XFN Composite Names 11

This chapter describes XFN composite names in detail.

Syntax

The standard string form for XFN composite names is the concatenation of the components of a composite name from left to right, with the XFN component separator character (/) separating each component. Components can be quoted using either double-quote (`""`) or single-quote (`''`) pairs. The XFN component separator or quote characters may be escaped using a backslash character (\) if the intention is for these characters not to behave as separators or quotes. Note that quotation marks and escape characters are interpreted as such only when they appear in places that need quotes or escapes. For example, a quote appearing in an unquoted component is not interpreted as a quote.

XFN defines an abstract data type, FN_composite_name_t, for representing the structural form of a composite name. XFN also defines the syntax of how component string names are composed into an XFN composite name and the corresponding rules for converting an XFN composite name to its structural form from its string form, and vice versa. The XFN client interface includes operations that perform these conversions.

Table 11-1 contains some examples of how the string form of XFN composite names are decomposed into components according to the syntax of XFN composite names. See also Appendix A, "XFN Composite Names Syntax," for more information.

Table 11-1 String and Structural Forms of XFN Composite Names

Stringform	**Components in** FN_composite_name_t
a	a
a/b/c	a, b, c
a/	a, ""
/a	"", a
a//	a, "", ""
a//b	a, "", b
""	""
/	"", ""
//	"", "", ""
"a/b/c"/d	a/b/c, d
"a.b.c"/d	a.b.c, d
a.b.c/d	a.b.c, d
a"b/c	a"b, c
a'b/c	a'b, c
"a/b/c	illegal name
\"a/b/c	"a, b, c
a\b\c/d	a\b\c, d
a\b\/c	a\b/c

Table 11-1 String and Structural Forms of XFN Composite Names *(Continued)*

Stringform	Components in `FN_composite_name_t`
`"a\"b"/c`	`a"b, c`
`'"a/b/c"'`	`"a/b/c"`
`'a\/b'/c`	`a\/b, c`
`a\\b/c`	`a\b, c`
`a/\"b`	`a, "b`

Composite Name and Naming System Boundaries

There may not be a one-to-one correspondence between component separators and naming system boundaries if a composite name contains names from naming systems that use the same character as the XFN component separator to separate their atomic names. Consequently, a component of a composite name may represent an atomic name from a hierarchical naming system that uses the XFN component separator or a compound name. S*trong separation* and *weak separation* refer to whether a context always treats the XFN component separator as a naming system boundary.

Strong Separation

An XFN context that treats the XFN component separator as a naming system boundary supports strong separation. An XFN component separator that appears *within* a component to be resolved by the context must be escaped or quoted.

Support for strong separation is a property of a context. A context that supports strong separation expects to receive the name that it is going to resolve entirely in one component of the composite name structure. When a composite name is supplied to such a context, it consumes the leading component of the name; any remaining components are left to be resolved by subordinate naming systems.

An XFN context with a name syntax that is either flat or hierarchical, and does not use the XFN component separator as its atomic separator, supports strong separation. Examples of naming systems that support strong separation are

DNS and NIS+, both of which have right-to-left dot-separated names. The following are examples of names with DNS and NIS+ components, respectively.

```
.../wiz.com/orgunit/ppt
orgunit/accountspayable.finance/user/jsmith
```

Weak Separation

An XFN context that does not always treat the XFN component separator as a naming system boundary supports weak separation. This arises when the component naming system associated with the context uses the same character as the XFN component separator as its atomic component separator, and the context allows its atomic separator to appear unescaped and unquoted in its compound names when they occur in composite names. This means that an XFN component separator may not necessarily signify a naming system boundary.

Support for weak separation is a property of a context. A context that supports weak separation expects to receive its atomic names in separate components of the composite name structure. When a composite name is supplied to a context that supports weak separation, the context consumes the leading components of the name (and treats them as atomic components); any remaining components are resolved by subordinate naming systems. The number of components consumed is determined either syntactically or dynamically.

CDS names and X.500 names are examples of names that use the XFN component separator as their atomic name separator. X.500 supports weak separation using a syntactic method (by scanning for typed names) while CDS supports weak separation by determining the naming system boundary dynamically.

The following example shows a composite name with an X.500 component.

```
.../c=us/o=wiz.com/orgunit/ppt
```

Note – An XFN context that supports weak separation using only syntax-specific discovery of its naming system boundary may not always be able to be federated with arbitrary subordinate naming systems. If the subordinate naming system has a naming syntax that is indistinguishable from that of the superior naming system, the superior naming system would not be able to identify the naming system boundary.

Naming systems that use the same character as the XFN component separator as their atomic separator, and which cannot support weak separation because it cannot use a syntactic or dynamic method to determine the naming system boundary, must provide context implementations that support strong separation. This means that occurrences of atomic separators must be quoted or escaped when they appear in compound names within composite names.

Composite Name Resolution

Composite name resolution combines resolution in each component naming system and resolution across federated naming system boundaries. There are several techniques for resolving an XFN composite name in the underlying federation of naming systems.

This section describes two implementation techniques for composite name resolution across a naming system boundary. One technique uses an *explicit* next naming system pointer (NNSP) to resolve across a naming system boundary. The other uses an *implicit* NNSP to resolve across a naming system boundary.

An NNSP is the XFN reference of an XFN context in which composite name components from subordinate naming systems are to be resolved. NNSPs are entities that “tie” naming systems together into a federated system. NNSPs can be bound to names, in which case they are *explicit* NNSPs or *junctions*. NNSPs can also be nameless, in which case they are *implicit* NNSPs.

Explicit NNSPs: Junctions

A junction is an atomic name that is bound to an NNSP. It is a terminal name in the superior naming system. There is no limit on the number of junctions bound in a single context, except that imposed by the context. A context may reserve certain names for use as junctions or have other policies for selecting names for use as junction. The conventions used for identifying junctions and their references are context-specific.

Composite name resolution involving junctions proceeds as follows, depending on whether the context supports strong or weak separation.

A context that supports strong separation and junctions consumes the first component of the composite name supplied to it. The last atomic name of the first component must be a junction. Any remaining components are resolved in the context named by the junction.

A context that supports weak separation and junctions resolves a composite name by consuming leading components until a junction is reached, at which point resolution of any remaining components is continued in the context resolved to by the junction. Determination of whether a component is a junction can be done statically using a syntactic policy or dynamically during resolution.

Implicit NNSPs

When a context does not want to use part of its namespace for junctions, it uses implicit NNSPs for federating subordinate naming systems. An implicit NNSP is named using the XFN component separator. For example, the name `wiz.com/` names the implicit NNSP of `wiz.com`. Each context can have one implicit NNSP.

Composite name resolution involving implicit NNSPs proceeds as follows, depending on whether the context supports strong or weak separation.

A context that supports strong separation and resolves composite names using an implicit NNSP consumes the first component of the composite name supplied to it. Any remaining components are resolved in the context pointed to by the implicit NNSP of the first component.

A context that supports weak separation and implicit NNSPs in its implementation needs to distinguish the use of the XFN component separator character as an XFN component separator or an atomic separator. This means that such a context needs to know when to exit the current (native) naming system and follow the NNSP. This can be achieved using a static, syntactic policy or a dynamic, resolution-based policy.

With the syntactic policy, a context syntactically discovers where the boundary between its naming system and the subordinate naming system lies. This may impose certain restrictions on the syntax of subordinate naming systems. Subordinate naming systems must not permit as valid top-level names that are syntactically indistinguishable from names allowed in the superior naming system. For example, assume the superior naming system has a name syntax whose distinguishing feature is that each atomic part must have an equal sign (=). The superior naming system might impose as a policy that subordinate naming systems must not have top-level names that have an equal sign in them. Resolution in the superior naming system continues until all leading components of the supplied composite name fitting the syntactic rule are consumed. Any remaining components are resolved in the context of the NNSP of the last component fitting the syntactic rule.

If a context is not able to syntactically differentiate between atomic components and composite name components, or does not want to impose any syntactic restrictions, it may be able to determine the naming system boundary at runtime during resolution. The policy is to continue resolution in the current naming system until resolution fails, at which point the implicit NNSP associated with the last context at which resolution succeeded is used to continue the resolution. A conflict arises if the same atomic name is bound both in the last context and the context pointed to by the last context's implicit NNSP. In this case, the binding in the last context takes precedence. Note that this way of supporting weak separation requires the context to have the capability of returning remaining unresolved parts of a given name.

Coexistence of Explicit and Implicit NNSPs

Naming systems that implement either technique may coexist in a federation. A naming system that supports composite name resolution using junctions can be federated with one that supports implicit NNSPs, and vice versa.

XFN Links

An XFN link affects name resolution in the following way. Suppose *lname* is a link bound to the atomic name *aname* in the context *ctx*. If at some point resolution of a composite name *cname* reaches the context *ctx* and the next atomic name is *aname*, resolution of *aname* results in the resolution of the link name *lname*. This is termed "following the link." If the first component of the link *lname* is the atomic name "`.`," the remaining components of *lname* are resolved relative to *ctx*; otherwise, *lname* is resolved from the initial context. The resolution of any remaining portion of the name *cname* proceeds from the reference that results by resolving *lname*.

The link name may itself cause resolution to resolve through other links. This gives rise to the possibility of a cycle of links whose resolution could not terminate normally. As a simple means to avoid such nonterminating resolutions, implementations may define limits on the number of XFN links that may be resolved in any single operation invoked by the caller.