XFN Composite Names Syntax

A

This appendix provides supplemental information about XFN composite name syntax.

Composite Name Encoding

All XFN implementations are required to support the ISO 646 portable representation (same encoding as ASCII) for XFN composite names. All other representations are optional.

All characters of the string form of an XFN composite name use a single encoding. There cannot be characters with different encodings in the same name string. This does not preclude component names of a composite name in its structural form from having different encodings. Code set mismatches that occur during the process of converting a composite name structure to its string form are resolved in an implementation-dependent way. Strings with code sets that are determined by the implementation to be compatible are converted without loss of information into a single representation, which is also determined by the implementation. When an implementation discovers that a composite name has components with incompatible code sets, it returns the error code `FN_E_INCOMPATIBLE_CODE_SETS`.

A

Backus-Naur Form (BNF)

The following defines the standard string form of XFN composite names in Backus-Naur Form (BNF). Note that all the characters of the string representation of one name must uniformly use the same encoding and locale information. The notations used are as follows:

Table 12-2 Backus-Naur Notation

<table>
<tr><th>Symbol</th><th>Meaning</th></tr>
<tr><td>::=</td><td>Is defined to be</td></tr>
<tr><td>|</td><td>Alternatively</td></tr>
<tr><td><<i>text</i>></td><td>Nonterminal element</td></tr>
<tr><td>‘ ’</td><td>Literal expression</td></tr>
<tr><td>*</td><td>The preceding syntactic unit can appear 0 or more times.</td></tr>
<tr><td>+</td><td>The preceding syntactic unit can appear 1 or more times.</td></tr>
<tr><td>{ }</td><td>The enclosed syntactic units are grouped as a single syntactic unit (can be nested).</td></tr>
</table>

The XFN composite name syntax in BNF is as follows.

Table 12-3 XFN Composite Name Syntax Using BNF

<table>
<tr><th>XFN Composite Name</th><th>BNF Syntax</th></tr>
<tr><td>NULL ::=</td><td>// Empty set</td></tr>
<tr><td><PCS> ::=</td><td>// Portable Character Set
The set consists of the glyphs:
//!"#$%&'()*+,\0123456789:;<=>?
//@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_’
//abcdefghijklmnopqrstuvwxyz{|}~</td></tr>
<tr><td><CharSet> ::=</td><td><PCS>
| Characters from the repertoire of a string representation</td></tr>
<tr><td><EscapeChar> ::=</td><td>\</td></tr>
<tr><td><ComponentSep>::=</td><td>/</td></tr>
<tr><td><Quote1>::=</td><td>"</td></tr>
<tr><td><Quote2> ::=</td><td>‘</td></tr>
</table>

Table 12-3 XFN Composite Name Syntax Using BNF *(Continued)*

<table>
<tr><th>XFN Composite Name</th><th>BNF Syntax</th></tr>
<tr><td><MetaChar> ::=</td><td><EscapeChar> | <ComponentSep></td></tr>
<tr><td><SimpleChar> ::=</td><td>// any character from <CharSet> with <MetaChar>, <Quote1>,
// and <Quote2> excluded. An<EscapeChar> <MetaChar>, or
// <EscapeChar> <Quote1>, or <EscapeChar> <Quote2> is equivalent to
// <SimpleChar>.</td></tr>
<tr><td><Component> ::=</td><td><SimpleChar>*
| <SimpleChar>+ {<Quote1> | <Quote2> | <SimpleChar>}*
| <Quote1><CharSet>* {<EscapeChar><Quote1>}* <Quote1>
// <CharSet> must not contain unescaped <Quote1>
// (note that <Quote2> can appear unescaped)
| <Quote2><CharSet>* {<EscapeChar><Quote2>}* <Quote2>
// <CharSet> must not contain unescaped <Quote2>
// (note that <Quote1> can appear unescaped)</td></tr>
<tr><td><CompositeName> ::=</td><td>NULL
| <Component> {<ComponentSep> <Component>}*</td></tr>
</table>

Decomposing the Composite Name String

The function `fn_composite_name_from_string()` returns an XFN composite name in its structural form, `FN_composite_name_t`, given the composite name's string representation. The syntax rules used by `fn_composite_name_from_string`() are as follows.

An XFN composite name is decomposed into an ordered set of components (<Component>). Each component represents a compound name, or a single atomic name of a compound name if the compound name's syntax uses the XFN component separator (/) as a separator for its atomic parts and the compound name is not quoted.

The following are the rules for parsing a composite name.

1. Any <ComponentSep> character that is neither escaped nor enclosed in quoted strings is considered to be a component separator.
2. Any string enclosed by component separators is a component (<Component>).

3. A composite name is parsed and decomposed into components from left to right:
    a. The first component is the string preceding the first occurrence of a component separator.
    b. Empty components are processed as follows:
        i. A leading component separator (the composite name begins with a component separator) means a leading null component.
        ii. A trailing component separator (the composite name ends with a component separator) means a trailing null component.
    c. Two consecutive component separators mean a null component.
    d. The name string that immediately follows the last component separator of the composite name is the final component.
4. A component string is evaluated from left to right and converted into its standard form according to the following rules:
    a. A component string is considered to be quoted if it is enclosed in a pair of matching unescaped quote characters (either a <Quote1> or a <Quote2> pair). The quoted string must represent the full component; that is, a begin quote must immediately be preceded by a component separator or no character, and the end quote must immediately be followed by a component separator or no character.
    b. If a component does not contain a valid begin quote (a <Quote1> or <Quote2>immediately preceded by either a component separator or no character), any occurrence of <Quote1> or <Quote2> within that component is treated just as any other <SimpleChar>.
    c. An unmatched begin quote (missing or misplaced end quote) fails with an FN_E_ILLEGAL_NAME status.
    d. Quotes are considered to be escaped in quoted strings if a matching quote character is preceded immediately by the unescaped <EscapeChar>.
    e. Quoted components are resolved by eliminating the quote characters from the component name and substituting possibly escaped quotes by simple quote characters. <MetaChar>s and the nonmatching quote characters enclosed in quoted strings are treated just as any other <SimpleChar>.

f. Any of the defined metacharacters (<ComponentSep> and <EscapeChar>) is considered to be escaped in an unquoted component name string if preceded immediately by the unescaped <EscapeChar> (for instance, the sequence <EscapeChar> <EscapeChar> <ComponentSep> denotes an escaped <EscapeChar> but an unescaped <ComponentSep>).

g. <Quote1> and <Quote2> are considered to be escaped in an unquoted component if and only if <EscapeChar> is preceded by a component separator (that is, sequences <ComponentSep> <EscapeChar> <Quote1> or <ComponentSep> <EscapeChar> <Quote2>). Other occurrences of <Quote1> and <Quote2> in an unquoted component are treated just as any other <SimpleChar>.

h. Any occurrence of escaped <MetaChar>, escaped <Quote1>, or escaped <Quote2> in unquoted components is substituted by the corresponding unescaped character.

i. No substitution is done for <EscapeChar> <SimpleChar>. <EscapeChar> <SimpleChar> simply maps to <EscapeChar> <SimpleChar>.

Composing the Composite Name String

The function `fn_string_from_composite_name()` returns the string representation of an XFN composite name given its structural form (`FN_composite_name_t`). The following are the rules used by `fn_string_from_composite_name()`.

1. The components are added to the composite name string in left to right order (that is, rightmost is the tail).
2. Successive components are separated by the component separator (<ComponentSep>).
3. Empty components are handled in the following way:
   a. A leading empty component is represented by a leading <ComponentSep>.
   b. A trailing empty component is represented by a trailing <ComponentSep>.
   c. An empty component occurring within a composite name is represented by two consecutive <ComponentSep>s.

4. A composite name denoting a single non-empty component does not contain any unescaped component separator.

5. Any occurrence of <ComponentSep> in a component is escaped by inserting <EscapeChar> immediately preceding <ComponentSep>.

6. If the first character of a component is either <Quote1> or <Quote2>, it will be escaped by inserting <EscapeChar> immediately preceding the quote.

7. Any occurrence of <EscapeChar> before <ComponentSep> in a component is escaped by inserting <EscapeChar> immediately preceding the <EscapeChar>.

8. Any occurrence of <EscapeChar> as the first character of a component with <Quote1> or <Quote2> as the second character in a component is escaped by inserting <EscapeChar> immediately preceding the <EscapeChar>. Subsequent <EscapeChar> occurring before any matching quote character is also escaped by inserting <EscapeChar> immediately preceding the <EscapeChar>.