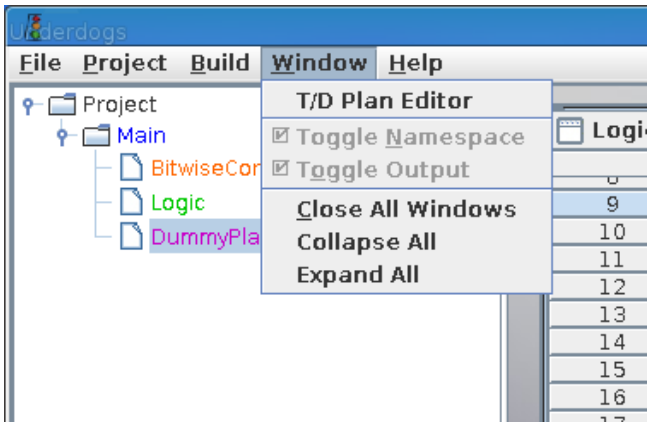
Commit

After a successful compile the code can be uploaded to the database. The uploaded code is the runtime byte code generated during the compile process. When committing the location of the database servert and the database name must be entered. This information is stored for future use.



Clicking New will create a new entry in the connection list. These entries are stored on the hard drive and will be available next time the ODT is loaded. When an item in the list is selected, its values are filled out into the Connection Properties area. If they need to be modified then these changes will only take effect when the Apply button is pressed. Old connection information can be deleted by selecting it in the list and pressing delete. Once the correct information has been entered it can be entered into the DB (ready for the runtime system) by pressing Commit.

2.2.4 Window Menu



T/D Plan Editor

Displays the Time Distance diagram editor. The TDDE Editor allows a user to load and manipulate plans in order to help investigate plan changes, and also to generate new plans that are offset from the original. This can be useful when implementing hurry calls and extension timers.

Toggle Namespace (Currently Disabled)

This check box removes or replaces the Namespace window. This is generally used once the namespace window has been closed. The Namespace window displays all the Namespaces and their contents in a standard tree view. Double clicking an object in the tree view opens it in the main area and pressing F2 allows the user to edit the name of that element

Toggle Output (Currently Disabled)

This toggles the output window. The output window is used to display messages to the user including the state of the system and errors during compilation. Errors can be double clicked to open the sheet that they appear in. The output window is cleared when a compile is started.

Close All Windows

Closes all the sheets, GUIs, and plans open in the workspace area of the ODT.

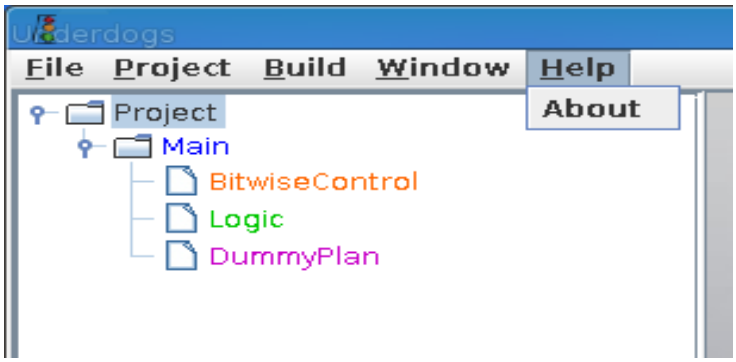
Collapse All

Collapses the namespace tree.

Expand All

Expands the namespace tree.

2.2.5 Help Menu



About

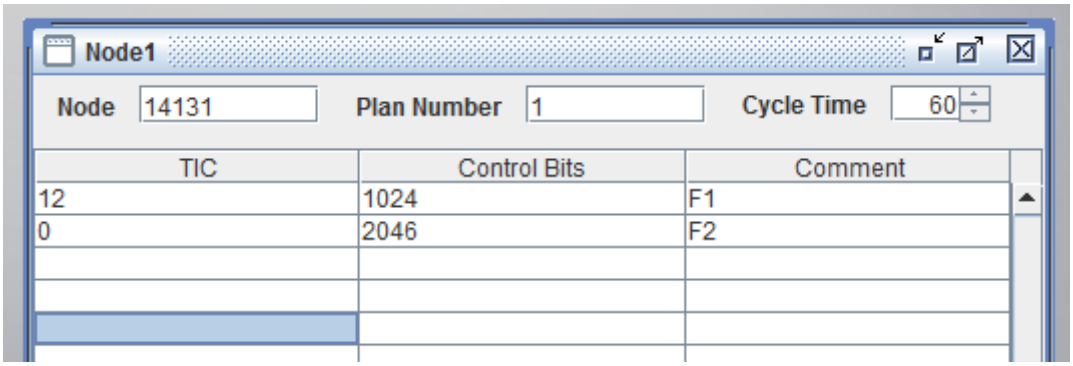
Shows information about the ODT, most importantly its version

2.3 Entering Data into Sheets

Sheet data is entered by the keyboard. When a cell is selected it goes dark blue, select a cell and begin typing to append the text to the current contents of the cell. When a cell is being edited it is displayed in yellow. Pressing the escape key will cancel the edit removing any changes made by the user for that cell. Use Ctrl+C, Ctrl+X and Ctrl+V to copy, cut and paste respectively when editing within a cell. Multiple cells can be copied using 'select' and the 'copy, paste' options using the RMB.

The 3 types of sheet containing tables (Sheet, Function & Plan) are all edited in the same way, but the execution of each in the core system is very different and this influences the way that they are written.

2.3.1 Plans

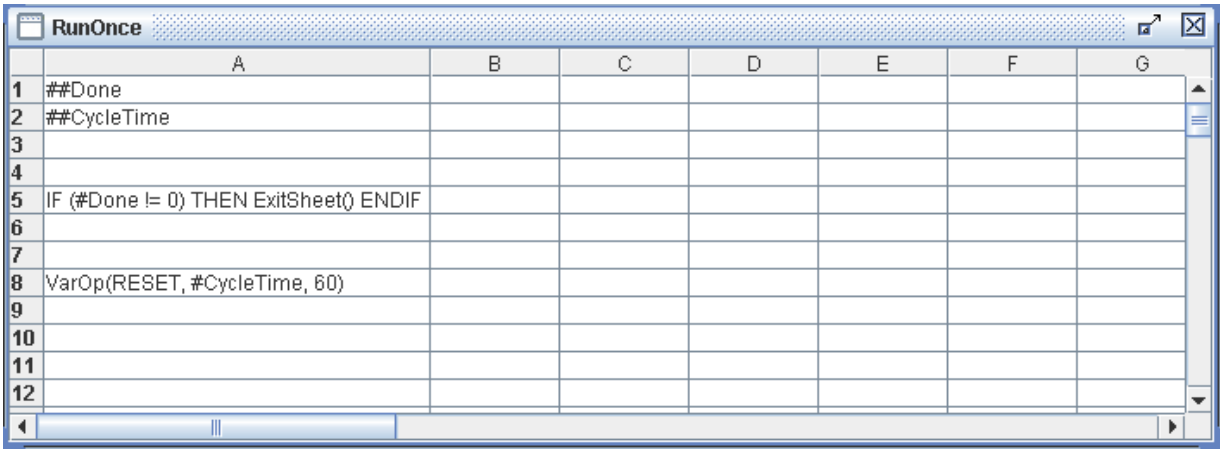


The simplest table sheet is the plan. Plans are loaded at the start of the system and can only be modified by the Apply Plan function. A plan is run using the RunPlan function and every node to be controlled should have a plan run on it every STM cycle (in real time this means every second), failure to do so will cause the system to give up control of that particular node. Plans are defined as a sequence of change times (TIC) verses bit pattern (Control Bits) pairs – currently defined as a binary pattern (e.g. 1024) but eventually to be defined by control bit name (e.g. F1). Each line in the plan can also have a comment. When a plan is executed the bit pattern associated with the change time equal to or less than the current cycle time (CT) of the node is the pattern that is sent to street for that node. All plans have a CT which marks the end of the plan. When the time in cycle (TIC) reaches the CT it is immediately reset to 0, thus the TIC is never equal to the CT. When constructing a plan the first blank

cell in either the TIC or the Control Bits column marks the end of the plan. A warning is raised if there is data after this point.

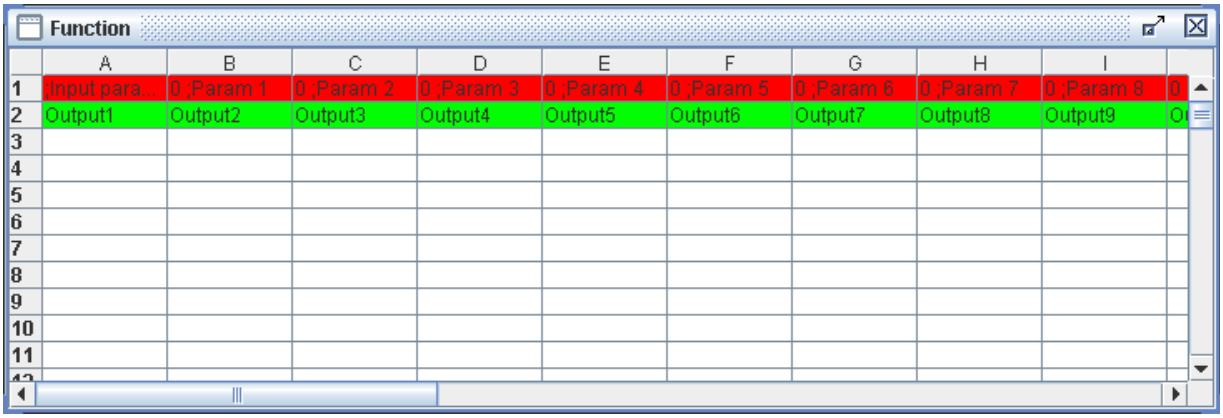
2.3.2 MERV Code Sheets

The main flow of the system is controlled using MERV code in sheets.



Sheets are designed to be similar to Excel spread sheets in look and feel as this was the how a prototype to the STM system was written. As such there are a number of cells that can each contain a small section of code. The cells are not necessarily executed in a defined order as references cause the execution order to vary, however in the absence of references the cells are executed left to right, row by row, starting from cell A1. An important part of a sheet is its priority. The priority of a sheet will determine when is gets executed and this influences the state of the variables it can access (see section 3.1).

2.3.3 User Functions



A function is used to collect a sequence of logic into a simple interface. A function can only be used on a sheet. It is only processed as a sequence of references, there is no default sequence. When a sheet references any output cell of a function it causes the following sequence of events:

1. All the input cells to the function on the MERV code sheet are calculated

2. The input cell values are copied from the sheet to the function (row 1 in red)

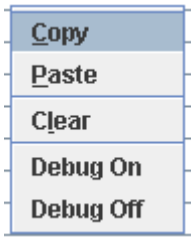
3. The output row cells of the function are calculated causing a chain of references within the function (row 2 in green).

4. The output row of the function is copied back to the calling sheet.

All functions are global so several sheets may use the same function. Functions may not reference AutoVar variables however, unlike most programming languages, these functions have multiple output parameters so variables can be set once the function has returned. An example function is given in section .

2.3.4 Right Mouse Button Menu

The RMB menu is available for any cell based sheet (i.e. Sheets, Functions and Plans). The selected option is generally applied to the selected cells starting from the top left.



Copy

Select any group of cells on any of the cell based sheets then select copy. These cells are copied to an internal buffer ready for pasting. Note: the copy is disabled when using the multiple selection tool.

Paste

Select the top left corner of where you would like the copied cells to be pasted and then select paste. The contents of the copied area will be pasted into the new location starting from the marked cell. Note: the paste is disabled when using the multiple selection tool.

Clear

Removes all the formatting and content from all the cells in the selected area. Function cells remain as function cells, but their contents and debug state is cleared.

Debug On

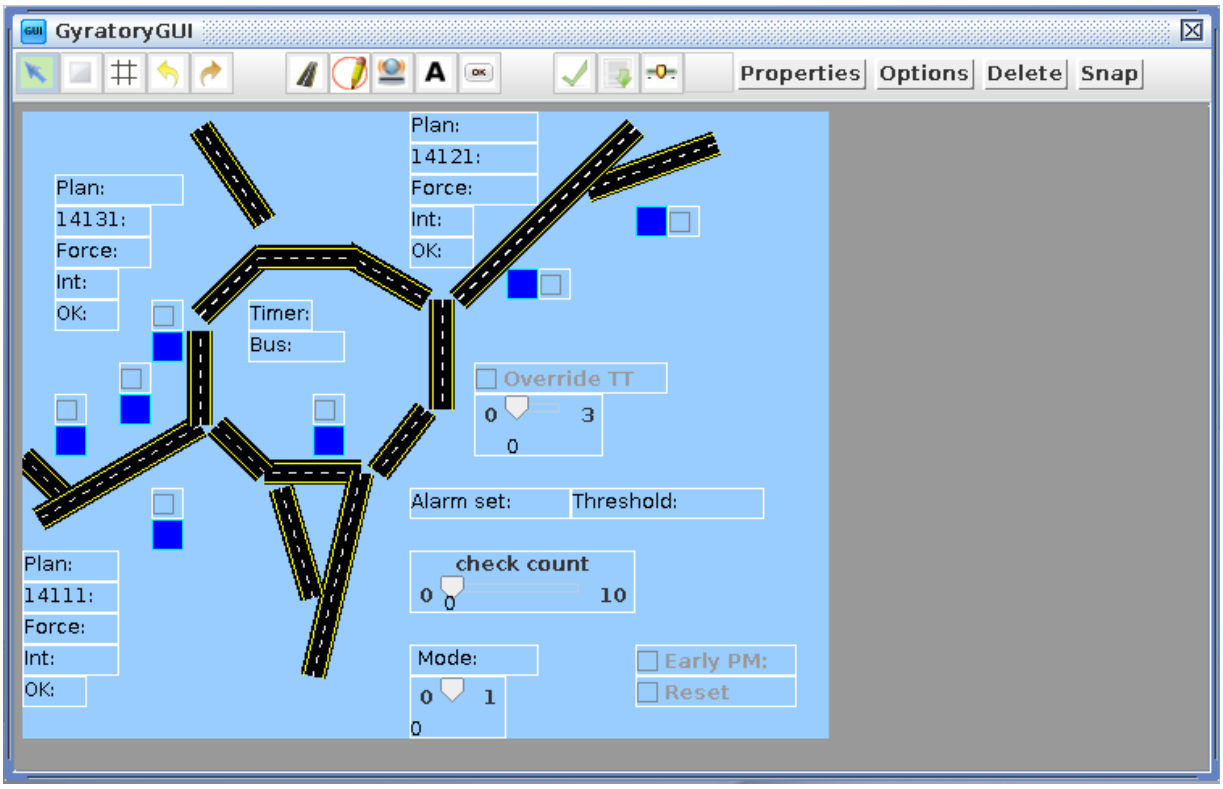
Turns on debugging for the selected cell or cells. Debugging can only be selected for cells that have content. Each of the debugging cells turns light blue to show that debug is ‘on’. Debugging has the effect of adding many more trace lines to the output of the core system (specifically the SSB module), therefore it is not recommended that is it left on by default as the amount of trace produced can impact on the responsiveness of the main system

Debug Off

Turns off debugging for the selected cells. The cell is returned to its normal colour and trace will no longer be generated by the core system for that cell. Adding or removing debug cells must be recompiled and committed to the relevant DB to take effect.

2.4 Entering Data to Layouts

Layouts define what a user of the STM GUI will be able to monitor and control. All input and output to the Layouts must be done through AutoVars. To access the properties of a component, first it must be selected then the Properties button can be clicked.



2.4.1 Tools

The various tools that allow a developer to create and manipulate GUI layouts to match the needs of the GUI users.

Select 

This is the default tool. It allows the developer to select, move and resize objects using the mouse. When an object is selected its outline is yellow, when it is resizing its outline is red. To resize an object click near the top left or bottom right of an object, in the case of lines click at the end of the line. Clicking at the middle of an object will allow it to be moved.

Multiple Select 

Selects multiple GUI objects in the work area. This will also move multiple items at the same time.

Show/Hide Grid 

Shows or hides the grid on the GUI work area. Click on properties to set grid spacing.

Undo/Redo 

Undo/Redo previous action performed, such as a move or delete. Use keyboard shortcuts Ctrl + Z for Undo and Ctrl + Y for Redo.

Circle

Use the circle tool to create a circle on the GUI. Default size is 100 pixels. Right-click on the circle to change dimensions and line thickness.

Lane / Line 

Click and drag to place the lane. By default this tool produces a 2 lane black road, however the properties for this object allow it to simply be a line, or have many more lanes, or be a different colour. Lines have no active component to them, so only act as a visual guide for STM GUI users.

Detector 

This is the simplest layout component. At runtime it displays cyan if the referenced value is greater than 0, otherwise it displays dark blue.

Text Box 

Text boxes have the largest number of connections to the runtime system. They are mainly intended to display the value of a particular AutoVar (an AutoVar can also contain a fixed string), however there are 3 other connections to AutoVars. The first is the visibility of the text box - if the referenced variable is 0 then the text box is not visible. The last 2 options are the text and background colour. These values are set in the MERV code and are based on the value 0x00RRGGBB where RR is a value for the red component, GG is the green and BB is the blue. The following table shows the decimal values that would be typed into a sheet to get the desired colours. The colours must be assigned to an AutoVar for the GUI to be able to use them.

Colour	Hex Value (00 RR GG BB)	Decimal value
Red	00 FF 00 00	16,711,680
Green	00 00 FF 00	65,280
Blue	00 00 00 FF	255
Amber	00 FF FD 40	25,524,064
Violet	00 C0 9B C6	192,155,198

These colours can be calculated rather than being specified so, for example, a gradual shift from Green to Red can be produced dependant on the value of a variable. See example function GetWarningColour section

For flashing colours, the GUI uses the unused 4th byte. See the following table:

Flashing Colour	Hex Value (00 RR GG BB)	Decimal value
Flashing Red	01 FF 00 00	33,488,896
Flashing Green	01 00 FF 00	16,842,496
Flashing Blue	01 00 00 FF	16,777,471
Flashing Amber	01 FF FD 40	33,553,728
Flashing Violet	01 C0 9B C6	29,400,006

Background colours are set by the same mechanism

During the design process text boxes are surrounded by a white border for ease of positioning. This will not display on the final GUI.

Check Box 

A check box is similar to a detector in that it can show the state of a bit in the system. The major difference is that the bit is also controllable by the check box. There is no way to set the initial value of a check box from the GUI. The value is always taken from the MERV code. This is because more than 1 GUI may be connected at the same time and they could both access the same variables. The GUI will always show the current value of variables in the system, so if one user changes the variable this change will be displayed on the other system. A check box can also be made invisible, which allows a MERV code developer to implement a way of preventing repeated changes to the system before the system has had a chance to respond. e.g.. the check box may cause an green extension, but switching the check box on and off repeatedly would cause undesirable effects so it is set to invisible (and 'non modifiable') as soon as it is clicked and made visible when another extension is allowed.

During the design process check boxes are surrounded by a white border for ease of positioning. This will not display on the final GUI.

Combo Box 

Combo boxes allow a series of names to be attached to values. This allows the developer to create a set of human readable values that can be selected by a GUI user. E.g. "Short Delay" = 3, "Normal Delay" = 5, "Long Delay = 10". These values are then passed to the specified AutoVar when the user selects it. The combo box can also be made to be invisible and non modifiable.

Slider 

A slider bar allows a user to pick a value from a range of values. The maximum and minimum values can be set by the developer as well as the AutoVar that will be accessed by the slider bar. When the slider bar updates it set the AutoVar to its new value, and when the AutoVar updates it tells the slider bar to update. Slider bars can also be made invisible and non modifiable.

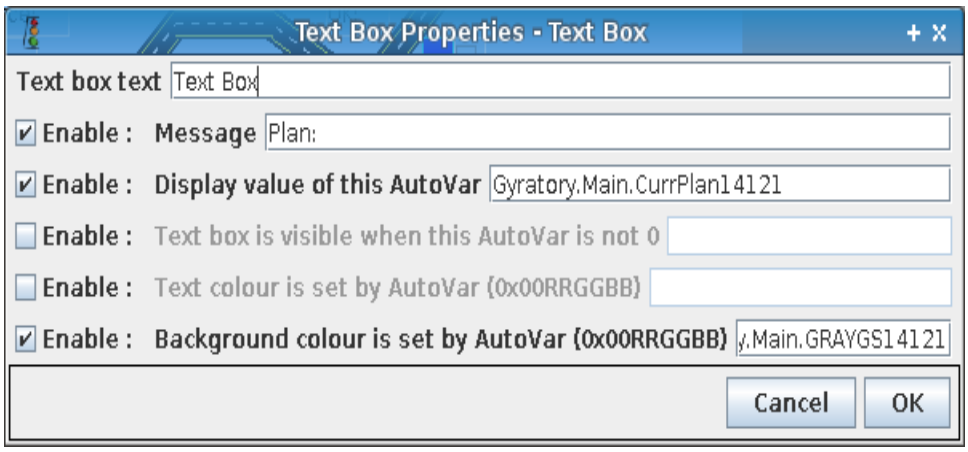
During the design process sliders are surrounded by a white border for ease of positioning. This will not display on the final GUI.

Button 

GUI 'pushbutton' object (i.e. ON only when pressed). The action of manually pressing a button could set a variable, which resets in the first 'second' following release of the button.

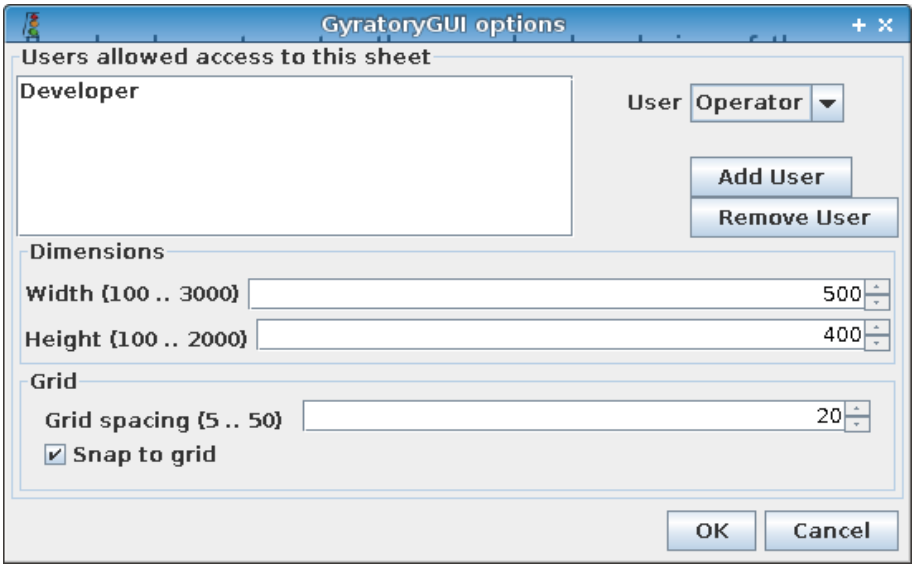
2.4.2 Properties

When an object is selected it will have properties. Each of the properties is different for each type of object, but they all allow the developer to set the name of the object. The name is generally only used when there is a compilation error. The properties allow the user to setup the specific connections for this object and other details of the object.



2.4.3 Options

The options dialog allows the developer to setup the user level and size of the current layout. The user level is particularly important as the default is 'Developer' level. This means that only developers are allowed to see this layout by the runtime GUI. This is done so that newly designed and potentially untested layouts do not immediately come under general use. The list of names of users and their assigned user level is defined in the database table gui_users.



2.4.4 Delete

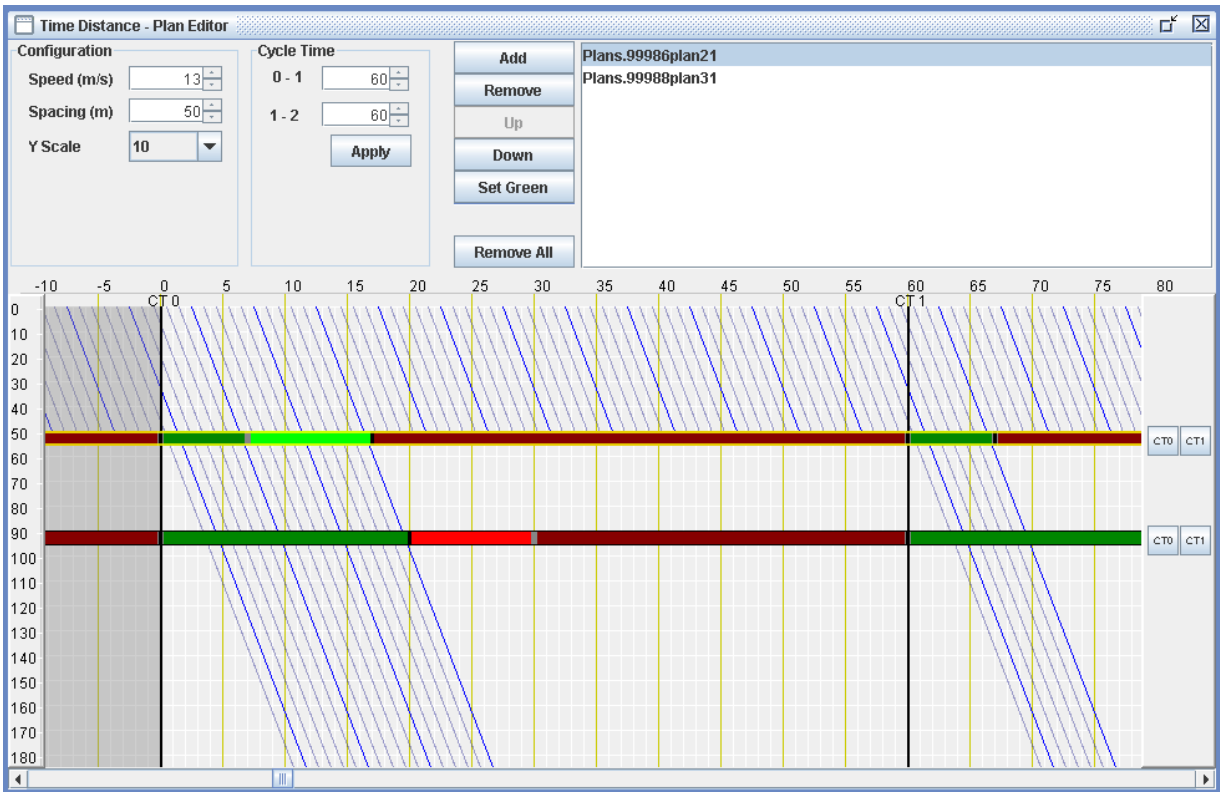
Deletes the currently selected item. Use Ctrl + D for keyboard shortcut.

2.4.5 Snap

This causes the currently selected object(s) to snap to grid regardless of the setting in the options dialog.

2.5 Time Distance Diagram Editor

The TDDE is a tool that is designed to aid developers in investigating and creating new plans from current plans. This can help identify coordination through a series of nodes in relation to modified plan timings under consideration.



2.5.1 Buttons

Add

When the add button is pressed a list of all the plans in the system is displayed in a list. Any plan can be chosen to be added to the TDDE. When a plan is selected it is shown in the display area.

Remove

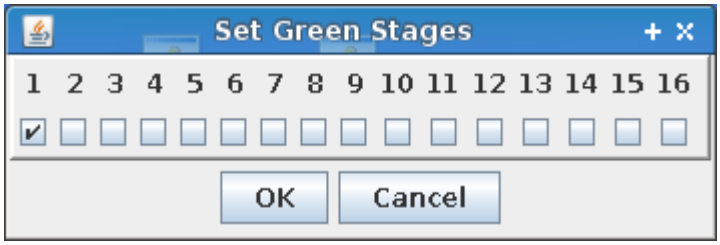
Plans in the display area can be selected, as can plans in the list. The selected plan is denoted by being highlighted in the list and bordered in yellow in the display area. When remove is pressed, the selected plan is removed.

Up / Down

Moves the currently selected plan up or down in the list and hence the display area. This is important if the TDDE is being used to display the coordination through a series of junctions (as above).

Set Green

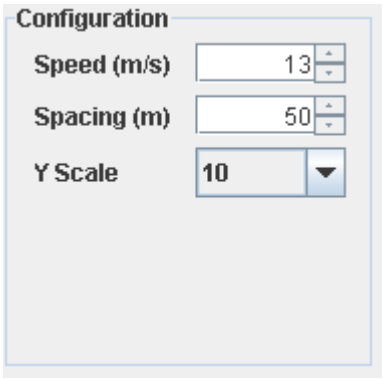
This allows the developer to set which SCN bits relate to which stage. If the flags that are mentioned in the Set Green table match the bits in the plan, then that section will be displayed in green and will allow the coordination ? lines to pass through.



Remove All

Removes all the plans in the list.

2.5.2 Configuration



Speed (m/s)

The lines drawn in the display area denote the position of a vehicle entering the junction at a specific time and travelling at a specific speed. Each line represents a different vehicle entering the junction 1 second apart. The default speed is 13m/s (29 mph).

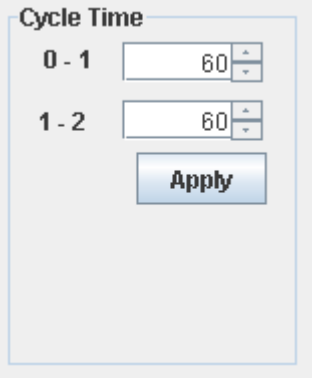
Spacing (m)

This sets the distance between the currently selected node and the upstream node (or other defined point). In general this is designed as a way of giving the developer a better representation of where the vehicle will arrive during the plan. e.g. if a detector is 100m upstream of the stop line, then the spacing before the first junction can be set to 100m.

Y Scale

When node spacings are large, it will often be necessary to change the scale to be able to see the plans on one screen.

2.5.3 Cycle Time



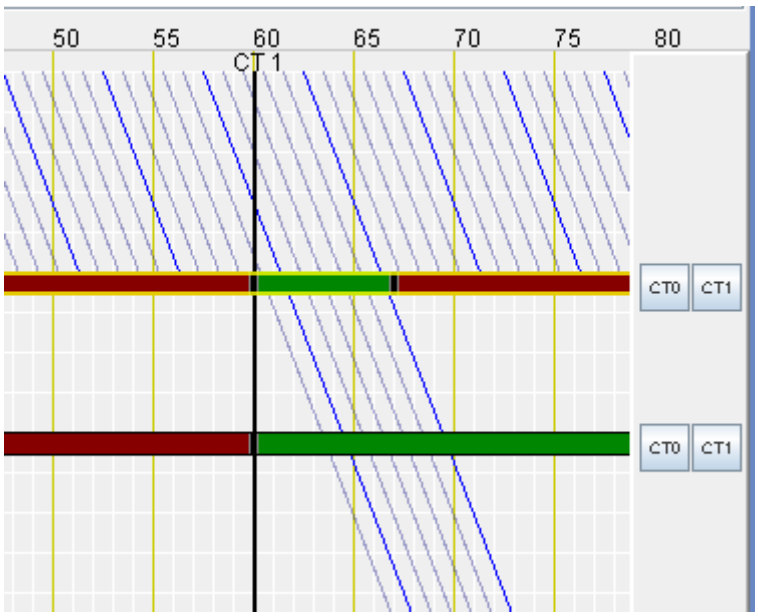
The nodes are displayed vertically, but they are repeated horizontally at least 4 times (more if a plan with a short cycle time is displayed with a plan with a long cycle time). The first cycle of every plan is always grey as this shows the default timings, whereas the next 2 cycles are editable as these are the cycles where user changes can take place. The last cycle is also grey to show what the plan will revert to. The cycle time of the 1st and 2nd editable cycles is variable. To change the value simply press 'apply'. It is not possible to add or remove plan points in the TDDE, so if an attempt is made to reduce the cycle time below the value of any plan point, then the value of that plan point, and subsequent plan poits, will reduced accordingly.(subject to a 3 second min green time per stage). If time is added to the cycle time then it is only added to the last plan point.

2.5.4 Active Display

The active display allows a developer to investigate the timings of plans to better see how vehicles will travel through a set of nodes. Each plan point in a plan can be moved using the mouse. A bright colour will denote where the change has been made. If the change is green then any coordination lines will pass through it. When satisfied the developer can generate the plans they have manipulated.

CT0 / CT1

The buttons to the right of each plan line allows a developer to create a new plan from either the first cycle or the second cycle. The plan is added to the same namespace where the original was located and is named with the original name plus an extension, either CT0 or CT1 depending on the button pressed.



3. ODT Development

3.1 Writing MERV Code

3.1.1 Overview

The ODT compiles the MERV code into P-Code which is then sent to a DB for later execution. During the compile process the ODT will attempt to identify any programmatical errors the user may have made including syntax errors (i.e. wrong number of parameters to a function) and logical errors (i.e. referencing an empty cell). There is also some provision for debugging, but as the ODT is an offline tool, this is generally done by the user as an analysis process on collected data, not at runtime.

An important part of the MERV code to understand is the AutoVars (Automatic Variables). They are special variables that allow the transfer of data between sheets, and which can also count' in the background. Because of the way the cells are executed, it is important that variables cannot change until all cells referencing them have been calculated (otherwise the value would be dependant on the execution order which the user has little or no control over). Generally in cell based languages such as Excel this is not a problem as only the current cell is being calculated and so as each cell references another cell a chain can be followed until a point where a constant is reached. Then the values can be finalised as the chain is followed back to the current cell. E.g.

	A	B	C	D
1	1	@A1 + 1	@B1 + 1	@C1 + 1
2				
3	##anAutoVar	@A3 + 1	@B3 + 1	@C3 + 1
4				
5				

To work out what cell D1 is equal to first C1 must be calculated. This in turn needs the value of cell B1, which needs A1 where the constant 1 is returned. The process can then 'un-wrap' giving 2 for B1, 3 for C1, which finally allows D1 to be evaluated as 4. If at some point later B1 were to be referenced then its value would have already been calculated as 2 and so that would be returned immediately. An AutoVar can have its value set by another cell, which means that the result may no longer be true. E.g. In cell A3 if the AutoVar could change from one call to the next then the values of A3, B3, C3 and D3 would also change leaving no way to be sure of the value of a cell when writing MERV code. Hence, in order to ensure that a cell's value is fixed for each cycle, the AutoVar will only get updated after all the calculations on that sheet have been completed and all the sheets with the same priority level have been completed (see section).

3.1.2 Basic Maths Operations

Overview

Each cell can contain a collection of operations and conditions so long as there is only 1 result. The compiler obeys the rule of BIDMAS (Brackets, Indices, Division, Multiplication, Addition, Subtraction) so that 2 + 3 * 4 = 14 = 4 * 3 + 2, whereas (2 + 3) * 4 = 20.

Strings

The ODT allows simple text in fixed strings to be placed into the sheets. Strings are required to define SCNs and can be used to report messages back to the user via the GUI. A string is defined by placing any sequence of characters in quotation marks. If a string is found where a value is expected then an error is reported by the core system at runtime and the value 0 is used in place of the string. There are no functions to manipulate, convert or interrogate strings - they can only be used as values to describe

SCNs or as messages. An AutoVar can store either a string or number, the AutoVar will work this out dynamically.

3.1.2.1 Numbers ( ) + - * / % ^

A basic set of mathematical operators exists in the MERV code. The result of all cell calculations is always a 32 bit signed integer. So the result of a boolean equation, sum or system function call will always be a single 32 bit signed integer (the only other value could be a string, which is treated as 0 if used in a calculation). Numbers and references use the following operations on fixed values or variables.

Operation	Examples	Comment
Number	1, -100, 65535	Any number that can be stored in a 32 bit signed integer.
( )	(1 + 4), ((@a3 + 2) / @a1)	Forces the order of calculation. Operations inside brackets are executed first.
+	1 + 1, @a3 + 5	Adds the 2 values either side of the +
-	-50, -@a3	Unary minus, Negates the value
-	1 -1, @d13 - @f4	Subtracts the second value from the first
*	2 * 4, @b13 * 3	Multiplies the 2 values either side of the *
/	4 / 2, @b3 / 2	Divides the first value by the second value. **NOTE:** Divide by 0 returns 0
%, MOD	5 % 2, @b43 mod 60	Modulus. Uses the either a % or the text MOD. Returns the remainder of the first value after division by the second, which is always a number between 0 and the second value less 1. E.g. 4%2=0, 5%3=2, 5%6=5 **NOTE:** Divide by 0 returns 0
^	3 ^ 2, 4 ^ 3, @a3 ^ 2	Power. Returns the first value raised to the power of the second.

3.1.2.2 Boolean operations

All boolean operations can exist in a cell on their own. The result of any boolean operation is either 0 or 1 (though this is still stored in a 32 bit integer). Boolean operations are more normally used by conditional statements, but their value can also be used in mathematical functions. E.g. (@A1 < 5) * 100, will return 100 if the cell A1 is less than 5 and 0 if it is greater.

! == != < > <= >= || && ^^ OR AND XOR

Operation	Examples	Comment
!	!10, !@a3	NOT. If the value is 0 then this returns 1 otherwise it returns 0.
=, ==	1 == 2, @a3 = 4	= and == have exactly the same behaviour. Compares the values on either side of the =. If they are the same it returns 1 if not it return 0. e.g. 0=1 gives 0.
!=	1 != 2, @a3 != 10	Not Equal. Returns 1 if the values are not equal, 0 if they are. E.g. 0!=1 gives 1
<	1 < 2, @A5 < @Z82	Less than. Returns 1 if the left hand side is less in value than the right. E.g. 1<0 gives 0
>	1 > 2, @f42 > 0	Greater than. Returns 1 if the left hand side is greater in value than the right.
<=	1 <= 2, @A5 <= @Z82	Less Than or Equal To. Returns 1 if the left hand side is less than or equal to the value of the right.
>=	1 >= 2, @A5 >= @Z82	Greater Than or Equal To. Returns 1 if the left hand side is greater than or equal to the value of the right.
||, OR	@a3 || @a4, @a3 OR 255	Logical OR. Uses the C++ style or text. Returns 1 if the left or right hand side is non zero.
&&, AND	@a3 && @a4, @c4 and @d4	Logical AND. Uses the C++ style or text. Returns 1 if the left and right hand side are non zero.
^^, XOR	@a3 ^^ @a4, 1 XOR @a3	Logical EXCLUSIVE OR. Uses the C++ style or text. Returns 1 if only one of the values is 0. Returns 0 if both are 0 or both non 0.

3.1.2.3 Bitwise Operations

As the name suggests these values operate on the bits in the value. There are 32 bits in any number or returned value in MERV code. The examples below only use 8 bits for demonstration purposes

~ << >> +< +> | & ^

Operation	Examples	Comment
~	~@a1	Bitwise NOT. 01101001 becomes 10010110
<<	@a2 << 1	Bitwise Shift Left. Increases the value of the number by moving the bit pattern to the left by the amount specified e.g. 00000100 (4) << 2 => 00010000 (16)
>>	@z99 >> 8	Bitwise Shift Right. Decreases the value of the number by moving the bit pattern to the right by the amount specified e.g. 00001000 (8) >> 1 => 00000100 (4)
+<	@L10 +< 2	Bitwise Roll Left. Moves the bit pattern to the left by the amount specified adding bits to the bottom as they move off the top e.g. 01001000 +< 3 => 010000010
+>	@S5 +> 16	Bitwise Roll Right. Moves the bit pattern to the right by the amount specified adding bits to the top as they move off the bottom e.g. 00001110 +> 4 => 11100000
|	#F1 | #D3	Bitwise OR. Combines bit patterns. E.g. 00010001 | 0101000 = 01010001
&	#F1 & 15	Bitwise AND. Masks bit patterns. E.g. 01001011 & 00001111 = 00001011
^	#G3 ^ #D2	Bitwise EXCLUSIVE OR. Inverting mask Equivalent to (A | B) & (~A | ~B) E.g. 01001011 ^ 00001111 = 01000100

An example of a 32 bit roll left.

A 32 bit signed number rolled to the left by 10 bits is treated as an unsigned number for the purposes of the roll. Thus (289938840 <+ 10) will become:

289,938,84010 = 1148 1D9816 = 0001 0001 0100 1000 0001 1101 1001 10002

1	0010 0010 1001 0000 0011 1011 0011 0000
2	0100 0101 0010 0000 0111 0110 0110 0000
3	1000 1010 0100 0000 1110 1100 1100 0000
4	0001 0100 1000 0001 1101 1001 1000 0001
5	0010 1001 0000 0011 1011 0011 0000 0010
6	0101 0010 0000 0111 0110 0110 0000 0100
7	1010 0100 0000 1110 1100 1100 0000 1000
8	0100 1000 0001 1101 1001 1000 0001 0001
9	1001 0000 0011 1011 0011 0000 0010 0010
10	0010 0000 0111 0110 0110 0000 0100 0101

0010 0000 0111 0110 0110 0000 0100 01012 = 2076 604516 = 544,628,80510

When using bit manipulation it is usually the case that the numbers are not very important, it is the pattern of the bits that matters.

3.1.3 Variables and references

The cells in a table are specified by column letter (A-Z) and row number (1-99) giving a total of 2574 cells. Each cell in a sheet is calculated in turn (top left to bottom right, horizontally first) at runtime however references cause this process to be disrupted. When a cell is referenced it is checked to see if it has been calculated for this cycle and if it has then the value is already known and it can be returned to the calling cell. However if the cell is not calculated then the cell is forced to be calculated so that a value can be returned. It is not allowed to create circular references and this will cause an error in the runtime code resulting in a 0 being returned, the only compile time circular reference that is done is when self referencing a cell. Historical cell references are always valid even self referencing in a cell.

3.1.3.1 Cell References @

Operation	Examples	Comment
@Cn	@A1 + @a2	References another cell. Invalid cell references are caught at compile time.
@Cn[x]	@A1[0] + @A2[1], @A1[@B1], @A1[-3]	References the cell which is located x cells to the right of the cell Cn. x may be negative thus moving the reference to the left. If there is not a valid cell then 0 is returned.
@Cn[x, y]	@A1[@B1, @C1], @A1[-@C1, @B1]	References the cell which is located x cells to the right and y cells down from the cell Cn. Both x and y may be negative thus moving the reference to the left and up respectively. If there is not a valid cell then 0 is returned.

3.1.3.2 Historical Cell References $

Operation	Examples	Comment
$Cn	$A1 + $a2	References the previous value of another cell. Invalid cell references are caught at compile time.
$Cn[x]	$A1[0] + $A1[1], $A1[-$B1]	References the previous value of cell which is located x cells to the right of the cell Cn. x may be negative thus moving the reference to the left. If there is not a valid cell then 0 is returned.
$Cn[x, y]	$A1[$B1, $C1], $A1[-1, -$D1]	References the previous value of the cell which is located x cells to the right and y cells down from the cell Cn. Both x and y may be negative. If there is not a valid cell then 0 is returned.

3.1.3.3 Force and Reply Bit(s) Reference ?

Operation	Examples	Comment
?IN(SCN)	?IN("99989")	Gets the most recent reply bit pattern from the specified SCN (the SCN must be a string).
?IN(SCN, Index)	?IN("99989", 12)	Gets value of the bit from the most recent reply bit pattern of the specified SCN (the SCN must be a string). This simply creates a mask for the reply word by shifting 1 Index times.
?OUT(SCN)	?OUT("99988")	Gets the most recent force pattern sent to the specified street node (the SCN must be a string).
?OUT(SCN, Index)	?OUT("99988", 3)	Gets the bit in the most recent force pattern sent to the specified street node by creating a mask from shifting 1 by Index (the SCN must be a string).

3.1.4 AutoVars and VarOp

No cell may access another cell from another sheet. To pass variables between sheets AutoVars must be used. An AutoVar can only be set by the sheet it was declared in, however it can be read by any sheet anywhere. VarOp commands manipulate the values of AutoVars, but VarOp commands only take effect at the end of the current priority layer. AutoVars are declared using ## and are referenced using #.

##myVar

This defines an AutoVar variable with the name myVar. All AutoVars are not case sensitive and can only be placed in cells on their own. AutoVars can only be set by cells on the same sheet, but any sheet can read the value of any AutoVar. If this cell is referenced using a cell reference then the current value of the AutoVar is returned.

#myVar

References an AutoVar myVar defined on this sheet.

#mySheet.myVar

References an AutoVar myVar defined on the sheet named mySheet in this namespace.

#myNamespace.mySheet.myVar

References an AutoVar myVar defined on the sheet named mySheet in the namespace called myNamespace.

VarOp (Reset, #Variable, value)

Sets or resets the AutoVar to be the specified value.

VarOp (INC, #Variable)

Increments the AutoVar by 1.

VarOp (DEC, #Variable)

Decrements the AutoVar by 1.

VarOp (TimerUp, #Variable, value)

Starts a timer from the value specified that automatically increments the AutoVar by 1 every cycle. A TimerUp, TimerDown or Stop will override any previous commands.

VarOp (TimerDown, #Variable, value)

Starts a timer from the value specified that automatically decrements the AutoVar by 1 every cycle. A TimerUp, TimerDown or Stop will override any previous commands.

VarOp (STOP, #Variable)

Stops an AutoVar timer from counting. Has no effect on a static AutoVar.

3.1.5 Time Functions

Overview

These methods get the current time and date from the machine the PDM is running on. In the case of minutes and seconds this may not be exactly the same as the time held on the host UTC system. All the time functions use 24 hour clock notation.

CurrHour()

Gets the current hour.

CurrMin()

Gets the current minute in the hour.

CurrSec()

Gets the current second in the minute.

IsTimeBetween(Hour1, Min1, Sec1, Hour2, Min2, Sec2)

Parameters :

Hour1, Min1, Sec1	The start of the time period to check
Hour2, Min2, Sec2	The end of the time period to check

This function checks to see if the current clock time is between the times specified. If the end time is an earlier time than the start time then it is assumed to be for the next day. E.g. IsTimeBetween(22, 0, 0, 1, 0, 0) will return 1 if the current time is between 10pm at night and 1am the next morning.

The behaviour is unspecified if the values are out of range e.g. Hour1 = 100

CurrMon()

Gets the current month. Jan = 1, Dec = 12.

CurrDay()

Gets the current day in the month. E.g. 1-31

CurrDOW()

Gets the current day of the week. Monday = 1, Sunday = 7

IsWE()

Returns 1 if the current day is a weekend.

IsDayBetween(Month1, Day1, Month2, Day2)

Paramters :

Month1, Day1	The start of the date period to check
Month2, Day2	The end of the date period to check

Returns 1 if the current month and day are between the month and days specified. If the end is before the start then it assumed to be for the next year. E.g. IsDayBetween(6, 1, 8, 31)

The behaviour is unspecified if the values are out of range e.g. IsDayBetween(12, 0, 2, 31) i.e. 0 Dec to 31st Feb

IsSpecialDay()

Checks a user defined set of individual days to see if they are special. The day has a value assigned to it that can denote the type of day that is special. Special days must be entered into the DB table utc_special_days (see section 5.2.3). All special days must be a specific date they may not define a range. If dates are repeated then the first value found by the DB query will be returned.

3.1.6 Plan Functions

GoneActive(SCN, Bit, Time)

Parameters :

SCN	The identity of the SCN this bit is located on
Bit	The index of the flag to check
Time	Number of seconds to check through

Looks back in time through the recorded bit patterns from this node and checks for a 0 to 1 change of the specified bit. i.e. Leading edge

GoneInActive(SCN, Bit, Time)

Parameters :

SCN	The identity of the SCN this bit is located on
Bit	The index of the flag to check
Time	Number of seconds to check through

Looks back in time through the recorded bit patterns from this node and checks for a 1 to 0 change of the specified bit. i.e. Falling edge

CurrPlan(SCN)

Parameters :

SCN	The string name of the street node to get the plan from

Gets the current plan number for the specified SCN.

RunPlan(SCN, Plan , TIC)

Parameters :

Plan	The number of the plan to run
SCN	The string name of the street node to run the plan on
TIC	Time In Cycle that the new plan should start at.

This causes the specified plan to be added to a list of plans that will be executed at the end of the processing cycle. A new RunPlan command will override an older one, even in the same cycle. If no RunPlan is specified for an SCN then the PDM will stop controlling that node. The TIC parameter forces the plan to start at a paticulat Time In Cycle, and it will override the internal TIC for as long as it is present. A TIC value of -1 will cause the RunPlan to default to using the internal TIC, and a TIC value greater than or equal to the cycle time will return zero.

CurrStage(SCN)

Parameters :

SCN	The string name of the street node to get the stage from

Looks up the current plan, then finds the stage in that plan that is currently being sent to street.

PredictStage(SCN, Time)

Parameters :

SCN	The string name of the street node to get the plan from
Time	How far into the future to look in seconds

First it gets the current stage then, assuming that plan will not change, calculates what the stage will be at the number of seconds from now specified by the time.

TimeUntilStage(SCN, Stage)

Parameters :

SCN	The string name of the street node to get the plan from
Stage	The stage bit pattern to look for

Searches the current plan running of the specified SCN for the stage pattern, then returns the number of seconds before that stage pattern will be sent to street. Returns 0 if the stage specified is the current stage or if the stage cannot be found in the current plan.

3.1.7 Other Functions

?TIC(SCN)

Parameters :

SCN	The string name of the street node to get the plan from

Returns the time in cycle of the currently running plan on the specified SCN.

?GTIC(INT)

Parameters :

INT	The parameter will be modulised with a time_t value derived from a user specified epoch to give a global TIC which should sync with the one that the Alpha is using.

Returns Global time in cycle.

?CT(SCN)

Parameters :

SCN	The string name of the street node to get the plan from

Returns the cycle time of the currently running plan on the specified SCN.

AVL(Reference, Duration, Field)

Reference	A string made up of 3 parts separated by dots.
Duration	The time over which to check the DB for data.
Field	The textual name of the field that is to be retrieved.

The AVL command is used to give access to the information collected by the AVL subsystem. The AVL data can be different depending on the system that it is connected to, however it can be summarised into these 3 parameters. The first parameter (Reference) indicates the location of the virtual detector. This parameter is a string consisting of 3 parts separated by dots. E.g. "100.10.1". In the Leeds system, for example, it represents the 'traffic signal id', 'movement' on that signal and finally the 'trigger point' for that movement (if a different AVL system were connected these value could be an x, y, z location). The parameters in the reference can be substituted for wild cards e.g. 100.10.* would give all the trigger points for this movement. The Duration is required to allow for delays in receiving the signal in the system. It is generally set to 1 second to request details about AVL information that has just come into the system, but it has a range of up to 10 minutes (600 seconds). The Field is a string that names the column in the DB that should be requested. This field is completely dependant on the AVL system connected. Currently the fields that are being stored are priority, schedule_deviation, local_vcc, vehicle, date, time and sequence. However these names are generated from the particular MIB in use, and if the MIB were updated then these table names could be extended or changed. When making an AVL request only the first valid response matching the criteria is returned.

WriteDB(String)

The WriteDB command will write a user specified string to the user table in the STM database. This may be used for post run analysis of the behavior of the MERV code.

IF (Exp) THEN true ELSE false ENDIF

All IF statements must start with IF and stop with ENDIF. They can be nested. The expression is calculated and if the result is not 0 then the true section is executed. If the expression does evaluate to 0 then the false part is executed. The ELSE and false part of the IF statement are optional and if they are omitted the statement is simply skipped when the expression results in false.

ExitSheet()

Causes the sheet to stop processing and move to the next sheet instead of moving on to process the next cell. This is a very powerful function that allows logical control over which functions or sheets get run. It can be used on its own to mark the start of functions that get called conditionally or in IF statements to determine if whole sections of code should be skipped. (See also section).

?STATUS(SCN)

Basic command to provide status feedback to the ODT derived logic, so as to allow the logic to be influenced by the output of the SSB interface safety checking (e.g. make 'mode of 'node' available from SSB to the logic so as to enable logic and GUI display changes to be made). Returns 1 if the node is under control, 0 at any other time.

3.2 MERV Code tips

A sheet is executed from the top left (A1) to the bottom right (Z99) i.e. A1 -> A99, then B1 -> B99, till Z1->Z99, end of sheet. However sheets can make references to other parts of the same sheet or to AutoVars. To prevent confusing behaviour when a sheet is executing, all the inputs to it are locked and cannot be modified till the end of the priority layer (see section). This means that if there is a reference to a cell that has already been processed then the value can be returned immediately as it cannot possibly change for that cycle. When a reference is made to a cell that has not been processed then it is processed immediately until a value is found which can be returned for that cell. There are possibilities of cyclic calls, but these are caught and reported as errors by the core system. It is important to recognise this behaviour as it can have some surprising consequences when writing MERV code. E.g. In the simple case

Consider the pseudo code:

Increment variable #Counter

If #Counter equals 10 Then reset #Counter to 0

If we write this in MERV code we get:

```
VarOp(INC, #Counter)

IF (#Counter == 10) THEN VarOp(RESET, #Counter, 0) ENDIF
```

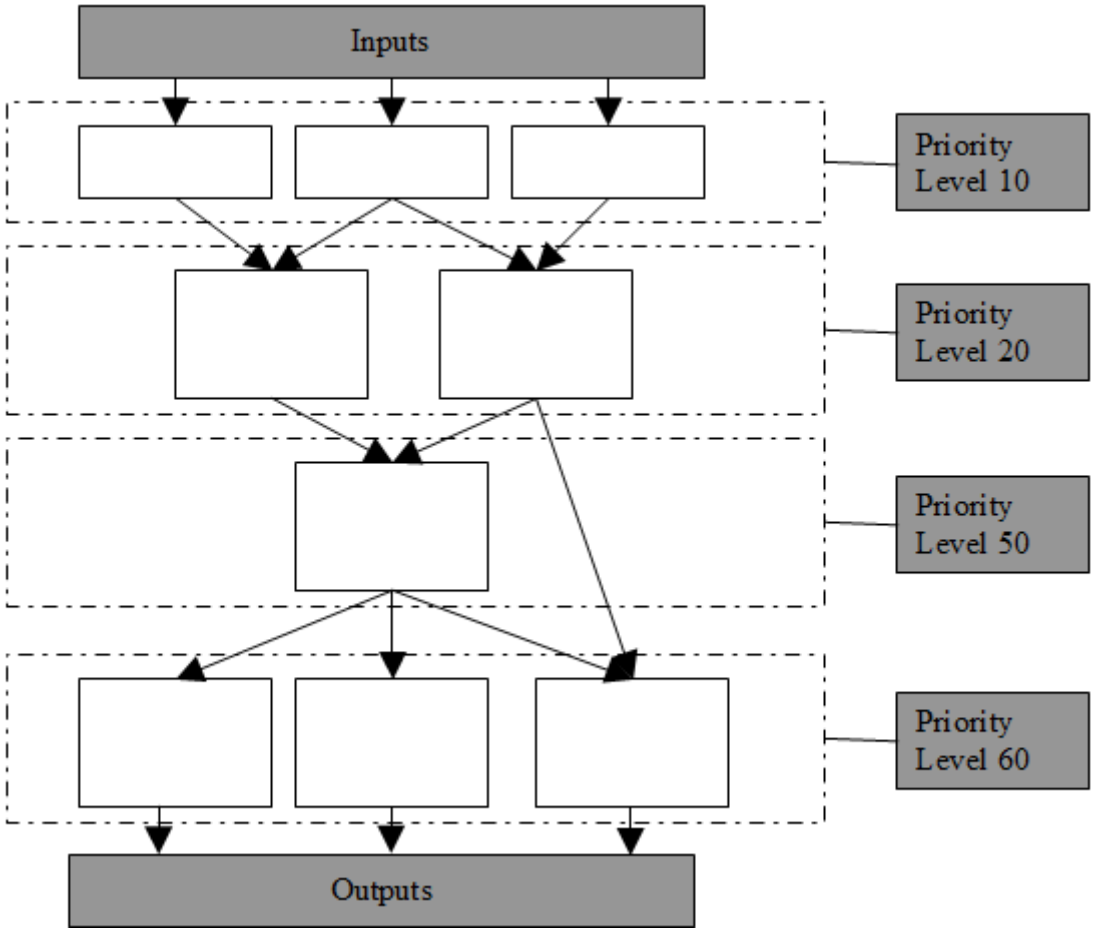
In a language such a C, Java or VB the value of #Counter will range between 0 and 9 inclusive, as when the variable is incremented to 10 it is immediately reset to 0. However in MERV code, because the value of Counter cannot be modified till the end of the priority layer, #Counter will equal 10 for one STM cycle and will not get reset to 0 till next cycle. Its range is therefore 0 to 10. Hence, in MERV code in order to give a range of 0 to 9 inclusive, the variable must be reset to 9.

A better way to write this in MERV code is:

```
IF (#Counter == 9) THEN VarOp(RESET, #Counter, 0) ELSE VarOp(INC, #Counter) ENDIF
```

Using Priority levels

A more important consequence of locking the state of variables occurs when trying to respond to inputs from street in the same second that they are detected. If the priority level for all sheets is the same then when the data from street is put into a variable it is not visible to the other sheets until the following cycle. On the other hand, if the process were to be spread over several sheets then it would take as many cycles as sheets to respond. The solution is to set sheets that have inputs to a higher priority e.g. 10. This means that they will get processed before the other sheets (with default priority 50) and more importantly their variables are updated when that priority level has been completed. This allows us to create a process flow when an input arrives; e.g.



Using ExitSheet()

Greater control of the way sheets can be executed is achieved with the ExitSheet() function. This method stops the sheet being processed at that cell, and is recommended as a means of reducing processor load. There are several useful features of this function that may not be obvious:

Initialisation

Consider the following 2 cells of MERV code, best placed in cells A1 and B1 on a sheet with priority 1.

```
##Initialised
IF (#Initialised == 0) THEN VarOp(RESET, #Initialised, 1) ELSE ExitSheet() ENDIF
```

Any code following this will only get executed once for the entire lifetime of this execution. In other words you can initialise variables to something other than 0.

Optimisation

Because there are no block IF statements and only one command is allowed in a cell the same IF statement must be repeated for each command in a block i.e.

In C we can write:

```
if ( i > 10 )
{
    i = 0;
    j = 1;
    k = 2;
}
```

In MERV code this is written as

```
IF (#i > 10) THEN VarOp(RESET, #i, 0) ENDIF
IF (#i > 10) THEN VarOp(RESET, #j, 1) ENDIF
IF (#i > 10) THEN VarOp(RESET, #k, 2) ENDIF
```

This is rather clumsy and can be optimised by using the ExitSheet function.

```
IF (#i <= 10) THEN ExitSheet() ENDIF
VarOp(RESET, #i, 0)
VarOp(RESET, #j, 1)
VarOp(RESET, #k, 2)
```

GoSub

If an ExitSheet() is placed at some point in a sheet it would seem that the rest of the sheet is redundant, however this is not the case. The default order the cells are processed in is not the only way the cells may get processed. If a cell is referenced that is ahead of the cell currently being processed then that cell will get processed immediately. It does not matter if the cell is after the ExitSheet cell, as the ExitSheet command only stops the default sequence, it has no effect on a chain of references. E.g.

	A	B	C	D
1	##Counter	VarOp(INC, #Counter)		
2	@A1 % 10	@A1 / 10		
3	IF (#Counter > 10) THEN @A8 ENDIF			
4	RunPlan(@A11, @B11)			
5	ExitSheet()			
6				
7	ModifyPlan	@A2	@B2	
8				
9				
10	GetPlanToRun	@A2	@B2	
11				
12				

The commands on cells A2 and B2, define input values for the functions ModifyPlan and GetPlanToRun. However GetPlanToRun is not called until the Runplan references the output cell A11. At this point, even though it is after the ExitSheet call the function is executed. When the function returns both the cells A11 and B11 have values, so the RunPlan call can now be executed. It is important to remember that after an ExitSheet only cells that are referenced will get executed, all other cells will simply be ignored. The IF statement on cell A3 causes the ModifyPlan function to be executed. It would generally be bad if this function were called every second, but because it will only get called when it is referenced it can be used to modify the plan returned by the GetPlanToRun function (i.e. when the #Counter is greater than 10).

3.2.1 Further Examples

Time since last value change

	A	B
1	##Var	
2	##LastChange	
3	IF (@A1 != $A1) THEN VarOp(TimerUp, #LastChange, 0) ENDIF	
4		
5		

Assume that the variable #Var will be changed by the GUI and we want to calculate how long it was since the last change (so we can decide if another change is allowed yet). #LastChange is the variable we want to use to hold the time. The IF statement checks the contents of the cell A1 (i.e. the current value of #Var), and compares it to the previous value of this cell. If they are not the same then something has changed so a timer is started counting up from 0. To decide if the GUI should be allowed to affect the system yet, we simply check to see if #LastChange is greater than a particular number of STM cycles e.g. 1 second.

Is this sheet running

	A	B
1	#Control.Reference.SheetNumber	
2	##Running	
3	VarOp(Reset, #Running, " ")	
4	IF (@A1 != 5) THEN ExitSheet() ENDIF	
5	VarOp(Reset, #Running, "Sheet 5 is Running")	

#SheetNumber identifies the sheet that is currently running. This, combined with the ExitSheet in the IF statement, means that only when the #SheetNumber variable is set to the correct value will this sheet get executed. When the sheet is running, the variable #Running is set to "Sheet 5 is running" because the ExitSheet function does not get called. This can be passed back to the GUI to give the user a meaningful message.

Override GUI control

	A	B
1	#Control.Reference.BlockGUIControl	
2	##GUIVar	
3	IF (@A1==0) THEN RunPlan("12345",1) ELSE RunPlan("12345", #GUIVar) ENDIF	
4		

In this example the GUI can select which plan to run. However there may be cases when this is not allowed so when the #BlockGUIControl variable is set only plan 1 can be run on the specified SCN.

GetWarningColour

This is a function that can be called to get a colour from red through yellow to green depending on the value of an input in a range.

	A	B	C	D
1	GetWarningColour			
2	@A16			
3		; Min	; Max	; Input
4	@C1 - @B1	; Range		
5	@B1 + (@A4 / 2)	; Mid Point		
6	@A5 - @B1	; Low Range		
7	@C1 - @A5	; High Range		
8	@D1 - @B1	; Low Diff		
9	@C1 - @D1	; High Diff		
10	(255 * @A8)/@A6	; Low diff/low range		
11	(255 * @A9)/@A7	; High diff/high range		
12	IF (@D1 <= @A5) THEN @A10 ELSE 255 ENDIF	; Green max when low		
13	IF (@D1 >= @A5) THEN @A11 ELSE 255 ENDIF	; Red max when high		
14	@A12 << 16	; Shift into red		
15	@A13 << 8	; Shift into green		
16	@A14 | @A15	; Create colour		

The inputs to this function are on cells B1, C1 and D1. The cells A4 to A9 work out the ranges and relative position of the input variable. A10 and A11 get a value from 0 to 255 (the maximum ranges of an individual colour, see section) for each half of the range. A12 and A13 force the colour at one end to its maximum value, this gives the colour pattern bellow:

Green is maximum on the left then reduces to the right and red is maximum on the right then reduces to the left. The last 3 lines merge the 2 individual colours into 1 colour to fit the hexadecimal format 0x00RRGGBB giving the blended colour range above. The red part from 0-255 (0 to 0xFF in hex) is shifted to the left 16 bits, and the green part is shifted 8 bits, giving 0x00RR000000 for the red component and 0x0000GG00 for the green (where RR and GG are hexadecimal values from 0 to 0xFF). These values are merged using the | (bitwise or) to give 0x00RRGG00 as there is no blue component used in this function.

The result is a reference on cell A2 to the calculated colour which is returned to the calling sheet. Now any sheet can work out the colour for a text box given a value and a range simply by calling this function and then assigning the returned value into an AutoVar.

3.3 Compiling and Committing

Once a related group of sheets have been created they can be compiled. The compile process goes through each sheet checking the code for syntactical and logical errors. Syntax errors are generally caused by incorrect numbers or types of parameters to inbuilt functions or mismatched brackets. Logical checking is not so comprehensive, but references to empty cells are some of the logical errors and will prevent the code from compiling. The plans, functions and layouts are also compiled and checking is done on each of these to make sure that they are valid. Not all errors are caught at compile time, and errors identified at runtime will result in that cell returning a value of 0. 0 should be used as the default value in the code so that if errors occur a 'safe' behaviour is selected. The GUI layouts are also compiled and any AutoVars referenced are validated. Errors from the GUI can often be caused because the name of the AutoVar has not been fully qualified (namespace.sheet.variable) or has been incorrectly spelt.

When making any changes, including turning on or off the debug level of a cell, the code must be recompiled. The compilation process generates the P-Code in memory that is required for the PDM. This is not the same as the save file that is created when saving the MERV code in the ODT. Once the code has been compiled it can be committed to a database. The IP address of the Postgres DB must be known and the Postgres DB server must be running so that the commit can take place. Each of the files will be listed as it is commited to the DB.

3.3.1 Errors

Sheet Errors

3.3.1.1 Not a valid AutoVar [name] in object [object name].

Reported when a GUI object contains something other than a variable reference.

3.3.1.2 Cannot find variable [name] on sheet [sheet] in object [object name]

The variable name looks like a valid variable reference, but when it is looked up it cannot be found. Check for spelling errors or try fully qualifying the variable name using namespace.sheet.variable

3.3.1.3 Multiple declaration of [variable]

There are 2 instances of ##VarName on the same sheet. This is often caused by copy and pasting cells.

3.3.1.4 Malformed IF statement found

An IF statement was found without either a matching THEN or ENDIF.

3.3.1.5 Mismatched brackets

The number of brackets in the cell do not add up. There must be equal numbers of open and close brackets of all types.

3.3.1.6 Retrieve plan MUST specify a node and a plan in column A

3.3.1.7 Apply plan MUST specify a node and a plan in column A

These errors are reported when using the insert functions from the menu. A cell in column A must be selected as the top left point of the 2 line function.

3.3.1.8 Apply plan MUST specify a number of plan points and a cycle time in column B

When using apply plan the B column must contain either a fixed value or reference.

3.3.1.9 No closing " found

When defining a string there must be " on either end of the string.

3.3.1.10 Unknown command or function name

This means that the ODT does not recognise the text as a valid command, but is more often caused because a variable does have the preceding #.

3.3.1.11 Brackets missing

Though similar in name to the "Mismatched brackets" error, this error means that a function that takes no parameters was not followed by (). The () are to make sure that the user intended to call a function and not a similarly named variable.

3.3.1.12 Function takes no parameters

Parameters were given to an inbuilt function that took no parameters. Check the function lists in this document for the correct parameters.

3.3.1.13 Parameters missing

3.3.1.14 No parameters found

3.3.1.15 All parameters missing

3.3.1.16 [function] requires [number] parameters

3.3.1.17 [number] parameters required

3.3.1.18 parameter only must be supplied to identify the SCN

The inbuilt function did not have the correct number of parameters specified. Check the function lists in this document for the correct parameters.

3.3.1.19 VarOp commands can only affect AutoVars on their own sheet. AutoVars can be read from any sheet

This error occurs when trying to use VarOp to set an AutoVar that is declared on another sheet. AutoVars can be read by all sheets, but they can only be written to by the sheet on which they are declared. This can mean that some code needs to be redesigned to accommodate this behaviour, but without this rule it would be impossible to know the value of a AutoVar as it would depend on the compilation order of the sheets and that is not controllable by the user.

3.3.1.20 Syntax for Runplan("SCN", PlanID [, TIC]), where TIC is optional

The parameters for the RunPlan function were not correctly specified. Check the definition of RunPlan in the function lists in this document.

3.3.1.21 IF found without expression

An IF statement was found without () immediately following it. IF a = b THEN 1 ENDIF is not a valid construct as a = b is not in brackets. i.e. IF (a = b) THEN 1 ENDIF

3.3.1.22 First parameter must be a string or reference to a string

Functions that require an SCN take a string as SCNs are alphanumeric. No further compile time checking is done if this a reference, a runtime error from the PDM will be produced if a number is found where a string was expected.

3.3.1.23 ## Autovar declaration MUST start at the beginning of the cell

3.3.1.24 ## Autovar declaration requires the entire cell

AutoVar declarations must be the only instruction in a cell. It is not permitted to conditionally declare an AutoVar i.e. IF (a==1) THEN ##SomeVar #ENDIF.

3.3.1.25 ## must define a variable

This generally means that the name used is already the name of a function in the system e.g. LastPlan. But it will also be generated in the first character following a ## declare in not a character.

3.3.1.26 Found [object] at end of cell

This is usually caused because [object] is the only character in a cell because only part of a cell was entered. E.g. 10 + 20 *

3.3.1.27 Found '?IN' without parameters

3.3.1.28 Found '?OUT' without parameters

3.3.1.29 Incorrect number of parameters for SCN input bit reference

3.3.1.30 Incorrect number of parameters for SCN output bit reference

The wrong number of parameters were given to the appropriate function. Check the function lists in this document for the correct parameters.

3.3.1.31 'VarOp' parameter error

3.3.1.32 Expected VarOp command

3.3.1.33 VarOp' 1st parameter MUST be 1 of : Reset, INC, DEC, TimerUp, TimerDown or Stop

3.3.1.34 Parameters missing, the second parameter must be the AutoVar to be modified

3.3.1.35 Second parameter must be the AutoVar to be modified

3.3.1.36 Missing variable reference

The parameters to a VarOp command were given incorrectly. Check the function lists in this document for the correct parameters.

3.3.1.37 Cell reference not found

The reference symbol @ or $ was followed by an incorrectly formed cell address. Cell addresses must be in the form: [letter A-Z] [number 1-99]

3.3.1.38 Invalid column letter

The first part of a cell address was not a letter from A to Z. This is often cause by accidental double letters e.g. AA

3.3.1.39 Row out of range

3.3.1.40 Cannot identify valid row number

A number given as the second part of a cell address was out of the range 1 to 99.

3.3.1.41 Self referencing is only allowed using historic reference e.g. $A1

Referencing the cell that you are in is only allowed when looking at the previous cycles results. Looking at this cycle's results would cause an infinite loop in the PDM and so is prevented at compile time. Cyclic references at runtime (ones that go through more than 1 cell) are caught at runtime.

3.3.1.42 Referencing an empty cell

Referencing an empty cell is considered as a logical error and not what the user intended. This often happens when code has been copied, but not all the references have been updated. This error only applies to non indexed cell references, i.e. it will not check @C1[5] or @S20[10, 30]. The default value of a cell is 0 and a warning will be produced at runtime if a reference to an empty cell is made. If the reference is desired then add a 0 into the empty cells that are being referenced.

3.3.1.43 Cannot reference autovars from a function. Pass the value to/from the calling sheet

As functions can get called multiple times from various sheets with different priorities they are not allowed to declare AutoVars. This in turn means that they cannot reference AutoVars as AutoVars can only be set on the sheet they are declared on.

3.3.1.44 # must reference a variable

The name of the variable was already defined as an inbuilt function name. Use a different name for this variable.

3.3.1.45 Cannot find variable [variable] on same sheet

The variable could not be found on this sheet. This means that either, the variable has not been declared, it is declared on a different sheet and/or namespace or it is miss spelt.

Plan Errors

3.3.1.46 Invalid time in cycle (TIC)

This generally means that the TIC for a specific plan point is not a number, or may be negative.

3.3.1.47 Time in cycle (TIC) is greater than the cycle time

A plan point in the currently compiling plan has a value greater than or equal to the cycle time.

3.3.1.48 No plan points found in plan [number] on node [node]

A plan with no plan points in not valid and would make no sense if selected.

Connection Errors

3.3.1.49 Failed to create a connection to [address]

The postgres DB was not found at the specified address. This can be caused because the IP address is incorrect, the postgres DB is not running or is invalid or there are network problems preventing communication between the ODT and the database.

3.3.1.49.1 Connection failed

A connection to the machine with the postgres database was made but there was an error logging into the database. Check the database is setup correctly.

ODT Errors

3.3.1.50 Names cannot be empty

When editing the namespace tree view you must always give a name for every object in the list.

3.3.1.51 Names may not contain any of these symbols: '" ;@$#+-*/%~!=<>|&^.()[],"

These symbols cannot be used when creating a name in the namespace tree view.

3.3.1.52 Imported project does not contain any Namespaces

An imported project must contain something otherwise the import will be ignored.

3.3.1.53 Cannot save file.

An error occurred when saving the file. This is probably due to the file being read only, but it may be because the drive is not accessible or there was not sufficient disk space.

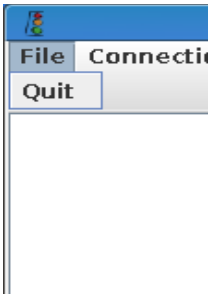
3.3.1.53.1 Unable to open file.

An error occurred when opening a file. This is unlikely to happen, but might imply that the file has been corrupted. Try loading the backup version of this file or moving the file to a different location.

4. GUI

4.1 Menus

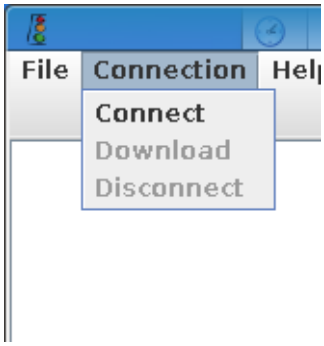
4.1.1 File



Quit

This exits the GUI.

4.1.2 Connection



Connect

This is the only available menu item when the GUI starts. The GUI can only display the current state of a variable in a live system, and therefore must be connected to a running IFU. The connection properties dialog allows the user to define the IP address of the running IFU to connect to. Select a connection from the list and press connect.

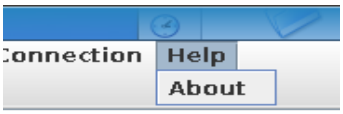
Download

Once connected, the GUI will show the download dialog. This will list all the GUIs currently available to the current user (if there are none listed then try a developer user login as this implies you have forgotten to set the user level options for the GUI layouts). Each layout must be loaded individually. Once loaded it will be displayed in the main window area with the contents of that layout.

Disconnect

Disconnect closes all the currently open layouts and closes the connection to the IFU.

4.1.3 Help



About

Displays information about the GUI including the version number.

4.2 Errors

4.2.1 Unable to connect to : [server]

There was a problem making a connection to the IP address that was specified. This is usually caused because the IFU is not running, it may be a simple case of a mistyped IP address, but often it can be related to network problems.

4.2.2 Unknown host [IP address]

4.2.3 Failed to create connection to [IP address]

When sending a message a problem occurred. This can be caused if the IFU stops running.

4.2.4 User [user name] failed to logon.

There was an error in the IFU that meant that the user name could not be identified. This happens very rarely but might indicate problems with the DB. If this message is received then the connection to the live IFU is working correctly.

4.2.5 No layouts found for current user

The query on the DB that happens internally when a user logs in did not return any matches. There are a number of possibilities for this, but the most common are the username you have typed is incorrect, or the access level of the user does not match any layouts. This is often the case with new layouts as all layouts default to developer only access, so that new sheets cannot be loaded by accident while they are still under development. Go to the options for this layout in the ODT and give the appropriate access level to the sheet or add more access privileges to the user in question.

5. Document Control

5.1 Maintenance and Distribution

This document is subject to formal change and control procedures as required by the Quality Management System (QMS).

5.2 Amendment History

Issue	Date	Change Descriptions	Author
1	09/04/14	Initial issue	A Smith
2	19/010/14	Minor layout issues	A Smith
3	01/08/15	Rebranding	A Smith
4	08/08/15	Minor layout issues	A Smith
5	02/11/15	Add WriteDB() command	A Smith

5.3 Abbreviations and Other Terms

ACIS	Suppliers of Real Time Passenger Information
Alpha	UTC Platform
AIMSUN	integrated transport modelling software
AutoVar	Automatic Variable
AVL	Automatic Vehicle Locator
Base-T	Baseband twisted pair cable
Beta	STM Prototype 2002
Bit	transmitted data
CLF	Cable less linking facility. Plans are stored in
Control Bits	requested transmitted data to traffic controller
CT	Cycle Time
FEP Line	Front End processor line
Force Bits	force bits to the controller to move to a specified stage
Duplex	Simultaneous data transmission
FT	Fixed time plan.
FTP	File Transfer Protocol
GBIT	tells the UTC which stage is active
GUI	Graphical User Interface
IFU	Interface Unit
IP	Internet Protocol
LAN	Local Area Network
ODT	Off-line Development Tool
MERV	Management Environment for Road Vehicles
MIB	Management Information Base
MOVER	Vehicle Actuated microprocessor
MVD	Microwave Vehicle Detectors
NTP	Network Time Protocol
PDM	Process Decision Model
PostgreSQL	Object-relational database
RTIG	Real time information group
RTPI	Real Time Passenger Information
SCOOT	Split Cycle Offset Optimisation Technique
SCN	unique identity of street control node
SNMP	Simple Network Management Protocol is a UDP-based network protocol
SPRUCE	Selected Priority in UTC Environment
SQL	Structured Query Language

SQLite	Embedded relational database management system
SSB	Sub System B (part of the core system)
STM	Strategic Traffic Management tool
TIC	Time In Cycle
TMS	Sheffield SCOOT Implementation
TSEU	Traffic Signals Europe
TSS	Transport Simulation Systems
UDP	User Datagram Protocol
UTC	Urban Traffic Control
VA	Vehicle Actuated

Referenced Documents

Title	Doc Ref	Issue

5.4 Related Documents

Title	Doc Ref	Issue
[1]		
[2]		