Structs Data Format

Structs data files can contain mnemonic and/or event data. They have two basic forms, **archive** files, which must use the XBin format, and **buffer** files, which may either use the structs DSV format or the XBin format. In either format, they essentially provide a set of time-key-value mappings. By default keys are interpretted as mnemonics, unless they start with the dollar sign (`$`) character, in which case they are interpretted according to the rules of this document. Currently this is only used to embed event data, but additional features may be added in the future.

Note that buffer files are only intended to be imported with the STRUCT BUFFER IMPORT API action.

Mnemonics

Mnemonic data may either be denoted by the numeric mnemonic ID, or text mnemonic name. The value will be interpretted as a mnemonic ID if it is a numeric XBin type, or if it is a string containing only digit characters. If the ID is used, it must already be associated with an existing mnemonic definition, or an error will be thrown. Otherwise the text will be parsed as follows:

```
mnemonic   = name [ ';' subname ] [ ( '::' unit-enums ) | ( '(' unit-enums ')' ) ] [ '#' description ]
unit-enums = unit [ ';' enums ]
enums      = enum | ( enums [ '|' enum ] )
enum       = [ integer '=' ] label
```

The parsed mnemonic has five elements: the **name**, **subname**, **unit**, **enum map**, and **description**. Only the name is required. The combination of name, subname, and unit are used to lookup an existing mnemonic. The comparison is case insensitive and any whitespace is treated as a single underscore. If a matching mnemonic is not found, a new mnemonic is created automatically. Note that the overall name cannot contain the colon (`:`), semicolon (`;`), dollar sign (`$`), or number sign (`#`) characters.

The enum map specifies a one-to-one map of integer values to text labels. This is optional and unlike the other parameters, not interpretted as a key component of the mnemonic. They will be included in the generated mnemonic definition if it doesn't exist.

Events

Event operations may also be embedded in structs data files. These are indicated by keys which start with `$event`. The supported operations are inserting, opening, and closing events. Additionally, events may specify a `"name"` instead of `"e_id"`, which can be used to lookup a corresponding event definition ID, or create a new event definition if one does not exist with the provided name.

For the purposes of the following examples, assume a model "m" contains a pipe "p" with 3 event databases, "e", "ef" (single file per event), and "efs" (multi-file per event).

See also the Events and Event Definitions references.

Insert Event

```
$event.insert.<database name>
```

The database name must correspond to an event database in the associated pipe. The value must be a JSON object in the standard record format, with fields appropriate to the specified database. The event must omit the `"t_end"` value, and may omit the `"t_start"` value, to be populated by the time value from the corresponding row in the buffer file.

Open Event

```
$event.open.<database name>
```

Creates a single open event (interval with a start time and no end time). Uses the key `$event.open.<database name>`. The value must be a single JSON object, defining the content of the event record. The event must omit both the `"t_start"` and `"t_end"` values (`"t_start"` is populated by the row time, and `"t_end"` is `null` for an open interval).

Close Event

```
$event.close.<database name>
```

Closes an existing open event. The value must be a single JSON object, which may be used to update the values of any fields of the event. The object must omit both the `"t_start"` and `"t_end"` values (`"t_start"` cannot be edited, and `"t_end"` is populated by the row time).

Revision #24
Created 9 July 2024 18:02:57 by Nick Dobson
Updated 16 September 2024 21:25:00 by Bradley Tse