Subject: Re: [railML3] How to deal with UUIDs
Posted by Thomas Nygreen on Wed, 04 Sep 2024 12:27:48 GMT
View Forum Message <> Reply to Message

Dear all,

One of the powerful ideas behind including UUIDs in the railML 3 syntax was the ability to reference objects not present in the railML file. When referencing heavily from one domain (such as TT) into another (such as IS), this simplifies the export and resulting file, by not having to include all the referenced elements every time you export. It also facilitates sending only the parts of a previous export that have been changed, without including unchanged objects that are referenced. Obviously, these references cannot be formally validated by any XML parser. Instead, it is implied that both the exporting and the importing software either know how to look up each UUID or can work with it as a black box.

To take advantage of the possibilities offered by UUIDs, the software systems involved must obviously either support UUIDs already or be modified to support them. Systems that do not support UUIDs can still read railML files containing UUIDs, but will not be able to preserve them, and may experience problems reading files where UUIDs are used to reference external entities. To my knowledge, the adoption of UUIDs in the railway sector has been quite limited. I appreciate all feedback from the community on your experiences concerning UUIDs.

It is worth noting that UUIDs were included before <designator>s were applied as widely as now. They both offer stable external references, but they have a key difference: An object has (or is supposed to have) one UUID across all systems that know the object. On the other hand it may have multiple other keys in different registers, each specified by a separate <designator> in railML. This means that a foreign key from a <designator> cannot be used in the same way as a UUID to reference something outside of the current file.

Splitting tID, so that internal XML IDs and UUIDs go into different attributes is an easy change, that will help with some of the practical problems railML users encounter with the current implementation.

Changing tRef is harder. We essentially have three alternatives:

1: The current implementation.

2: Change all (>100) attributes of type tRef to elements offering separate attributes for references of types tGenericRef and tUUID. This keeps the option to provide either a file-internal XML ID or a global UUID, but it explicitly specifies which option you have used.

3: Remove tUUID from tRef. This completely removes the ability to reference objects not present in the file, and one would instead have to include at least a minimal representation containing the XML ID and the external key, either in the form of a @uuid attribute or a <designator>. For systems that use <designator>s and cannot use UUIDs, this is already the only alternative.

I look forward to more input from the community.

Best regards,
Thomas