

**NAME**

Heimdal Kerberos 5 credential cache functions -

**Functions**

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_register** (krb5\_context context, const krb5\_cc\_ops \*ops, krb5\_boolean override)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_resolve** (krb5\_context context, const char \*name, krb5\_ccache \*id)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_new\_unique** (krb5\_context context, const char \*type, const char \*hint, krb5\_ccache \*id)

KRB5\_LIB\_FUNCTION const char \*KRB5\_LIB\_CALL **krb5\_cc\_get\_name** (krb5\_context context, krb5\_ccache id)

KRB5\_LIB\_FUNCTION const char \*KRB5\_LIB\_CALL **krb5\_cc\_get\_type** (krb5\_context context, krb5\_ccache id)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_get\_full\_name** (krb5\_context context, krb5\_ccache id, char \*\*str)

KRB5\_LIB\_FUNCTION const krb5\_cc\_ops \*KRB5\_LIB\_CALL **krb5\_cc\_get\_ops** (krb5\_context context, krb5\_ccache id)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_switch** (krb5\_context context, krb5\_ccache id)

KRB5\_LIB\_FUNCTION krb5\_boolean KRB5\_LIB\_CALL **krb5\_cc\_support\_switch** (krb5\_context context, const char \*type)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_set\_default\_name** (krb5\_context context, const char \*name)

KRB5\_LIB\_FUNCTION const char \*KRB5\_LIB\_CALL **krb5\_cc\_default\_name** (krb5\_context context)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_default** (krb5\_context context, krb5\_ccache \*id)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_initialize** (krb5\_context context, krb5\_ccache id, krb5\_principal primary\_principal)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_destroy** (krb5\_context context, krb5\_ccache id)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_close** (krb5\_context context, krb5\_ccache id)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_store\_cred** (krb5\_context context, krb5\_ccache id, krb5\_creds \*creds)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_retrieve\_cred** (krb5\_context context, krb5\_ccache id, krb5\_flags whichfields, const krb5\_creds \*mcreds, krb5\_creds \*creds)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_get\_principal** (krb5\_context context, krb5\_ccache id, krb5\_principal \*principal)

KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL **krb5\_cc\_start\_seq\_get** (krb5\_context

```

context, const krb5_ccache id, krb5_cc_cursor *cursor)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_next_cred (krb5_context context,
const krb5_ccache id, krb5_cc_cursor *cursor, krb5_creds *creds)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_end_seq_get (krb5_context
context, const krb5_ccache id, krb5_cc_cursor *cursor)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_remove_cred (krb5_context
context, krb5_ccache id, krb5_flags which, krb5_creds *cred)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_set_flags (krb5_context context,
krb5_ccache id, krb5_flags flags)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_get_flags (krb5_context context,
krb5_ccache id, krb5_flags *flags)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_copy_match_f (krb5_context
context, const krb5_ccache from, krb5_ccache to, krb5_boolean(*match)(krb5_context, void *, const
krb5_creds *), void *matchctx, unsigned int *matched)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_copy_cache (krb5_context context,
const krb5_ccache from, krb5_ccache to)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_get_version (krb5_context context,
const krb5_ccache id)
KRB5_LIB_FUNCTION void KRB5_LIB_CALL krb5_cc_clear_mcred (krb5_creds *mcred)
KRB5_LIB_FUNCTION const krb5_cc_ops *KRB5_LIB_CALL krb5_cc_get_prefix_ops (krb5_context
context, const char *prefix)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_cache_get_first (krb5_context
context, const char *type, krb5_cc_cache_cursor *cursor)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_cache_next (krb5_context context,
krb5_cc_cache_cursor cursor, krb5_ccache *id)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_cache_end_seq_get (krb5_context
context, krb5_cc_cache_cursor cursor)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_cache_match (krb5_context
context, krb5_principal client, krb5_ccache *id)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_move (krb5_context context,
krb5_ccache from, krb5_ccache to)
KRB5_LIB_FUNCTION krb5_boolean KRB5_LIB_CALL krb5_is_config_principal (krb5_context
context, krb5_const_principal principal)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_set_config (krb5_context context,
krb5_ccache id, krb5_const_principal principal, const char *name, krb5_data *data)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cc_get_config (krb5_context context,
krb5_ccache id, krb5_const_principal principal, const char *name, krb5_data *data)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cccol_cursor_new (krb5_context
context, krb5_cccol_cursor *cursor)
KRB5_LIB_FUNCTION krb5_error_code KRB5_LIB_CALL krb5_cccol_cursor_next (krb5_context

```

context, krb5\_cccol\_cursor cursor, krb5\_ccache \*cache)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cccol\_cursor\_free** (krb5\_context context, krb5\_cccol\_cursor \*cursor)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_last\_change\_time** (krb5\_context context, krb5\_ccache id, krb5\_timestamp \*mtime)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cccol\_last\_change\_time** (krb5\_context context, const char \*type, krb5\_timestamp \*mtime)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_get\_friendly\_name** (krb5\_context context, krb5\_ccache id, char \*\*name)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_set\_friendly\_name** (krb5\_context context, krb5\_ccache id, const char \*name)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_get\_lifetime** (krb5\_context context, krb5\_ccache id, time\_t \*t)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_set\_kdc\_offset** (krb5\_context context, krb5\_ccache id, krb5\_deltat offset)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_get\_kdc\_offset** (krb5\_context context, krb5\_ccache id, krb5\_deltat \*offset)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_gen\_new** (krb5\_context context, const krb5\_cc\_ops \*ops, krb5\_ccache \*id) **KRB5\_DEPRECATED\_FUNCTION**('Use X instead')  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_copy\_creds** (krb5\_context context, const krb5\_ccache from, krb5\_ccache to)  
**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_get\_validated\_creds** (krb5\_context context, krb5\_creds \*creds, krb5\_principal client, krb5\_ccache ccache, char \*service)

**Variables**

**KRB5\_LIB\_VARIABLE** const krb5\_cc\_ops **krb5\_acc\_ops**  
**KRB5\_LIB\_VARIABLE** const krb5\_cc\_ops **krb5\_fcc\_ops**  
**KRB5\_LIB\_VARIABLE** const krb5\_cc\_ops **krb5\_mcc\_ops**

**Detailed Description****Function Documentation**

**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_cache\_end\_seq\_get** (krb5\_context context, krb5\_cc\_cache\_cursor cursor)

Destroy the cursor 'cursor'.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION** krb5\_error\_code **KRB5\_LIB\_CALL** **krb5\_cc\_cache\_get\_first** (krb5\_context context, const char \* type, krb5\_cc\_cache\_cursor \* cursor)

Start iterating over all caches of specified type. See also `krb5_cccol_cursor_new()`.

**Parameters:**

*context* A Kerberos 5 context

*type* optional type to iterate over, if NULL, the default cache is used.

*cursor* cursor should be freed with `krb5_cc_cache_end_seq_get()`.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_cache\_match (krb5\_context context, krb5\_principal client, krb5\_ccache \* id)**

Search for a matching credential cache that have the ‘principal’ as the default principal. On success, ‘id’ needs to be freed with `krb5_cc_close()` or `krb5_cc_destroy()`.

**Parameters:**

*context* A Kerberos 5 context

*client* The principal to search for

*id* the returned credential cache

**Returns:**

On failure, error code is returned and ‘id’ is set to NULL.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_cache\_next (krb5\_context context, krb5\_cc\_cache\_cursor cursor, krb5\_ccache \* id)**

Retrieve the next cache pointed to by (‘cursor’) in ‘id’ and advance ‘cursor’.

**Parameters:**

*context* A Kerberos 5 context

*cursor* the iterator cursor, returned by `krb5_cc_cache_get_first()`

*id* next ccache

**Returns:**

Return 0 or an error code. Returns KRB5\_CC\_END when the end of caches is reached, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION void KRB5\_LIB\_CALL krb5\_cc\_clear\_mcred (krb5\_creds \* mcred)**

Clear ‘mcreds’ so it can be used with `krb5_cc_retrieve_cred`

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_close (krb5\_context context,**

**krb5\_ccache id)**

Stop using the ccache 'id' and free the related resources.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_copy\_cache (krb5\_context context, const krb5\_ccache from, krb5\_ccache to)**

Just like `krb5_cc_copy_match_f()`, but copy everything.

@

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_copy\_creds (krb5\_context context, const krb5\_ccache from, krb5\_ccache to)**

MIT compat glue

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_copy\_match\_f (krb5\_context context, const krb5\_ccache from, krb5\_ccache to, krb5\_boolean(\*)(krb5\_context, void \*, const krb5\_creds \*) match, void \* matchctx, unsigned int \* matched)**

Copy the contents of 'from' to 'to' if the given match function return true.

**Parameters:**

*context* A Kerberos 5 context.

*from* the cache to copy data from.

*to* the cache to copy data to.

*match* a match function that should return TRUE if cred argument should be copied, if NULL, all credentials are copied.

*matchctx* context passed to match function.

*matched* set to true if there was a credential that matched, may be NULL.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_default (krb5\_context context, krb5\_ccache \* id)**

Open the default ccache in 'id'.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION const char\* KRB5\_LIB\_CALL krb5\_cc\_default\_name (krb5\_context context)**

Return a pointer to a context static string containing the default ccache name.

**Returns:**

String to the default credential cache name.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_destroy (krb5\_context context, krb5\_ccache id)**

Remove the ccache 'id'.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_end\_seq\_get (krb5\_context context, const krb5\_ccache id, krb5\_cc\_cursor \* cursor)**

Destroy the cursor 'cursor'.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_gen\_new (krb5\_context context, const krb5\_cc\_ops \* ops, krb5\_ccache \* id)**

Generate a new ccache of type 'ops' in 'id'.

Deprecated: use `krb5_cc_new_unique()` instead.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_get\_config (krb5\_context context, krb5\_ccache id, krb5\_const\_principal principal, const char \* name, krb5\_data \* data)**

Get some configuration for the credential cache in the cache.

**Parameters:**

*context* a Keberos context

*id* the credential cache to store the data for

*principal* configuration for a specific principal, if NULL, global for the whole cache.

*name* name under which the configuraion is stored.

*data* data to fetched, free with `krb5_data_free()`

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_get\_flags (krb5\_context context, krb5\_ccache id, krb5\_flags \* flags)**

Get the flags of 'id', store them in 'flags'.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_get\_friendly\_name**  
(*krb5\_context* context, *krb5\_ccache* id, *char \*\** name)

Return a friendly name on credential cache. Free the result with `krb5_xfree()`.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_get\_full\_name** (*krb5\_context* context, *krb5\_ccache* id, *char \*\** str)

Return the complete resolvable name the cache

**Parameters:**

*context* a Keberos context

*id* return pointer to a found credential cache

*str* the returned name of a credential cache, free with `krb5_xfree()`

**Returns:**

Returns 0 or an error (and then *\*str* is set to NULL).

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_get\_kdc\_offset** (*krb5\_context* context, *krb5\_ccache* id, *krb5\_deltat* \* offset)

Get the time offset between the client and the KDC

If the backend doesn't support KDC offset, use the context global setting.

**Parameters:**

*context* A Kerberos 5 context.

*id* a credential cache

*offset* the offset in seconds

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_get\_lifetime** (*krb5\_context* context, *krb5\_ccache* id, *time\_t* \* t)

Get the lifetime of the initial ticket in the cache

Get the lifetime of the initial ticket in the cache, if the initial ticket was not found, the error code `KRB5_CC_END` is returned.

**Parameters:**

*context* A Kerberos 5 context.  
*id* a credential cache  
*t* the relative lifetime of the initial ticket

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION const char\* KRB5\_LIB\_CALL krb5\_cc\_get\_name (krb5\_context context, krb5\_ccache id)**

Return the name of the ccache 'id'

**KRB5\_LIB\_FUNCTION const krb5\_cc\_ops\* KRB5\_LIB\_CALL krb5\_cc\_get\_ops (krb5\_context context, krb5\_ccache id)**

Return `krb5_cc_ops` of a the ccache 'id'.

**KRB5\_LIB\_FUNCTION const krb5\_cc\_ops\* KRB5\_LIB\_CALL krb5\_cc\_get\_prefix\_ops (krb5\_context context, const char \* prefix)**

Get the cc ops that is registered in 'context' to handle the prefix. prefix can be a complete credential cache name or a prefix, the function will only use part up to the first colon (:) if there is one. If prefix the argument is NULL, the default ccache implementation is returned.

**Returns:**

Returns NULL if ops not found.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_get\_principal (krb5\_context context, krb5\_ccache id, krb5\_principal \* principal)**

Return the principal of 'id' in 'principal'.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION const char\* KRB5\_LIB\_CALL krb5\_cc\_get\_type (krb5\_context context, krb5\_ccache id)**

Return the type of the ccache 'id'.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_get\_version (krb5\_context context, const krb5\_ccache id)**

Return the version of 'id'.



**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_initialize (krb5\_context context, krb5\_ccache id, krb5\_principal primary\_principal)**

Create a new ccache in 'id' for 'primary\_principal'.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_last\_change\_time (krb5\_context context, krb5\_ccache id, krb5\_timestamp \* mtime)**

Return the last time the credential cache was modified.

**Parameters:**

*context* A Kerberos 5 context

*id* The credential cache to probe

*mtime* the last modification time, set to 0 on error.

**Returns:**

Return 0 or and error. See `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_move (krb5\_context context, krb5\_ccache from, krb5\_ccache to)**

Move the content from one credential cache to another. The operation is an atomic switch.

**Parameters:**

*context* a Keberos context

*from* the credential cache to move the content from

*to* the credential cache to move the content to

**Returns:**

On success, *from* is freed. On failure, error code is returned and *from* and *to* are both still allocated, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_new\_unique (krb5\_context context, const char \* type, const char \* hint, krb5\_ccache \* id)**

Generates a new unique ccache of 'type' in 'id'. If 'type' is NULL, the library chooses the default credential cache type. The supplied 'hint' (that can be NULL) is a string that the credential cache type can use to base the name of the credential on, this is to make it easier for the user to differentiate the credentials.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_next\_cred (krb5\_context context, const krb5\_ccache id, krb5\_cc\_cursor \* cursor, krb5\_creds \* creds)**

Retrieve the next cred pointed to by ('id', 'cursor') in 'creds' and advance 'cursor'.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_register (krb5\_context context, const krb5\_cc\_ops \* ops, krb5\_boolean override)**

Add a new ccache type with operations 'ops', overwriting any existing one if 'override'.

**Parameters:**

*context* a Keberos context

*ops* type of plugin symbol

*override* flag to select if the registration is to override an existing ops with the same name.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_remove\_cred (krb5\_context context, krb5\_ccache id, krb5\_flags which, krb5\_creds \* cred)**

Remove the credential identified by 'cred', 'which' from 'id'.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_resolve (krb5\_context context, const char \* name, krb5\_ccache \* id)**

Find and allocate a ccache in 'id' from the specification in 'residual'. If the ccache name doesn't contain any colon, interpret it as a file name.

**Parameters:**

*context* a Keberos context.

*name* string name of a credential cache.

*id* return pointer to a found credential cache.

**Returns:**

Return 0 or an error code. In case of an error, id is set to NULL, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_retrieve\_cred (krb5\_context context, krb5\_ccache id, krb5\_flags whichfields, const krb5\_creds \* mcreds, krb5\_creds \* creds)**

Retrieve the credential identified by 'mcreds' (and 'whichfields') from 'id' in 'creds'. 'creds' must be free by the caller using `krb5_free_cred_contents`.

**Parameters:**

*context* A Kerberos 5 context

*id* a Kerberos 5 credential cache

*whichfields* what fields to use for matching credentials, same flags as whichfields in

**krb5\_compare\_creds()**

*mcreds* template credential to use for comparing

*creds* returned credential, free with **krb5\_free\_cred\_contents()**

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_set\_config (krb5\_context context, krb5\_ccache id, krb5\_const\_principal principal, const char \* name, krb5\_data \* data)**

Store some configuration for the credential cache in the cache. Existing configuration under the same name is over-written.

**Parameters:**

*context* a Kerberos context

*id* the credential cache to store the data for

*principal* configuration for a specific principal, if NULL, global for the whole cache.

*name* name under which the configuration is stored.

*data* data to store, if NULL, configuration is removed.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_set\_default\_name (krb5\_context context, const char \* name)**

Set the default cc name for 'context' to 'name'.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_set\_flags (krb5\_context context, krb5\_ccache id, krb5\_flags flags)**

Set the flags of 'id' to 'flags'.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_set\_friendly\_name (krb5\_context context, krb5\_ccache id, const char \* name)**

Set the friendly name on credential cache.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_set\_kdc\_offset (krb5\_context context, krb5\_ccache id, krb5\_deltat offset)**

Set the time offset between the client and the KDC

If the backend doesn't support KDC offset, use the context global setting.

**Parameters:**

*context* A Kerberos 5 context.

*id* a credential cache

*offset* the offset in seconds

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_start\_seq\_get (krb5\_context context, const krb5\_ccache id, krb5\_cc\_cursor \* cursor)**

Start iterating over 'id', 'cursor' is initialized to the beginning. Caller must free the cursor with `krb5_cc_end_seq_get()`.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_store\_cred (krb5\_context context, krb5\_ccache id, krb5\_creds \* creds)**

Store 'creds' in the ccache 'id'.

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_boolean KRB5\_LIB\_CALL krb5\_cc\_support\_switch (krb5\_context context, const char \* type)**

Return true if the default credential cache support switch

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cc\_switch (krb5\_context context, krb5\_ccache id)**

Switch the default default credential cache for a specific credcache type (and name for some implementations).

**Returns:**

Return an error code or 0, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cccol\_cursor\_free (krb5\_context context, krb5\_cccol\_cursor \* cursor)**

End an iteration and free all resources, can be done before end is reached.

**Parameters:**

*context* A Kerberos 5 context

*cursor* the iteration cursor to be freed.

**Returns:**

Return 0 or and error, KRB5\_CC\_END is returned at the end of iteration. See `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cccol\_cursor\_new (krb5\_context context, krb5\_cccol\_cursor \* cursor)**

Get a new cache iteration cursor that will iterate over all credentials caches independent of type.

**Parameters:**

*context* a Keberos context

*cursor* passed into `krb5_cccol_cursor_next()` and free with `krb5_cccol_cursor_free()`.

**Returns:**

Returns 0 or and error code, see `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cccol\_cursor\_next (krb5\_context context, krb5\_cccol\_cursor cursor, krb5\_ccache \* cache)**

Get next credential cache from the iteration.

**Parameters:**

*context* A Kerberos 5 context

*cursor* the iteration cursor

*cache* the returned cursor, pointer is set to NULL on failure and a cache on success. The returned cache needs to be freed with `krb5_cc_close()` or destroyed with `krb5_cc_destroy()`. MIT Kerberos behaves slightly different and sets cache to NULL when all caches are iterated over and return 0.

**Returns:**

Return 0 or and error, KRB5\_CC\_END is returned at the end of iteration. See `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION krb5\_error\_code KRB5\_LIB\_CALL krb5\_cccol\_last\_change\_time (krb5\_context context, const char \* type, krb5\_timestamp \* mtime)**

Return the last modification time for a cache collection. The query can be limited to a specific cache type. If the function return 0 and *mtime* is 0, there was no credentials in the caches.

**Parameters:**

*context* A Kerberos 5 context

*type* The credential cache to probe, if NULL, all type are traversed.

*mtime* the last modification time, set to 0 on error.

**Returns:**

Return 0 or and error. See `krb5_get_error_message()`.

**KRB5\_LIB\_FUNCTION** `krb5_error_code` **KRB5\_LIB\_CALL** `krb5_get_validated_creds` (`krb5_context context, krb5_creds * creds, krb5_principal client, krb5_ccache ccache, char * service`)

Validate the newly fetch credential, see also `krb5_verify_init_creds()`.

**Parameters:**

*context* a Kerberos 5 context

*creds* the credentials to verify

*client* the client name to match up

*ccache* the credential cache to use

*service* a service name to use, used with `krb5_sname_to_principal()` to build a hostname to use to verify.

**KRB5\_LIB\_FUNCTION** `krb5_boolean` **KRB5\_LIB\_CALL** `krb5_is_config_principal` (`krb5_context context, krb5_const_principal principal`)

Return TRUE (non zero) if the principal is a configuration principal (generated part of `krb5_cc_set_config()`). Returns FALSE (zero) if not a configuration principal.

**Parameters:**

*context* a Keberos context

*principal* principal to check if it a configuration principal

**Variable Documentation**

**KRB5\_LIB\_VARIABLE** `const` `krb5_cc_ops` `krb5_acc_ops`

**Initial value:**

```
{
    KRB5_CC_OPS_VERSION,
    'API',
    acc_get_name,
```

```
acc_resolve,  
acc_gen_new,  
acc_initialize,  
acc_destroy,  
acc_close,  
acc_store_cred,  
NULL,  
acc_get_principal,  
acc_get_first,  
acc_get_next,  
acc_end_get,  
acc_remove_cred,  
acc_set_flags,  
acc_get_version,  
acc_get_cache_first,  
acc_get_cache_next,  
acc_end_cache_get,  
acc_move,  
acc_get_default_name,  
acc_set_default,  
acc_lastchange,  
NULL,  
NULL,  
}
```

Variable containing the API based credential cache implementation.

**KRB5\_LIB\_VARIABLE** const krb5\_cc\_ops krb5\_fcc\_ops

**Initial value:**

```
{  
  KRB5_CC_OPS_VERSION,  
  'FILE',  
  fcc_get_name,  
  fcc_resolve,  
  fcc_gen_new,  
  fcc_initialize,  
  fcc_destroy,  
  fcc_close,  
  fcc_store_cred,  
  NULL,  
}
```

```

fcc_get_principal,
fcc_get_first,
fcc_get_next,
fcc_end_get,
fcc_remove_cred,
fcc_set_flags,
fcc_get_version,
fcc_get_cache_first,
fcc_get_cache_next,
fcc_end_cache_get,
fcc_move,
fcc_get_default_name,
NULL,
fcc_lastchange,
fcc_set_kdc_offset,
fcc_get_kdc_offset
}

```

Variable containing the FILE based credential cache implementation.

### **KRB5\_LIB\_VARIABLE** const krb5\_cc\_ops krb5\_mcc\_ops

**Initial value:**

```

{
  KRB5_CC_OPS_VERSION,
  'MEMORY',
  mcc_get_name,
  mcc_resolve,
  mcc_gen_new,
  mcc_initialize,
  mcc_destroy,
  mcc_close,
  mcc_store_cred,
  NULL,
  mcc_get_principal,
  mcc_get_first,
  mcc_get_next,
  mcc_end_get,
  mcc_remove_cred,
  mcc_set_flags,
  NULL,
}

```



```
mcc_get_cache_first,  
mcc_get_cache_next,  
mcc_end_cache_get,  
mcc_move,  
mcc_default_name,  
NULL,  
mcc_lastchange,  
mcc_set_kdc_offset,  
mcc_get_kdc_offset  
}
```

Variable containing the MEMORY based credential cache implementation.