Portable Systems Group

Windows NT Prefix Table Specification

**Author:** Gary D. Kimura

Revision 1.2, August 2, 1989

Do not remove any of the division marks (:::) in this template. They control the basic layout of the document, including the way page numbers are printed.

The text below (".Begin Table C.") is hidden text that is necessary for the table of contents to work correctly. Don't delete the hidden text, or you'll end up with the TOC at the end of your document.

.Begin Table C.

1. Introduction 1

2. Initializing a Prefix Table 1

3. Adding a New Prefix 2

4. Removing a Prefix 2

5. Locating a Prefix 2

6. Enumerating a Prefix Table 3

.End Table C.

Create section headers by using the H1-H4 styles. You make a line a header by typing Alt-H-n, where n is 1-4, when the cursor is anywhere within the line. You don't need to make the header text bold; the style does that for you. You can switch a paragraph back to normal by typing either Alt-X-P or Alt-P-S.

The following text shows how to format various textual elements using the SPEC style sheet. You should remove these examples.

# 1. Introduction

This specification describes the **Windows NT** routines that implement a prefix table package. The **Windows NT** prefix table package is designed for storing and matching path name prefixes.

The prefix table package exports two opaque types, one is a prefix table used to denote a collection of prefixes, and the other is a prefix table entry used to denote a prefix. A user of this package first initializes a prefix table variable and then either inserts or deletes prefixes, or finds the longest matching prefix in the table.

To utilize this package, the caller needs to define a local structure to contain a prefix table entry. When inserting a new prefix, the caller then supplies a prefix table, prefix string, and a prefix table entry.

To look up a prefix the caller supplies a prefix table and a full path name. If a prefix match is found, the look up procedure returns a pointer to the prefix table entry corresponding to the located prefix. The programmer can then use the CONTAINING\_RECORD macro to associate the prefix table entry with the local data structures.

Only prefixes that match whole logical parts of a path name are returned. For example, if a table contains the prefix "\Alpha\Beta" then a look up on "\Alpha\", "\Alpha\Bet" and "\Alpha\BetaGamma" will be unsuccessful, but a look up on "\Alpha\Beta" and "\Alpha\Beta\Gamma" will be successful.

The **APIs** that implement the prefix table package are the following:

**PfxInitialize** - Initialize a prefix table.

**PfxInsertPrefix** - Add a new prefix to a prefix table.

**PfxRemovePrefix** - Remove an existing prefix from a prefix table.

**PfxFindPrefix** - Search a prefix table for the longest matching prefix.

**PfxNextPrefix** - Enumerate all of the prefixes stored in a prefix table.

# 2. Initializing a Prefix Table

A prefix table is initialized with the **PfxInitialize** procedure.

**VOID**

**PfxInitialize** (

**IN PPREFIX\_TABLE** *PrefixTable*

);

Parameters:

*PrefixTable* - A pointer to the prefix table variable being initialized

A prefix variable cannot be used by the other procedures until it has been initialized.

# 3. Adding a New Prefix

A user adds a new prefix to a prefix table with the **PfxInsertPrefix** procedure. The prefix is only added if it is not already in the table.

**BOOLEAN**

**PfxInsertPrefix** (

**IN** **PPREFIX\_TABLE** *PrefixTable*,

**IN** **PSTRING** *Prefix*,

**IN** **PPREFIX\_TABLE\_ENTRY** *PrefixTableEntry*

);

Parameters:

*PrefixTable* - A pointer to the prefix table being modified

*Prefix* - The string to add to the prefix table

*PrefixTableEntry* - A pointer to the prefix table entry to use to denote the prefix

This procedure has a return value of TRUE if the prefix was not already in the table, and FALSE otherwise.

The prefix table keeps a reference to the input *Prefix* string, so once a prefix is added the input string must not be changed by the user.

# 4. Removing a Prefix

A user removes an existing prefix from a prefix table with the **PfxRemovePrefix** procedure.

**VOID**

**PfxRemovePrefix** (

**IN** **PPREFIX\_TABLE** *PrefixTable*,

**IN** **PPREFIX\_TABLE\_ENTRY** *PrefixTableEntry*

);

Parameters:

*PrefixTable* - A pointer to the prefix table being modified

*PrefixTableEntry* - A pointer to the prefix table entry to remove from the prefix table

# 5. Locating a Prefix

A user searches a prefix table for the longest matching prefix with the **PfxFindPrefix** procedure.

**PPREFIX\_TABLE\_ENTRY**

**PfxFindPrefix** (

**IN** **PPREFIX\_TABLE** *PrefixTable*,

**IN** **PSTRING** *FullString*,

**IN** **BOOLEAN** *CaseInsensitive*

);

Parameters:

*PrefixTable* - A pointer to the prefix table being queried

*FullString* - A pointer to the string to use as the key in searching the prefix table

*CaseInsensitive* - Indicates if the prefix look up should be preformed in a manner that ignores the case (*CaseInsensitive* is TRUE) or expects an exact, case sensitive, match (*CaseInsensitive* is FALSE)

This procedure returns a pointer to the prefix table entry corresponding to the longest prefix that matches the input string if one exists and NULL otherwise.

# 6. Enumerating a Prefix Table

A user can enumerate all of the elements of a prefix table with the **PfxNextPrefix** procedure.

**PPREFIX\_TABLE\_ENTRY**

**PfxNextPrefix** (

**IN** **PPREFIX\_TABLE** *PrefixTable*,

**IN** **BOOLEAN** *Restart*

);

Parameters:

*PrefixTable* - A pointer to the prefix table being enumerated

*Restart* - Indicates if the enumeration should start over

This procedure returns a pointer to the next prefix stored in the prefix table, or NULL if there are no more entries. The following code fragment illustrates how a programmer uses this procedure to enumerate all of the elements of a prefix table.

for (PfxTableEntry = PfxNextPrefix( PrefixTable, TRUE );

PfxTableEntry != NULL;

PfxTableEntry = PfxNextPrefix( PrefixTable, FALSE )) {

LocalRecord = CONTAINING\_RECORD( PfxTableEntry,

LOCAL\_RECORD,

PrefixTableEntry);

...

}

**Revision** **History:**

Original Draft 1.0, August 1, 1989

Revision 1.1, August 2, 1989

1. Added statement concerning how the prefix table keeps a pointer back to the input prefix string.

2. Changed **PfxAddPrefix** to **PfxInsertPrefix**.

3. Dropped *PrefixLength* from **PfxFindPrefix**, and added *CaseInsensitive* parameter.

4. Changed **PfxNextPrefix** prototype and added example of its use.

Revision 1.2, August 3, 1989

1. Changed **PfxFindPrefix** to return a pointer to the table entry if one is found instead of an OUT parameter.