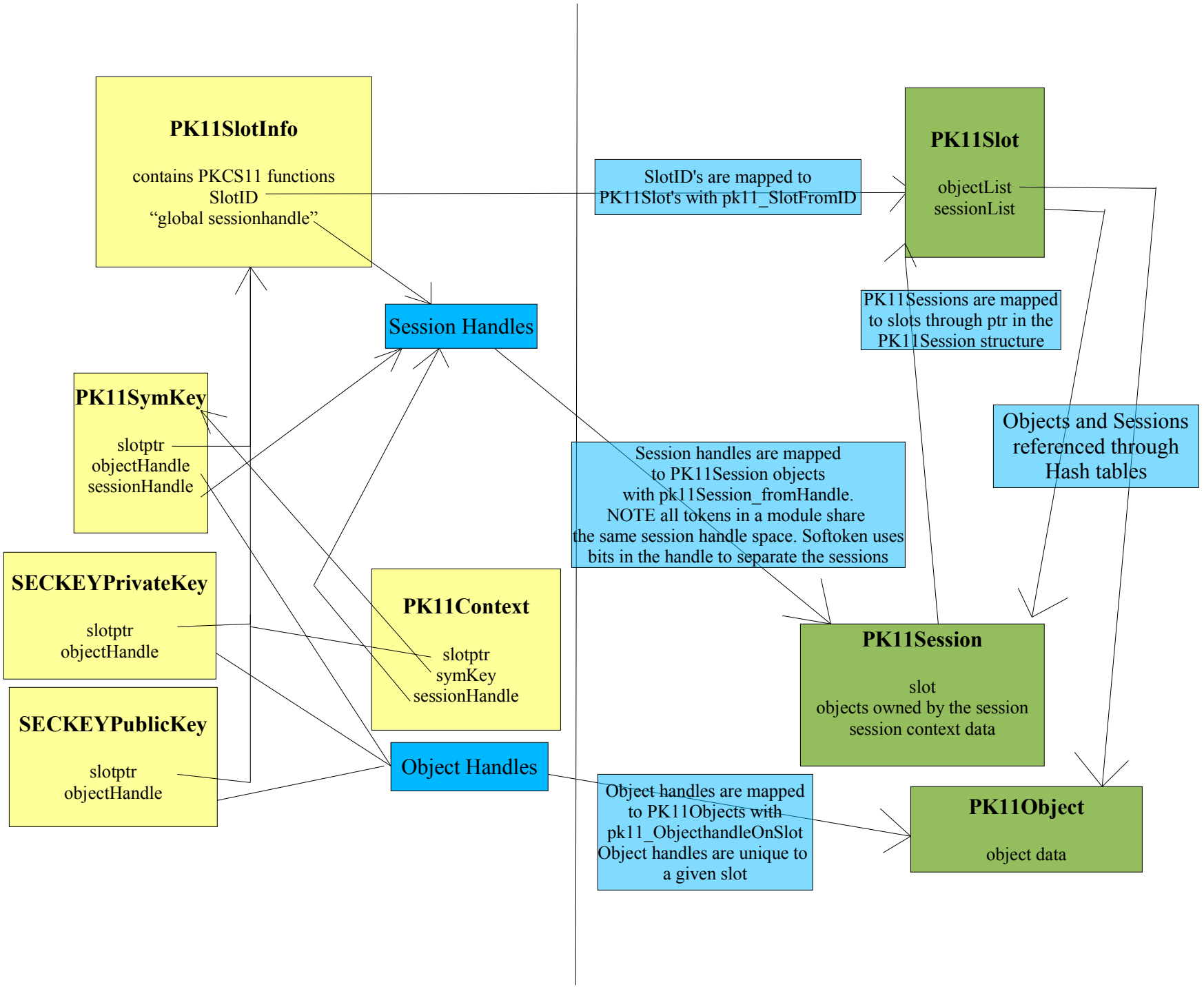

PK11SlotInfo
contains PKCS11 functions
SlotID
“global sessionhandle”
SlotID's are mapped to PK11Slot's with pk11_SlotFromID
PK11Slot
objectList
sessionList
Session Handles
PK11Sessions are mapped to slots through ptr in the PK11Session structure
PK11SymKey
slotptr
objectHandle
sessionHandle
Objects and Sessions referenced through Hash tables
Session handles are mapped to PK11Session objects with pk11Session_fromHandle. NOTE all tokens in a module share the same session handle space. Softoken uses bits in the handle to separate the sessions
SECKEYPrivateKey
slotptr
objectHandle
PK11Context
slotptr
symKey
sessionHandle
PK11Session
slot
objects owned by the session
session context data
SECKEYPublicKey
slotptr
objectHandle
Object Handles
Object handles are mapped to PK11Objects with pk11_ObjecthandleOnSlot Object handles are unique to a given slot
PK11Object
object data

The way PKCS #11 is defined:

SlotIDs are unique within a Module.

Session Handles point to unique sessions within a Module or Library. Each session lives on the Slot it was created on. Several PKCS #11 calls take only the Session Handle (not the Slot ID), so the Session Handle implies a slot. Session Handles are provided for 3 purposes: 1) to identify a slot, 2) to manage objects (all session objects created with a give Session Handle are automatically destroyed when the session is closed, and 3) to carry intermediate context for ongoing pkcs #11 operations (multipart encrypt, hash, find, etc). Session handles are also the PKCS #11 threading primitive. For thread safe operations, the application is responsible to make sure no to calls into PKCS #11 on the same thread should overlap. This means once you find your Session object, access to context data on that object is safe.

Object handles are unique within a Slot. An object handle without a slot (or a session which implies the slot) is meaningless. All object handles must be visible to all sessions (independent on whether or not they are referenced from the session that 'owns' them).

The way Softoken defines Handles:

SlotIDs are predefined as follows:

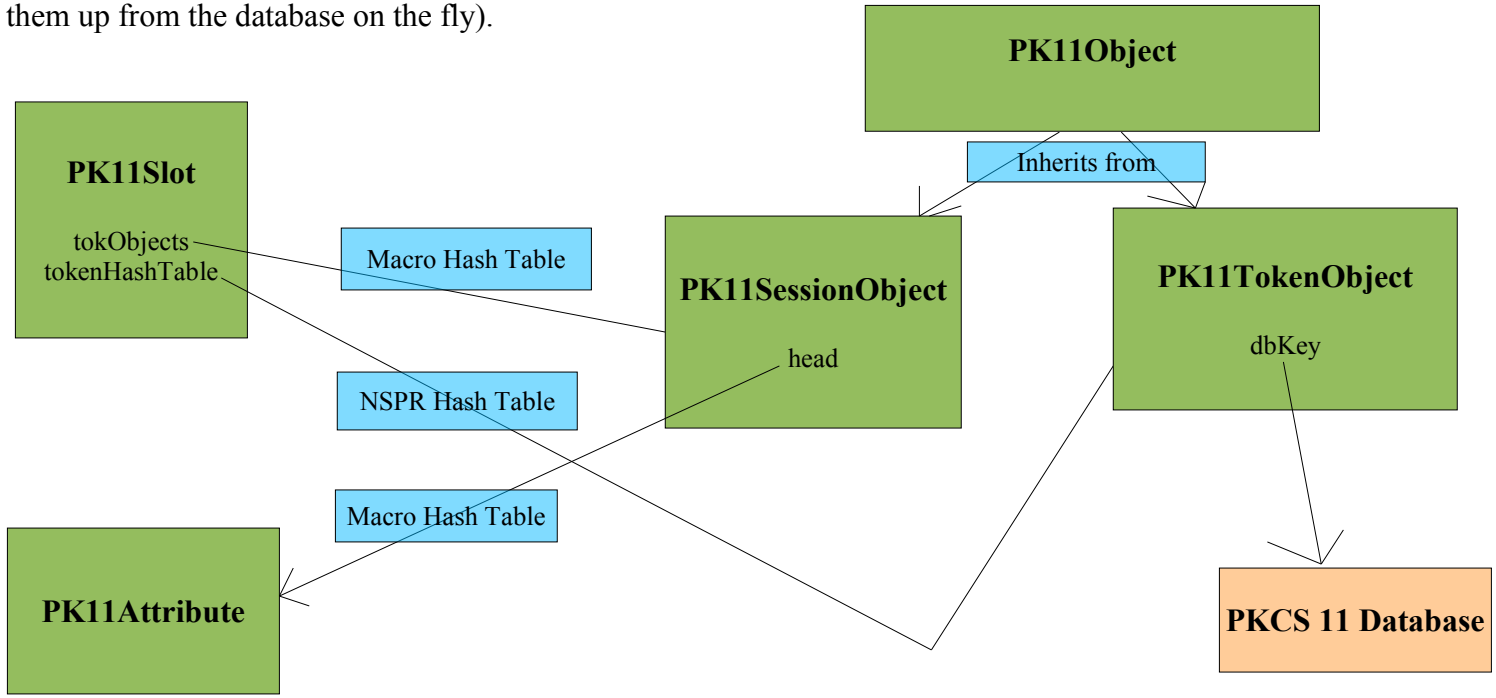
NETSCAPE_SLOT_ID=1 (Crypto module)
PRIVATE_KEY_SLOT_ID=2 (Database module),
FIPS_SLOT_ID=3 (FIPS module)

In addition Softoken can handle user defined slots. These have never been used to my knowledge. Softoken looks up the PK11Slot structures from the SlotIDs using an NSPR hash table. There is no locking (it only changes on initialization and shutdown). The table has at most 2 entries currently (NETSCAPE_SLOT_ID and PRIVATE_KEY_SLOT_ID).

Session Handles are defined so the upper 8 bits can identify the slot. The remaining 24 bits are used to distinguished between sessions on a given slot. By splitting the address space like this, each slot can keep track of it's own session list rather than having a global session list for the module. Sessions are looked up in a hash table (implmented with macros). To spread out the locking, the hashtables are split in groups of buckets. Tuning defines control how many buckets are covered with a single lock (trading off creation of lots of locks with lock contention of so many buckets). These tables live in the PK11Slot structure. New Session handles are allocated sequentially, starting at 1, verifying that the new session handle doesn't already exist (wrap around case).

Object Handles are alocated sequentially for session object (hmm I'm not sure we check wrap around in ObjectHandles). Token objects are allocated by hashing the database key and checking for collisions. Session objects all have the high bit set to '0' and token objects have the high bit set to 0x8. They are in a different hash table than the session objects (an NSPR hash table named tokenHashTable in the PK11Slot structure). Session objects are stored in a poorly named field called 'tokObjects', which is a macro based hash table similar to the one used to store Session Handles. The difference is there is only one lock protecting all session objects (no bucketted locks).

NOTES about **PK11Object**, **PK11SessionObject** and **PK11TokenObject**. Think of PK11Object as a superclass. Most of the common object stuff is stored in PK11Object. Most internal calls in softoken take PK11Object. PK11SessionObject and PK11TokenObject each have a PK11Object imbedded in them. You can use the 'narrowTo' calls to get a PK11SessionObject or a PK11TokenObject from a PK11Object. 'narrowTo' simply does a check on the handle and then a cast (so if the handle is messed up, you're hosed). PK11SessionObjects contain lists of attributes, again accessed by a hash table (I think this is where nelson believes we can probably completely dispense with the locks – I think he's probably right, with a little work). BTW this part can be completely changed to store attributes in other ways without changing a lot of code (all attributes are accessed through a limitted sets of functions... in fact TokenObjects do not store attributes in any sort of table at all, but generate them or look them up from the database on the fly).



How pk11wrap (above the line) uses these handles.

SlotID – Obviously, they are the basic access to the Slot. Pk11wrap uses PK11SlotInfo to store them, and cached information about the slot, including a pointer to the function table.

Session Handles –

First, pk11wrap creates a global session handle for 3 purposes.

1) for use if we can't get any new session handles from the token (typically this is only single session smart cards),
2) for use in operations that don't need to be overlapped. This list is dwindling. It used to be sign, verify, find, etc operations used this single session. I'm not sure if find operations still use it (I believe with stan, it uses a newly created session for that purpose). **I think temparary public and private keys are still created on this session**.
3) to detect token changes. If you remove a token and reinsert it, the only way to truly tell that that has happenned to to notice that the session has become invalid.

Second, a new R/W session is created whenever we need to create a token object (doesn't happen in servers).

Third, any sign or verify operation creates a new session to hold the sign and verify context. This prevents lock contention on the sign or verify.

Fourth, any new symetric key is created with it's own session, again to avoid lock contention when dealing with the keys. Notable exceptions are: 1) token symetric keys (not applicable to servers), and 2) SSL derived keys, which all share the same session. Symetric Keys try to preserve the session when they are placed on the free list to limit the expense of multiple C_OpenSession calls. There's a bug about how this is self-defeating for SSL operations.

Finally, any PK11Context has it's own session, to keep track of whatever crypto state it's managing.

Object Handles -

For token objects, Object handles are 'looked up'. The various users of these handles are marked as 'not the owner' so when the pk11wrap data structure associated with the object handle goes away, the object handle is not explicitly

deleted. Cert and Crl objects presume that their object handles are token objects.

Life Cycle in the softoken.

PK11Slot objects are created at init time and remain until finalize. They are allocated and freed from the heap.

PK11Session objects are created whenever C_OpenSession is called. Looking at the list above, you can see about how often that is. They are freed when the Session is closed (C_CloseSession, C_CloseAllSessions, C_Finalize). Freeing session objects 1) frees all the objects associated with the session, frees any intermediate contexts that have been allocated (these contexts are only allocated as needed), and frees the session itself. With the exception of objects, all these objects are freed back to the heap. (PORT_Free)

PK11SessionObjects are created when the SessionObject is created in C_CreateObject. They are freed either 1) when the session closes, or 2) when it explicitly destroyed by C_DestroyObject. When a session object is destroyed, all if it's attributes are clear. This is controlled with a #define, in theory you could build NSS to use a number of schemes. In practice we only ever use PKCS11_STATIC_ATTRIBUTES. In the STATIC_ATTRIBUTE scheme, a fixed number of attributes 'objects' (MAX_OBJ_ATTRS) are allocated when the object is allocated (contiguous with the session object data structure). Each of these attribute 'objects' has space to hold an attribute value of up to ATTR_SPACE bytes (big enough to hold an SSL master secret, but not big enough to hold a cert or a modulus). If an attribute with size greater than ATTR_SPACE is stored in a session object, the space for the value of that attribute is allocated from the heap. When the session object is destroyed, all the attributes in their array are walked down, data is cleared, and heap allocated data is freed.

Session objects are managed with a session object free list. There is a runtime variable which controls whether or not a free list is kept of session objects. This variable is called 'optimizeSpace'. Object Free lists are capped at MAX_OBJECT_LIST_SIZE. If the free list is that large, no new objects are added to that list and the additional objects are freed to the heap. If there are no objects on the free list, the objects are allocated out of the heap. Attribute locks and object reference locks are only freed if the object is returned to the heap.

PK11TokenObjects are created on the fly in the softoken whenever the object is referenced. The object is 'freed' before the PK11 call has returned to the caller (and created again on the next call). PK11TokenObjects do no store their attributes in the object themselves. The data for the objects are either 1) fixed for all tokens of that class, or 2) looked up from the database. Temporary PK11Attribute objects are allocated from the heap at 'pk11_FindAttribute()' time and returned to the heap at

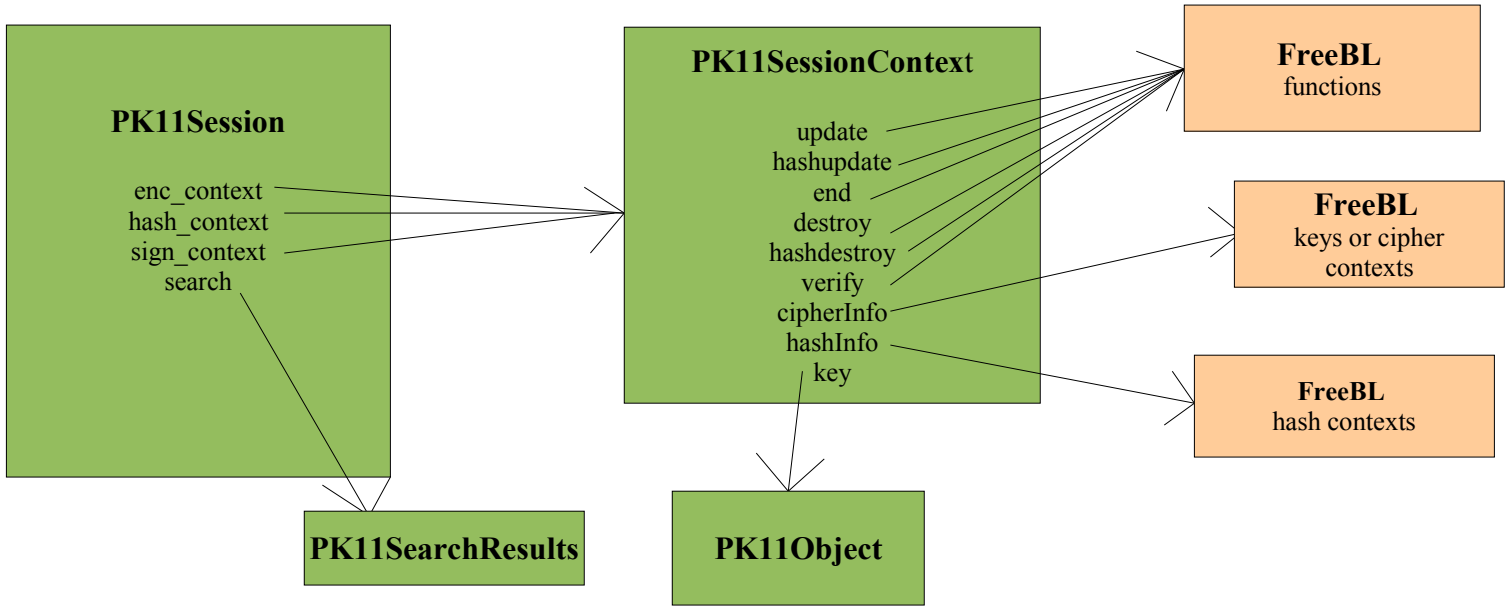
pk11_FreeAttribute time.

Token objects are managed with a parallel object free list. It has the same semantics except token objects do not have attribute locks

PK11Attribute
Logically the code is written assuming that functions like pk11_FindAttribute() and pk11_NewAttribute() return fresh references to an attribute object, which must be freed with pk11_FreeAttribute(). If actual fact these attributes may be 1) a static attribute from a session object, 2) a static global attribute (like {CKA_TOKEN, true, 1}), or 3) a dynamically created attribute for a token object. For cases 1-2, pk11_FreeAttribute() is a noop, the attribute will be recycled. For case 3, pk11_FreeAttribute will return the attribute and possibly allocated data for that attribute back to the heap.

New attributes for session objects are allocated from the session object array. If we go passed MAX_OBJS_ATTRS in the session object, pk11_NewAttribute will fail. MAX_OBJS_ATTRS is set to be a little larger the the maximum attribute count of the objects we support. New attributes for token objects are allocated on the fly from the heap.

PK11Session Data in more detail:



When PK11Sessions are first created, there have neither any PK11SessionContexts nor any PK11SearchResults allocated.

PK11SearchSearch results are allocated whenever C_FindObjectInit is called. They are allocated from the heap. Allong with the Search results, an array of handles, stored in the search results is allocated. This array starts at a fixed size and grows ans necessary to hold all the found objects. This array is also allocated on the heapThey are freed when either 1) C_FindObjectsFinal is called, 2) when a subsequent C_FindObjectsInit is called, and 3) when the session is closed. Note: in the normal course of an SSL server session C_FindObjects* is not called unless processing a client auth cert chain.

PK11SessionContexts are allocated whenever the appropriate C_XXXXInit function is called. Because multiple operations are allowed on a single session, the PK11Session has contexts for each of the classes of operations that can be operating at the same time. No matter what C_ function is called to initialize, they all use pk11_InitGeneric to create their PK11SessionContext. The pk11_InitGeneric function also initializes the 'key' field in the PK11SessionContext. Once the context is created, softoken initializes the cipherInfo, hashInfo, and FreeBL function pointers based on the operation and selected mechanism. The cipherInfo and hash values are usually created using the FreeBL XXX_CreateContext() function (which is probably allocated out of the heap). PK11SessionContexts are freed back to the heap when C_XXXXFinal is called, or when the session is closed. Unlike PK11_FindObjectsInit, calls to C_XXXInit while an operation is active is illegal and PKCS 11 and softoken returns CKR_OPERATION_ACTIVE in that case. When the contexts are freed, the FreeBL contexts are also free. Finally, for RSA and DSA operations, the FreeBL contexts are actually the FreeBL representations of the RSA and DSA keys (SECKEYLowPrivateKey, etc).

Life Cycle in pk11wrap

PK11SlotInfo objects are created whenever a module is loaded, and not freed until the module is unloaded and all the objects referencing it is destroyed. PK11SlotInfo objects are allocated from the heap, and they are stored on several lists to aid finding them.

PK11SymKeys are allocated whenever a symetric key is 1) Generated, 2) Unwrapped, 3) Derived, 4) Imported, 5) Looked up, or Created from a handle. In addition, new SymKeys may be created by NSS using the previous mechanisms if it becomes necessary to move a key to a different token in order to complete an operation. In general, SymKeys have are created on their own session. The exceptions to this rule are 1) it was not possible to get a session for this symkey because the token only supports limitted sessions, 2) the symKey points to a set of Keys derived from the master secret in an SSL session. Cases 1 the key contains the Slot's global session handle. Case 2 contains the parent key's session handle. Because PKCS #11 requires serializing calls which are made against the same session, all operations for cases (1 & 2) are serialized with the slot's session

lock. The sessions for the symKey are used in it's own generate, derive, unwrap, createobject, and destroyobject.

SymKeys are created by pulling entries off the freeList. Each PK11SlotInfo maintains is own freelist of keys because the session handle for the key is associated with that slot. If the key on the freeList has a valid session, that session is used, otherwise a new session is created. NOTE: In case 2 where the key doesn't own the session (above), that session is then immediately closed.

SymKeys are freed when all their references are freed. Typically that would be when the crypto operation the key was created for is completed. Some SymKeys may be held arround for long periods of time for ongoing operations (wrapping keys for instance). When the SymKey is freed, the object handle for the key is destroyed (exception – token Symkeys), and the key is added to the freeList (sessions and locks intact).

PK11Contexts are allocated as a result of PK11_CreateContextXXX. Contexts have their own sessions. Again, contexts can share the global session but using save and restore state. I doubt this has every really been tested. Contexts create their own sessions at creation time. Contexts are allocated out of the heap. When an operation is complete, Contexts are destroyed using PK11_DestroyContext. At this time any key references are freed, the Context Session handle is destroyed, and the Context is returned to the heap.