6.6. `stringprep` — Internet String Preparation

Source code: Lib/stringprep.py

When identifying things (such as host names) in the internet, it is often necessary to compare such identifications for “equality”. Exactly how this comparison is executed may depend on the application domain, e.g. whether it should be case-insensitive or not. It may be also necessary to restrict the possible identifications, to allow only identifications consisting of “printable” characters.

RFC 3454 defines a procedure for “preparing” Unicode strings in internet protocols. Before passing strings onto the wire, they are processed with the preparation procedure, after which they have a certain normalized form. The RFC defines a set of tables, which can be combined into profiles. Each profile must define which tables it uses, and what other optional parts of the `stringprep` procedure are part of the profile. One example of a `stringprep` profile is `nameprep`, which is used for internationalized domain names.

The module `stringprep` only exposes the tables from **RFC 3454**. As these tables would be very large to represent them as dictionaries or lists, the module uses the Unicode character database internally. The module source code itself was generated using the `mkstringprep.py` utility.

As a result, these tables are exposed as functions, not as data structures. There are two kinds of tables in the RFC: sets and mappings. For a set, `stringprep` provides the “characteristic function”, i.e. a function that returns true if the parameter is part of the set. For mappings, it provides the mapping function: given the key, it returns the associated value. Below is a list of all functions available in the module.

`stringprep.`**`in_table_a1`**(*code*)

Determine whether *code* is in tableA.1 (Unassigned code points in Unicode 3.2).

`stringprep.`**`in_table_b1`**(*code*)

Determine whether *code* is in tableB.1 (Commonly mapped to nothing).

`stringprep.`**`map_table_b2`**(*code*)

Return the mapped value for *code* according to tableB.2 (Mapping for case-folding used with NFKC).

`stringprep.`**`map_table_b3`**(*code*)

Return the mapped value for *code* according to tableB.3 (Mapping for case-folding used with no normalization).

`stringprep.`**`in_table_c11`**(*code*)
: Determine whether *code* is in tableC.1.1 (ASCII space characters).

`stringprep.`**`in_table_c12`**(*code*)
: Determine whether *code* is in tableC.1.2 (Non-ASCII space characters).

`stringprep.`**`in_table_c11_c12`**(*code*)
: Determine whether *code* is in tableC.1 (Space characters, union of C.1.1 and C.1.2).

`stringprep.`**`in_table_c21`**(*code*)
: Determine whether *code* is in tableC.2.1 (ASCII control characters).

`stringprep.`**`in_table_c22`**(*code*)
: Determine whether *code* is in tableC.2.2 (Non-ASCII control characters).

`stringprep.`**`in_table_c21_c22`**(*code*)
: Determine whether *code* is in tableC.2 (Control characters, union of C.2.1 and C.2.2).

`stringprep.`**`in_table_c3`**(*code*)
: Determine whether *code* is in tableC.3 (Private use).

`stringprep.`**`in_table_c4`**(*code*)
: Determine whether *code* is in tableC.4 (Non-character code points).

`stringprep.`**`in_table_c5`**(*code*)
: Determine whether *code* is in tableC.5 (Surrogate codes).

`stringprep.`**`in_table_c6`**(*code*)
: Determine whether *code* is in tableC.6 (Inappropriate for plain text).

`stringprep.`**`in_table_c7`**(*code*)
: Determine whether *code* is in tableC.7 (Inappropriate for canonical representation).

`stringprep.`**`in_table_c8`**(*code*)
: Determine whether *code* is in tableC.8 (Change display properties or are deprecated).

`stringprep.`**`in_table_c9`**(*code*)
: Determine whether *code* is in tableC.9 (Tagging characters).

`stringprep.`**`in_table_d1`**(*code*)

Determine whether *code* is in tableD.1 (Characters with bidirectional property “R” or “AL”).

`stringprep.`**`in_table_d2`**(*code*)

Determine whether *code* is in tableD.2 (Characters with bidirectional property “L”).