MODULE *Md2LaTeXSystemDesign*

EXTENDS *Md2LaTeXCorrectness*, *FiniteSets*

CONSTANTS *ANY*, *PATH* Any object, any path

CONSTANTS

STRING_ALPH, The words of the latin alphabet

STRING_ALPH_NONEMPTY, $\triangleq$ *STRING_ALPH* \STRING_ALPH

STRING_LATEX All *LaTeX Markups*/commands, including "".

CONSTANT *RECORD* Any record

CONSTANT *NAT* Any integer 0, 1, 2, ...

The preferences define a unique record

CONSTANTS *DOMAIN_OF_PREFERENCES*, *SET_OF_PREFERENCES*

* *

"yes", "on", and "true" are synonyms;

"no", "off", and "false" are synomyms.

The way we express "yes", "no" in a *JSON* file.

* *

CONSTANTS *Y_N*, *JSON_YES*, *JSON_NO*, *EXCLUDED_BY_YES_OR_NO_POLICY*

* *

The preferences are identified with a file 'preferences'.

In practice, this is a *JSON* file ${}.*preferences.json*

(see CONSTANT *SET_OF_PREFERENCES*),

even if no semantics push on that.

isPreferencesFileCompliant${} keep track of preferences compliance.

* *

VARIABLES *preferences*,

isPreferencesFileCompliantConjectured,

isPreferencesFileCompliantProved,

isPreferencesFileCompliant

Convenient operator.

Recall that Yes = True and that No = False

$JSON\_BOOL \triangleq JSON\_YES \cup JSON\_NO$

YesOrNo policy: BEGINNING

* *

So, here is the specification of a file ${}.*preferences.json* .

See CONSTANTS *DOMAIN_OF_PREFERENCES*, *SET_OF_PREFERENCES*;

or *Md2LaTeXSystemDesignPreferencesFile*

Such a file must implement, or at least "follow", a specific policy,

that I named "*YesOrNo*".

The *YesOrNo* policy:

Goal: The very purpose of all that verbose is about implementing a
key -namely, Y_N - you can see as a switch on/off button.

Definitions:
1. No: Means "no action"; which we define as follows:
   i. If you do something, then it is discarded.
   ii.*If* you announce something, then it is disregarded.
2. Saying "No": The current key is mapped to some value in $JSON\_NO$.
3. Saying "Yes":The current key is mapped to some value in $JSON\_YES$.

Statement:
1. It is Yes *XOR* No (see above definitions 2, 3).
1.1.1. If you do not say anything, then it is No.
1.1.2. If you say "emptyset" (None, NULL, "", ...), then it is No.
1.1.3. If you say "No", then it is No.
1.2. If you say "Yes", then you do value of key right now.
4. You do not neither do nor say anything else.

Implementation
The "yes or no" key Y_N
(see Statement 1 for existence, Statement 4 for uniqueness)
is always the String "Y/N".
Moreover, we expect you actually do something relevant/nontrivial
This latter requirement cannot be implemented from a general case,
since:
(a) "relevant" and "nontrivial" are context-dependent.
(b) The context space is countable but infinite.
A complementary approach is about defining
$EXCLUDED\_BY\_YES\_OR\_NO\_POLICY$
as the minimal set of what is either trivial or irrelevant.
This set is not constructed ;) .
(In practice, $EXCLUDED\_BY\_YES\_OR\_NO\_POLICY$ should contain,
at least, boolean and numerical value
Hence, we cannot guarantee that the *YesOrNo* policy is implemented.
But we can check that the policy is "followed", in the sense that:
i. The policy is partially implemented and:
ii. If the provided content is actually relevant,
then the policy is (nonprovably) implemented.

Test / action'$isFollowingYesOrNoPolicy(f)$'
We expect the atom f to be a "first-degree subrecord" of preferences
($documentclass \mapsto \ldots$, $import\_packages \mapsto \ldots$, and so on).
$isFollowingYesOrNoPolicy(f)$ is TRUE *if*.f f follows *YesOrNo*.
* *
$isFollowingYesOrNoPolicy(f) \triangleq$

$\land\ Y\_N \in \text{DOMAIN } f$ the *YesOrNo* switch button
$\land\ Cardinality(\text{DOMAIN } f) = 2$ See Statement 4
$\land\ \lor\ f[Y\_N] \in JSON\_NO$ It is No
$\lor\ \land f[Y\_N] \in JSON\_YES$ It is Yes, and we "do well":
$\land \forall\, key \in (\text{DOMAIN } f) \setminus \{Y\_N\} :$
$\land\ f[key] \in EXCLUDED\_BY\_YES\_OR\_NO\_POLICY$

YesOrNo policy: END

* *
Either you want to implement *YesOrNo* (see above),
either you want to do something entirely different.
* *

$isCompatibleWithYesOrNoPolicy(f) \triangleq XOR($
$isFollowingYesOrNoPolicy(f),$
$Y\_N \notin \text{DOMAIN } f)$

* *
isPreferencesFollowingSpec = TRUE *if .f*
preferences follow the specs.
* *

$isPreferencesFollowingSpec \triangleq$

First, only a specific range for the keys:
$\land\ \text{DOMAIN } preferences \subseteq DOMAIN\_OF\_PREFERENCES$
Next, every "subrecords" must be compatible with *YesOrNo*.
$\land \forall\, key \in \text{DOMAIN } preferences :$
$isCompatibleWithYesOrNoPolicy(preferences[key])$

* *
Remark: If it is *YesOrNo*, then it is optional,
since you cannot turn off a mandatory feature.
In other words, we have the following criterion:
* *

$isOptional(record) \triangleq$
IF
$isFollowingYesOrNoPolicy(record)$
THEN TRUE ELSE FALSE

* *
Initial state
* *

$InitPreferences \triangleq$
$\land\ preferences \in SET\_OF\_PREFERENCES$

$InitSystemDesign \triangleq$
$\land\ InitCorrectness$

$\wedge$ *InitPreferences*

IF we do not believe that our current preferences file is legal,
then, there is no process at all, we just go nack to work:

$\wedge$ *isPreferencesFileCompliantConjectured* = TRUE

Of course, up to now, nothing has been proved:

$\wedge$ *isPreferencesFileCompliantProved* = FALSE
$\wedge$ *isPreferencesFileCompliant* = TRUE

* *
Next step
* *

$NextSystemDesign \triangleq$
$\wedge$ *NextCorrectness*
$\wedge$ *isPreferencesFollowingSpec*
$\wedge$ $isPreferencesFileCompliantProved' = isPreferencesFollowingSpec$
$\wedge$ $isPreferencesFileCompliantConjectured' =$ FALSE
$\wedge$ $isPreferencesFileCompliant' = XOR($
$isPreferencesFileCompliantConjectured'$,
$isPreferencesFileCompliantProved')$
$\wedge$ UNCHANGED *preferences*

* *
Invariants
* *

* *
Properties
* *
We can assume that our preferences comply with all policies:
Under the specs:
$\Box[isPreferencesFileCompliant]\_\langle$
$isPreferencesFileCompliantConjectured$,
$isPreferencesFileCompliantProved$
$\rangle$
Check with *TLC* must be *OK*.
I consider it as an invariant, even if it's not syntactically true,
since *isPreferencesFileCompliant*${} variables are primed.