Brainstorming is a good activity. I do this every time, but inside a group, it's better :-)

"Do I understand it correctly that you want the Activity Editor (and in the future Route Editor, ...) to be free of MSTS legacy? "

By my side... Yes, it would be rather independant but we can provide an import tool for those who want to try their old MSTS activities. But, for me, there is no need for a backward compatibility.

If this is indeed the case then the consequence is that ORTS needs to be able to run both the MSTS activities and the ORTS activities (for a long time to come). Developing an ORTS Activity Editor than needs to go hand-in-hand to developing the ORTS code to run that activity.

I don't know what is the Rob's mind, but for me, ORTS must go forward, as long as possible with the MSTS activities. On a first step, we can translate the activity's informations (from MSTS or our Editor) in Rob's needs.

It also needs very much an architecture of how an ORTS activity needs to be defined. In other words, how does an ORTS activity need to be stored in files. The current MSTS activities have an .act file, one or more .pat files, one or more .con files and one or more .srv files. I assume various people already have ideas on that, but has anything already been defined or at least been written down?

For now, I had mainly work on the tool's environment without care about data exchange. Now, with the current Editor, I would like to go on the activity definition. The problem would be in the 'save' action, when we generate the json files. The file's structure must be defined, I have a 'first idea' but we can 'talk about'. I know that Rob will work with some kind of CSV files. It would be possible to make the jump.

Related to the question above, is there already a wish-list for things that the ORTS activities should contain (on top of what is already available in MSTS). I am aware of various people adding comments in some posts, but has that information been gathered already?

No, there is no 'wish-list' written, I have a good idea of what MSTS can do, so I start with this first 'list'. The code is relatively evolutive in term of data class, I will sent you an overview for this part.

Regarding code development

Personally I have good experience with what perhaps could be called 'agile' development. Instead of trying to create a big specification and taking a long time to implement everything, I really like starting small but functional. In this particular case, that would mean trying as fast as possible to generate a very basic activity with the editor, and having ORTS being able to run that activity. Such a basic activity probably does not need to be more than a combination of a header, description, an existing consist and existing path. Having the basic activity functioning, it is then easy to add things. But also easier to get feedback. And the good thing is, it is easy for the developers to see progress and keep on being motivated.

I use also the 'agile' methods in this project, because, for personnal reason, I work mainly during the day and not the evening. Yes, I have a job with many works to do, but I have also many 'temps mort' and I don't want to 'surf' on the internet for time killing.

Nevertheless, some initial idea on architecture is still needed. For TrackViewer I had MSTS trackviewer as an example/target.

In my current project, it's already done, based on the OR Project.

Regarding code sharing

So, how interwoven is your code at the moment?

The complete structure is under a specific repository called 'Activity Editor' at the same level as 'RunActivity'. For the sources from 'RunActivity', they are linked to my project and, in some case, I have introduced some '#if ACTIVITY!EDITOR' to insert some parts. But, with the last release, I have to split some files because the ORTS one has some specific methods that I can't use.

Is the libAE a complete separate directory? I would expect that the RunActivity should also be independent of the Activity Editor, not?

My first intention with LibAE is to provide a way to exchange information between the two projects through json files. So, LibAE contains the classes to save/load these files from/to objects. But, I think it would be difficult to use it due to the way C# works. I haven't yet tested this usage. But RunActivity can link the sources and compile them as any others.

Is it possible, at least for now, that you send your code so I can already look at it and play with what you have made?

Yes, sure I can. I will made an archive with all you need. But keep in mind, it's a job in progress with many things to do and, many stupid things

* You mention that you use xml for the preferences. Within my TrackViewer I used standard available settings (via properties), which will be stored in a standard place without having to write your own read and write code. ORTS itself already has quite an extensive class for settings as well. Is there a reason to use xml?

* You also mention using JSON for defining paths etc. Have there already been discussions within the ORTS team on what the file format is they want to use (replacing the MSTS format)? I do think ASCII is a good idea. XML might perhaps be too verbose, but possibly it is more flexible on the long run, compared to JSON.

* If I am correct I see in your JSON example that the data also shows the C# class it belongs to. I am not sure that is wise. The data that will be stored in a route (like activities, paths, ...) should probably be independent of the code that deals with it.

We had a small discussion about this in another conversation. What we need is a way to store objects and reload them easily. XML and JSON are good formats with many available and free libraries. That's why I use them. Yes, simple data files can be another possibility, we can store datas in row format, but we need classes to load and present them through objects. As soon as we modify the needed datas, we need to review the way these classes store and load the datas. After some months of use, I have modified the classes many times and, except in some case, the libraries can cope with these changes. As soon as we have decided the needs in term of datas, these classes could be never modified. Remember, for now all the datas saved on a json file are not RunActivity need, this will be defined in a near future, I hope.

Back to your document

Cool !

OK, I must confess, I work on many things at the same time BUT it's a way to fulfill some GUI behaviour until my mind and others have defined what we need for Activity Definition. In my mind and at first, We need a tool that can do rather same things as MSTS done. But some parts are already available from third parties, like consists. When I start the job, we have discussions with Rob and some others about it but the things must be completed. I decide to start the job by the Editor facilities : draw the tracks, gives some way to ease the navigation (my 'tag'), add some route Metadata like station and station area. Like this, I can solve many tools like : add, remove, move, edit, snap on track, etc. And, I put in the editor some ideas, like a notepad that I can complete in another time, like the Traffic parts.

At this time, I retain two parts : the Route Metadata and the Activity.

For the Activity, I don't want to separate each sub parts like 'Path', 'Schedules','Activity Description' during the definition parts, but it can be separates files for storing and reuse. The files and there formats are a next step, I have the minimum.

Path editor

This is the basically replacing the path editor from MSTS. You need to be able to define a path. In the end you want to have more advanced features to be able to build these paths, but initially, it doesn't need to be more complicated.

Yes, I agree.

You mention 'PathEvent', inheriting from a PathEventItem which is an ItemWidget. I do not understand this part. Especially the link to a widget. I understand widget to be a certain element in a GUI (Graphical User Interface). Did you mean something else? Or are you really meaning an item on the GUI, hence the map. In principle the path is idependent of the Editor (just like the current .pat file is). But on the GUI you need to be able to move/rotate/snap these elements to be able to edit the path.

The names are not my cup of tea. At one moment, they are right and with the evolution, they are bad... Well,
ItemWidget :   A generic class which represent every items that the editor can manage : edit, move, rotate... It contains all the needed 'virtual' methods to do this and for save/load operation. It keeps also the localisation of the item, within the Editor windows and in MSTS format. It's the parent class. Here is

the common data, some are saved in the json file, somes are not.

```
[JsonProperty("Location")]
public PointF Location;
[JsonProperty("Location2D")]
public PointF Location2D;
[JsonProperty("typeWidget")]
public int typeWidget;
[JsonProperty("CoordMSTS")]
public MSTSCoord Coord;
[JsonIgnore]
private bool movable;
[JsonIgnore]
private bool rotable;
[JsonIgnore]
private bool editable;
[JsonIgnore]
private bool lineSnap;
[JsonIgnore]
private bool actEdit;
[JsonIgnore]
public bool isSeen;
```

For exemple : In the 'generateView', there is a split for each kind of item, but for the actions fired by mouse, there is no need to know that it's a 'Start Activity' or a 'Tag'. They are moved in a same maneer, through the ItemWidget function IF the flag 'movable' is set.

Here are the current items available :

#region SignalWidget

No flags set at this moment but 'isEditable' could be. They are displayed with a small Icon.

#region SwitchWidget

No flags set, the switch number can be shown.

#region BufferWidget

No flags set, a small rectangle is used to show it.

#region TagWidget

flag 'isMovable'. It can be view as a flag or by it's name, specific to the TagWidget.

#region StationWidget

flag 'isMovable', 'isRotable'. There is a small panel with all the relevant datas, so no need to be Editable.

#region StationAreaWidget

flag 'isMovable' (in RouteNetaDAta mode), 'isEditable' and 'isLineSnap'. When align on a track, it's a connector.

#region SidingWidget

No flags set, it is used to show the two limits and the name (platform and siding).

#region ShapeWidget

No more used I think.

#region LineSegment

No flags set. Used to draw the tracks, specific methods to fulfill some needed informations.

#region PathEventItem

Another ParentClass to manage all the items relativ to the Activity.
Flag 'isMovable', 'isEditable', 'isLineSnap' and 'isActEdit'. In fact, the main behaviours are the same for all Activities Items.
The current child are : ActStartItems, ActStopItem and ActWaitItem. It's a start !

I can do a little refactoring to change the name by replacing Widget by Item...

Traffic editor.

I am not completely sure here. But paths alone are not enough to describe trains with their schedules. So traffic is like putting consists on certain paths, at certain times or perhaps with a certain schedule. Is this then similar to the service files from MSTS?

Yes, it's more or less the same.

Activity meta information

With this I mean like the top-level information and file of an activity. The title, description, time, weather, ... And of course which consists to use, which paths, possibly which services.

Yes

Route and route metadata.

This is where you lost me a bit. I do not understand your concept of route metadata. The reason for this is probably that it is something completely new for me. So let me try to understand this. You are adding extra information, metadata, to the routes. I see 'Stations' which describe a certain area, colored in yellow. I also see 'Tags', but I do not understand what exactly is tagging (or is it just the siding/station names from MSTS?).

Yes, it's a way to complete MSTS datas and they can be shared by users.

Station Area is a way to group some parts of a route in a 'Station'. In MSTS, this notion is not used and there is no way to define the direction of train in a track. The 'Station Area' has 'connectors' that define the interface between the outside tracks and the station (inside tracks). It's not mandatory.

Tag are a kinf of 'marker' that you can use to define 'shortcuts' to a specific location on the route. Maybe I can rename it as 'marker'

Is it the intention that this metadata is going to be distributed with routes, or is it going to be a route-addon (like an activity) in the sense that different route metadata can be created by different people? Where, apart from path editing, do you foresee that this route metadata is going to be used.

The tags are something more personnal but they can be shared. The 'Station Area' are something more concerned by the route Creator. He knows his route, where are the stations, the direction assigned to tracks, etc. So he can provide this Metadata to ease the definition of Path and Activities.

What I do not understand, for now, is why this is needed for an activity editor (or perhaps better, why it is needed for a first version of a activity editor). I can see the fact that both need the map with the tracks. And in that sense, they probably should share the GUI (being two features that one can do with the GUI.

A complex path is difficult to define, one needs to specify everything. When this path wants to use alternate paths, we have to specify this in the definition. When there is two tracks to join two 'station', you have to keep in mind what is the current rules of sharing these tracks. Etc. With the Station and the connectors, a simple path can be defined by simply select the start station , the end station, then the editor can show you the list of all reached stations that you can define as stop station, and that's all. The path will be generated automatically by using the right track and the right alternate paths inside a station. Now, it's a kind of wizard, you can do this like before, or editing the generated path to specify your own rules. Remember, my first work on this project is to complete the editor's environment, I use this part of the editor, not mandatory, for improvement and testing.

I do like the concept of 'Auto mode'. But it is perhaps really early to start supporting this when it is not even possible at the moment to run an ORTS-activity from ORTS.

'Auto node' is the normal way of working in runactivity. You follow the defined path and take care to the signalisation. 'Manual mode' is when you take the control over the Simulator Engine. This mode is like when you are in 'explore' route. My proposition to mix the two modes is a proposition that can be easily used with the station definition as soon as the RunActivity take care about the Route Metadata, it's another story.

After writing all of this, I found that my biggest concern at the moment is that the various parts of 'Activity Editor' are not well separated. Perhaps they are in your mind, but not in your document. For instance, I would really like to see a distinction between how the activity needs to be stored (which also determines how RunActivity and Menu need to deal with it), and how the activity editor needs to enable people to create and edit activities.

Let me few days / weeks to copy and arrange what I think and imagine. But we can also continue to 'brainstorm' about this. I have also to finalize some works on the editor before sending to you a copy of the sources.

So, please, if you could provide me with either documents or links to discussions previously held on discussions that have been going on already, that would be nice.

This is for Rob, the conversation is on Elvas Tower, like this one.