NAME

krb5_get_default_principal, **krb5_principal**, **krb5_build_principal**, **krb5_build_principal_ext**, **krb5_build_principal_va**, **krb5_build_principal_va_ext**, **krb5_copy_principal**, **krb5_free_principal**, **krb5_make_principal**, **krb5_parse_name**, **krb5_parse_name_flags**, **krb5_parse_nametype**, **krb5_princ_set_realm**, **krb5_principal_compare**, **krb5_principal_compare_any_realm**, **krb5_principal_get_comp_string**, **krb5_principal_get_realm**, **krb5_principal_get_type**, **krb5_principal_match**, **krb5_principal_set_type**, **krb5_realm_compare**, **krb5_sname_to_principal**, **krb5_sock_to_principal**, **krb5_unparse_name**, **krb5_unparse_name_flags**, **krb5_unparse_name_fixed**, **krb5_unparse_name_fixed_flags**, **krb5_unparse_name_fixed_short**, **krb5_unparse_name_short** - Kerberos 5 principal handling functions

LIBRARY

Kerberos 5 Library (libkrb5, -lkrb5)

SYNOPSIS

#include <krb5.h>

krb5_principal;

void
krb5_free_principal(*krb5_context context*, *krb5_principal principal*);

krb5_error_code
krb5_parse_name(*krb5_context context*, *const char *name*, *krb5_principal *principal*);

krb5_error_code
krb5_parse_name_flags(*krb5_context context*, *const char *name*, *int flags*, *krb5_principal *principal*);

krb5_error_code
krb5_unparse_name(*krb5_context context*, *krb5_const_principal principal*, *char **name*);

krb5_error_code
krb5_unparse_name_flags(*krb5_context context*, *krb5_const_principal principal*, *int flags*, *char **name*);

krb5_error_code
krb5_unparse_name_fixed(*krb5_context context*, *krb5_const_principal principal*, *char *name*, *size_t len*);

krb5_error_code
krb5_unparse_name_fixed_flags(*krb5_context context*, *krb5_const_principal principal*, *int flags*,

*char *name, size_t len*);

krb5_error_code
krb5_unparse_name_short(*krb5_context context, krb5_const_principal principal, char **name*);

krb5_error_code
krb5_unparse_name_fixed_short(*krb5_context context, krb5_const_principal principal, char *name, size_t len*);

void
krb5_princ_set_realm(*krb5_context context, krb5_principal principal, krb5_realm *realm*);

krb5_error_code
krb5_build_principal(*krb5_context context, krb5_principal *principal, int rlen, krb5_const_realm realm, ...*);

krb5_error_code
krb5_build_principal_va(*krb5_context context, krb5_principal *principal, int rlen, krb5_const_realm realm, va_list ap*);

krb5_error_code
krb5_build_principal_ext(*krb5_context context, krb5_principal *principal, int rlen, krb5_const_realm realm, ...*);

krb5_error_code
krb5_build_principal_va_ext(*krb5_context context, krb5_principal *principal, int rlen, krb5_const_realm realm, va_list ap*);

krb5_error_code
krb5_make_principal(*krb5_context context, krb5_principal *principal, krb5_const_realm realm, ...*);

krb5_error_code
krb5_copy_principal(*krb5_context context, krb5_const_principal inprinc, krb5_principal *outprinc*);

krb5_boolean
krb5_principal_compare(*krb5_context context, krb5_const_principal princ1, krb5_const_principal princ2*);

krb5_boolean
krb5_principal_compare_any_realm(*krb5_context context, krb5_const_principal princ1,*

krb5_const_principal princ2);

*const char **
krb5_principal_get_comp_string(*krb5_context context, krb5_const_principal principal, unsigned int component*);

*const char **
krb5_principal_get_realm(*krb5_context context, krb5_const_principal principal*);

int
krb5_principal_get_type(*krb5_context context, krb5_const_principal principal*);

krb5_boolean
krb5_principal_match(*krb5_context context, krb5_const_principal principal, krb5_const_principal pattern*);

void
krb5_principal_set_type(*krb5_context context, krb5_principal principal, int type*);

krb5_boolean
krb5_realm_compare(*krb5_context context, krb5_const_principal princ1, krb5_const_principal princ2*);

krb5_error_code
krb5_sname_to_principal(*krb5_context context, const char *hostname, const char *sname, int32_t type, krb5_principal *ret_princ*);

krb5_error_code
krb5_sock_to_principal(*krb5_context context, int socket, const char *sname, int32_t type, krb5_principal *principal*);

krb5_error_code
krb5_get_default_principal(*krb5_context context, krb5_principal *princ*);

krb5_error_code
krb5_parse_nametype(*krb5_context context, const char *str, int32_t *type*);

DESCRIPTION

krb5_principal holds the name of a user or service in Kerberos.

A principal has two parts, a PrincipalName and a realm. The PrincipalName consists of one or more

components. In printed form, the components are separated by /. The PrincipalName also has a name-type.

Examples of a principal are nisse/root@EXAMPLE.COM and host/datan.kth.se@KTH.SE.
krb5_parse_name() and **krb5_parse_name_flags**() passes a principal name in *name* to the kerberos principal structure. **krb5_parse_name_flags**() takes an extra *flags* argument the following flags can be passed in

KRB5_PRINCIPAL_PARSE_NO_REALM
: requires the input string to be without a realm, and no realm is stored in the *principal* return argument.

KRB5_PRINCIPAL_PARSE_REQUIRE_REALM
: requires the input string to with a realm.

krb5_unparse_name() and **krb5_unparse_name_flags**() prints the principal *princ* to the string *name*. *name* should be freed with free(3). To the *flags* argument the following flags can be passed in

KRB5_PRINCIPAL_UNPARSE_SHORT
: no realm if the realm is one of the local realms.

KRB5_PRINCIPAL_UNPARSE_NO_REALM
: never include any realm in the principal name.

KRB5_PRINCIPAL_UNPARSE_DISPLAY
: don't quote

On failure *name* is set to NULL. **krb5_unparse_name_fixed**() and **krb5_unparse_name_fixed_flags**() behaves just like **krb5_unparse**(), but instead unparses the principal into a fixed size buffer.

krb5_unparse_name_short() just returns the principal without the realm if the principal is in the default realm. If the principal isn't, the full name is returned. **krb5_unparse_name_fixed_short**() works just like **krb5_unparse_name_short**() but on a fixed size buffer.

krb5_build_principal() builds a principal from the realm *realm* that has the length *rlen*. The following arguments form the components of the principal. The list of components is terminated with NULL.

krb5_build_principal_va() works like **krb5_build_principal**() using vargs.

krb5_build_principal_ext() and **krb5_build_principal_va_ext**() take a list of length-value pairs, the list is terminated with a zero length.

krb5_make_principal() works the same way as **krb5_build_principal**(), except it figures out the length of the realm itself.

krb5_copy_principal() makes a copy of a principal. The copy needs to be freed with **krb5_free_principal**().

krb5_principal_compare() compares the two principals, including realm of the principals and returns TRUE if they are the same and FALSE if not.

krb5_principal_compare_any_realm() works the same way as **krb5_principal_compare**() but doesn't compare the realm component of the principal.

krb5_realm_compare() compares the realms of the two principals and returns TRUE is they are the same, and FALSE if not.

krb5_principal_match() matches a *principal* against a *pattern*. The pattern is a globbing expression, where each component (separated by /) is matched against the corresponding component of the principal.

The **krb5_principal_get_realm**() and **krb5_principal_get_comp_string**() functions return parts of the *principal*, either the realm or a specific component. Both functions return string pointers to data inside the principal, so they are valid only as long as the principal exists.

The *component* argument to **krb5_principal_get_comp_string**() is the index of the component to return, from zero to the total number of components minus one. If the index is out of range NULL is returned.

krb5_principal_get_realm() and **krb5_principal_get_comp_string**() are replacements for **krb5_princ_component**() and related macros, described as internal in the MIT API specification. Unlike the macros, these functions return strings, not krb5_data. A reason to return krb5_data was that it was believed that principal components could contain binary data, but this belief was unfounded, and it has been decided that principal components are infact UTF8, so it's safe to use zero terminated strings.

It's generally not necessary to look at the components of a principal.

krb5_principal_get_type() and **krb5_principal_set_type**() get and sets the name type for a principal. Name type handling is tricky and not often needed, don't use this unless you know what you do.

krb5_sname_to_principal() and **krb5_sock_to_principal**() are for easy creation of "service" principals that can, for instance, be used to lookup a key in a keytab. For both functions the *sname* parameter will be used for the first component of the created principal. If *sname* is NULL, "host" will be used instead.

krb5_sname_to_principal() will use the passed *hostname* for the second component. If *type* is KRB5_NT_SRV_HST this name will be looked up with **gethostbyname**(). If *hostname* is NULL, the local hostname will be used.

krb5_sock_to_principal() will use the "sockname" of the passed *socket*, which should be a bound AF_INET or AF_INET6 socket. There must be a mapping between the address and "sockname". The function may try to resolve the name in DNS.

krb5_get_default_principal() tries to find out what's a reasonable default principal by looking at the environment it is running in.

krb5_parse_nametype() parses and returns the name type integer value in *type*. On failure the function returns an error code and set the error string.

SEE ALSO

krb5_425_conv_principal(3), krb5_config(3), krb5.conf(5)

BUGS

You can not have a NUL in a component in some of the variable argument functions above. Until someone can give a good example of where it would be a good idea to have NUL's in a component, this will not be fixed.