```
MODULE schematics
EXTENDS TLC

CONSTANTS
    InputSet,     set of ⟨key, value⟩ tuples used for input
    InputMap,     set of ⟨key, new_key⟩ tuples used to remap input keys
    AllowSet,     set of keys to allow on output (whitelist)
    OutputMap     set of ⟨key, new_key⟩ tuples used to remap output keys

 OPERATORS

 Map input keys to model fields and vice-versa
map_keys(keyval, mapping) ≜
    IF ∃ t ∈ mapping : t[1] = keyval[1] THEN
        ⟨(CHOOSE t ∈ mapping : t[1] = keyval[1])[2], keyval[2]⟩
     ELSE keyval

ASSUME
    map_keys(⟨"key", "val"⟩, {⟨"key", "new_key"⟩}) = ⟨"new_key", "val"⟩

 Field type functions (no-op)
StringType ≜ [
    convert   ↦ [value ∈ STRING ↦ value],
    validate  ↦ [value ∈ STRING ↦ TRUE],
    primitive ↦ [value ∈ STRING ↦ value]]
Field(x) ≜ StringType

 Convert values using field type
convert_values(keyval, function) ≜
    ⟨keyval[1], function[keyval[2]]⟩

ASSUME
    convert_values(⟨"key", "val"⟩, StringType.convert) = ⟨"key", "val"⟩

 Validate values using field type
validate_values(keyval, function) ≜
    function[keyval[2]]

ASSUME
    validate_values(⟨"key", "val"⟩, StringType.validate) = TRUE

 Filter fields for output
filter_keys(keyval, function, keyset) ≜
    function[keyval[1], keyset]

whitelist ≜ [key ∈ STRING, keyset ∈ SUBSET STRING ↦ key ∈ keyset]

ASSUME
    filter_keys(⟨"key", "val"⟩, whitelist, {"key2"}) = FALSE
```

```
--algorithm Schematics
variables
    $MapSet = \{\langle\rangle\}$, $ConvertSet = \{\langle\rangle\}$, $ValidSet = \{\langle\rangle\}$,
    $FilterSet = \{\langle\rangle\}$, $PrimitiveSet = \{\langle\rangle\}$, $OutputSet = \{\langle\rangle\}$
begin
    Map input keys to model fields:
    $MapSet := \{map\_keys(keyval, InputMap) : keyval \in InputSet\}$ ;
    Convert values using field type:
    $ConvertSet := \{convert\_values(\langle key, val\rangle, Field(key).convert) : \langle key, val\rangle \in MapSet\}$ ;
    Validate values using field type:
    $ValidSet := \{\langle key, val\rangle \in ConvertSet : validate\_values(\langle key, val\rangle, Field(key).validate)\}$ ;
    Filter fields for output:
    $FilterSet := \{keyval \in ValidSet : filter\_keys(keyval, whitelist, AllowSet)\}$ ;
    Convert values to primitive type:
    $PrimitiveSet := \{convert\_values(\langle key, val\rangle, Field(key).primitive) : \langle key, val\rangle \in FilterSet\}$ ;
    Map model fields to output fields:
    $OutputSet := \{map\_keys(keyval, OutputMap) : keyval \in FilterSet\}$ ;
    assert $\forall \langle key, val\rangle \in PrimitiveSet : key \in AllowSet$ ;
    print $OutputSet$ ;
end algorithm
```

BEGIN TRANSLATION

VARIABLES $MapSet, ConvertSet, ValidSet, FilterSet, PrimitiveSet, OutputSet,$
pc

$vars \triangleq \langle MapSet, ConvertSet, ValidSet, FilterSet, PrimitiveSet, OutputSet,$
$pc\rangle$

$Init \triangleq$ Global variables

$\land\ MapSet = \{\langle\rangle\}$
$\land\ ConvertSet = \{\langle\rangle\}$
$\land\ ValidSet = \{\langle\rangle\}$
$\land\ FilterSet = \{\langle\rangle\}$
$\land\ PrimitiveSet = \{\langle\rangle\}$
$\land\ OutputSet = \{\langle\rangle\}$
$\land\ pc =$ "Lbl_1"

$Lbl\_1 \triangleq$ $\land\ pc =$ "Lbl_1"

$\land\ MapSet' = \{map\_keys(keyval, InputMap) : keyval \in InputSet\}$
$\land\ ConvertSet' = \{convert\_values(\langle key, val\rangle, Field(key).convert) : \langle key, val\rangle \in MapSet'\}$
$\land\ ValidSet' = \{\langle key, val\rangle \in ConvertSet' : validate\_values(\langle key, val\rangle, Field(key).validate)\}$
$\land\ FilterSet' = \{keyval \in ValidSet' : filter\_keys(keyval, whitelist, AllowSet)\}$
$\land\ PrimitiveSet' = \{convert\_values(\langle key, val\rangle, Field(key).primitive) : \langle key, val\rangle \in FilterSet'\}$
$\land\ OutputSet' = \{map\_keys(keyval, OutputMap) : keyval \in FilterSet'\}$
$\land\ Assert(\forall \langle key, val\rangle \in PrimitiveSet' : key \in AllowSet,$
"Failure of assertion at line 69, column 5.")

$\land\ PrintT(OutputSet')$
$\land\ pc' = \text{“Done”}$

$Next \triangleq Lbl\_1$
$\quad\quad \lor$ Disjunct to prevent deadlock on termination
$\quad\quad\ (pc = \text{“Done”} \land \text{UNCHANGED } vars)$

$Spec \triangleq Init \land \Box[Next]_{vars}$

$Termination \triangleq \Diamond(pc = \text{“Done”})$

END TRANSLATION

* Modification History
* Last modified *Mon Jun* 30 17:12:56 *GMT*-03:00 2014 by *paul*
* Created *Wed Jun* 25 17:29:13 *GMT*-03:00 2014 by *paul*