Using HashTables in FDK's

**This note describes how to use hashtables when making modules for FrameViewer, FrameMaker and FrameMaker+SGML.
The motivation for writing this note is that the FDK documentation is unclear about memory issues when using HashTables.**

Version 1.10

**Copyright by Jesper Storm Bache, February 1998.
jesper@bache.com**

Latest version always found at www.bache.com/sgml.html

See section for 1.2 info. about "price"

1.1 Introduction

This note is a programmer oriented note and assumes knowledge of C++, the FrameMaker's Development Kit (FDK) and hashtables.

The note deals with version 5.5 of FrameMaker, but to the best of my knowledge it is also relevant to version 5.1 (and maybe 4.x). This note only deals with allocation and deallocation of memory associated to hashtables.

The FDK has a set of instructions dealing with hashtables. These are named F_Hashxx, where x is the specific function. The following instructions are relevant to memory handling:

- F_HashCreate
- F_HashDestroy
- F_HashSet

When you create a hashtable using the F_HashCreate you are asked to supply a number of parameters. Among these are: "keyLen" and "removeNotify". keyLen is a value determining the length of the hash-key. removeNotify is a procedure pointer.'

The exact meaning of "keyLen" is dependent on the key-type. If you want to use a string as the key you pass "KEY_IS_STRING" as the value otherwise you should pass the sized of the key (for example sizeof(<datatype>)) for the used datatype.

Memory allocation

The FDK always copies the key. This means that you can dispose of the value of the key after using a F_HashSet. If you use strings for the key you might have something like:

```
StringT thekeyName = F_ApiGetString(theDocId, theObj, FP_Name);

F_HashSet(theHashTable, thekeyName, theDocumentName);

F_Free(thekeyName);
```

Notice that thekeyName is Free'ed after passing it to the F_HashSet procedure.

The procedure you pass as the removeNotify parameter, should deal with deallocating the values of the hashtable.

Memory deallocation

You should only free the hash-values if they are memory pointers.

For example if the hashvalue is an integer (IntT), you don't have to worry about deallocation the value.

If the hashvalues are strings you must free the memory. Example:

```
VoidT removeNotify(GenericT dd) {
   F_Free(StringT(dd));
}
```

If the hashvalues are references to objects you would use:

```
VoidT removeNotify(GenericT dd) {
   myObjectType *theObj = (myObjectType *) dd;
   delete(theObj);
}
```

Misc notes

When you create a hashtable, you pass a reference to a "deadq" function. This reference should always be either NULL or point to a function that always return False. Why? Well - trust me (and the Adobe support people that I have been in touch with on this matter). The next FDK documentation will show an example where the deadq function returns False instead of True. If the function returns True, then you'll experience that entries are overwritten by other entries with a different key. This happens is the two different keys have the same hashing value.

1.2 Price

The information in this document is "Helpware". If you find it useful please donate $5 - $10 to your favorite grassroot organization and send a mail to donations@bache.com. Subject: HelpWare. Contents: amount and organization.

1.3 Version history

1.3.1 Version 1.00

First draft.