NAME

Win::Hivex − Perl bindings for reading and writing Windows Registry hive files

SYNOPSIS

```
use Win::Hivex;

$h = Win::Hivex->open ('SOFTWARE');
$root_node = $h->root ();
print $h->node_name ($root_node);
```

DESCRIPTION

The `Win::Hivex` module provides a Perl XS binding to the **hivex**(3) API for reading and writing Windows Registry binary hive files.

ERRORS

All errors turn into calls to `croak` (see **Carp**(3)).

METHODS

open

```
$h = Win::Hivex->open ($filename,
                       [verbose => 1,]
                       [debug => 1,]
                       [write => 1,]
                       [unsafe => 1,])
```

Open a Windows Registry binary hive file.

The `verbose` and `debug` flags enable different levels of debugging messages.

The `write` flag is required if you will be modifying the hive file (see ''WRITING TO HIVE FILES'' in **hivex**(3)).

This function returns a hive handle. The hive handle is closed automatically when its reference count drops to 0.

root

```
$node = $h->root ()
```

Return root node of the hive. All valid hives must contain a root node.

This returns a node handle.

last_modified

```
$int64 = $h->last_modified ()
```

Return the modification time from the header of the hive.

The returned value is a Windows filetime. To convert this to a Unix `time_t` see: <http://stackoverflow.com/questions/6161776/convert-windows-filetime-to-second-in-unix-linux/6161842#6161

node_name

```
$string = $h->node_name ($node)
```

Return the name of the node.

Note that the name of the root node is a dummy, such as `$$$PROTO.HIV` (other names are possible: it seems to depend on the tool or program that created the hive in the first place). You can only know the ''real'' name of the root node by knowing which registry file this hive originally comes from, which is knowledge that is outside the scope of this library.

The name is recoded to UTF-8 and may contain embedded NUL characters.

node_name_len

```
$size = $h->node_name_len ($node)
```

Return the length of the node name as produced by `node_name`.

This returns a size.

node_timestamp

```
$int64 = $h->node_timestamp ($node)
```

Return the modification time of the node.

The returned value is a Windows filetime. To convert this to a Unix `time_t` see: <http://stackoverflow.com/questions/6161776/convert-windows-filetime-to-second-in-unix-linux/6161842#6161

node_children

```
@nodes = $h->node_children ($node)
```

Return an array of nodes which are the subkeys (children) of `node`.

This returns a list of node handles.

node_get_child

```
$node = $h->node_get_child ($node, $name)
```

Return the child of node with the name `name`, if it exists.

The name is matched case insensitively.

This returns a node handle, or `undef` if the node was not found.

node_nr_children

```
$size = $h->node_nr_children ($node)
```

Return the number of nodes as produced by `node_children`.

This returns a size.

node_parent

```
$node = $h->node_parent ($node)
```

Return the parent of `node`.

The parent pointer of the root node in registry files that we have examined seems to be invalid, and so this function will return an error if called on the root node.

This returns a node handle.

node_values

```
@values = $h->node_values ($node)
```

Return the array of (key, value) pairs attached to this node.

This returns a list of value handles.

node_get_value

```
$value = $h->node_get_value ($node, $key)
```

Return the value attached to this node which has the name `key`, if it exists.

The key name is matched case insensitively.

Note that to get the default key, you should pass the empty string `""` here. The default key is often written `"@"`, but inside hives that has no meaning and won't give you the default key.

This returns a value handle.

node_nr_values

```
$size = $h->node_nr_values ($node)
```

Return the number of (key, value) pairs attached to this node as produced by `node_values`.

This returns a size.

value_key_len

```
$size = $h->value_key_len ($val)
```

Return the length of the key (name) of a (key, value) pair as produced by `value_key`. The length can legitimately be 0, so errno is the necessary mechanism to check for errors.

In the context of Windows Registries, a zero-length name means that this value is the default key for this node in the tree. This is usually written as `"@"`.

The key is recoded to UTF-8 and may contain embedded NUL characters.

This returns a size.

value_key

```
$string = $h->value_key ($val)
```

Return the key (name) of a (key, value) pair. The name is reencoded as UTF-8 and returned as a string.

The string should be freed by the caller when it is no longer needed.

Note that this function can return a zero-length string. In the context of Windows Registries, this means that this value is the default key for this node in the tree. This is usually written as `"@"`.

value_type

```
($type, $len) = $h->value_type ($val)
```

Return the data length and data type of the value in this (key, value) pair. See also`value_value` which returns all this information, and the value itself. Also, `value_*` functions below which can be used to return the value in a more useful form when you know the type in advance.

node_struct_length

```
$size = $h->node_struct_length ($node)
```

Return the length of the node data structure.

This returns a size.

value_struct_length

```
$size = $h->value_struct_length ($val)
```

Return the length of the value data structure.

This returns a size.

value_data_cell_offset

```
($len, $value) = $h->value_data_cell_offset ($val)
```

Return the offset and length of the value's data cell.

The data cell is a registry structure that contains the length (a 4 byte, little endian integer) followed by the data.

If the length of the value is less than or equal to 4 bytes then the offset and length returned by this function is zero as the data is inlined in the value.

Returns 0 and sets errno on error.

value_value

```
($type, $data) = $h->value_value ($val)
```

Return the value of this (key, value) pair. The value should be interpreted according to its type (see `hive_type`).

value_string

```
$string = $h->value_string ($val)
```

If this value is a string, return the string reencoded as UTF-8 (as a C string). This only works for

values which have type `hive_t_string`, `hive_t_expand_string` or `hive_t_link`.

value_multiple_strings

```
@strings = $h->value_multiple_strings ($val)
```

If this value is a multiple-string, return the strings reencoded as UTF-8 (in C, as a NULL-terminated array of C strings, in other language bindings, as a list of strings). This only works for values which have type `hive_t_multiple_strings`.

value_dword

```
$int32 = $h->value_dword ($val)
```

If this value is a DWORD (Windows int32), return it. This only works for values which have type `hive_t_dword` or `hive_t_dword_be`.

value_qword

```
$int64 = $h->value_qword ($val)
```

If this value is a QWORD (Windows int64), return it. This only works for values which have type `hive_t_qword`.

commit

```
$h->commit ([$filename|undef])
```

Commit (write) any changes which have been made.

`filename` is the new file to write. If `filename` is null/undefined then we overwrite the original file (ie. the file name that was passed to `open`).

Note this does not close the hive handle. You can perform further operations on the hive after committing, including making more modifications. If you no longer wish to use the hive, then you should close the handle after committing.

node_add_child

```
$node = $h->node_add_child ($parent, $name)
```

Add a new child node named `name` to the existing node `parent`. The new child initially has no subnodes and contains no keys or values. The sk-record (security descriptor) is inherited from the parent.

The parent must not have an existing child called `name`, so if you want to overwrite an existing child, call `node_delete_child` first.

This returns a node handle.

node_delete_child

```
$h->node_delete_child ($node)
```

Delete the node `node`. All values at the node and all subnodes are deleted (recursively). The`node` handle and the handles of all subnodes become invalid. You cannot delete the root node.

node_set_values

```
$h->node_set_values ($node, \@values)
```

This call can be used to set all the (key, value) pairs stored in `node`.

`node` is the node to modify.

`@values` is an array of (keys, value) pairs. Each element should be a hashref containing `key`, `t` (type) and `data`.

Any existing values stored at the node are discarded, and their `value` handles become invalid. Thus you can remove all values stored at `node` by passing `@values = []`.

node_set_value

```
$h->node_set_value ($node, $val)
```

This call can be used to replace a single `(key, value)` pair stored in `node`. If the key does not already exist, then a new key is added. Key matching is case insensitive.

`node` is the node to modify.

COPYRIGHT

LICENSE

SEE ALSO

hivex (3), **hivexsh** (1), <http://libguestfs.org>, **Sys::Guestfs** (3).