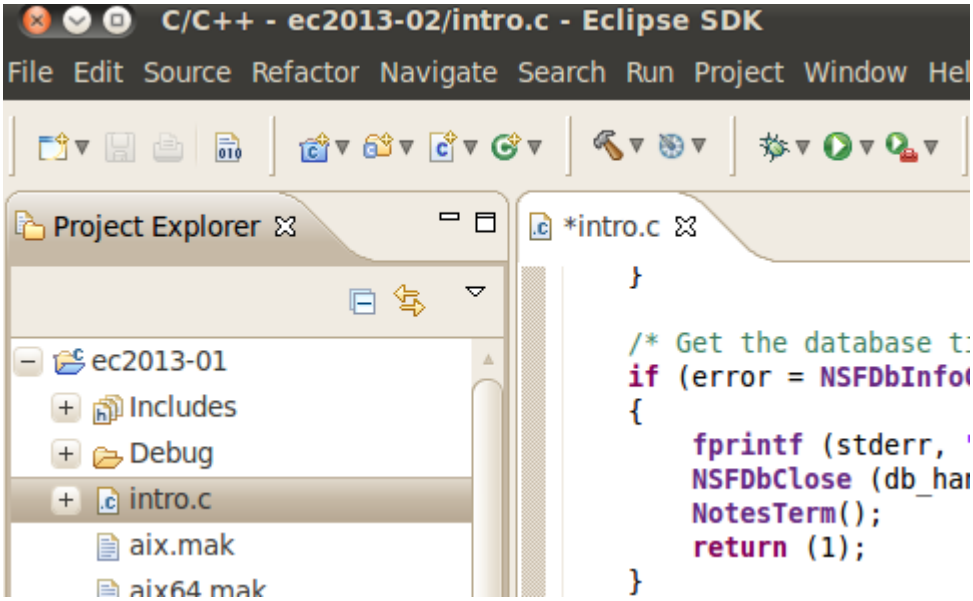
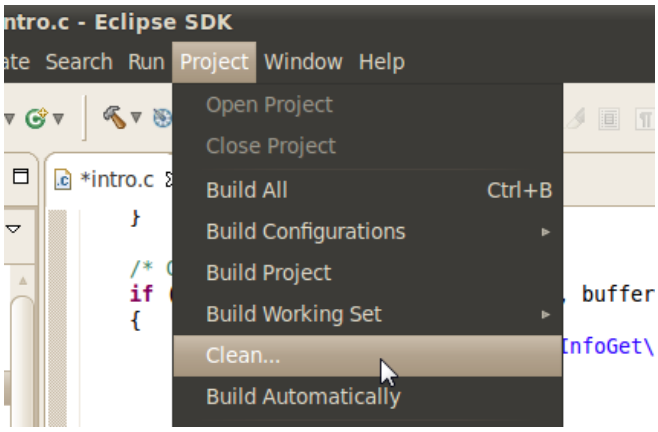
Starten der Umgebung

Nachdem die VM gestartet ist und sich darin angemeldet wurde, starten wir Eclipse.

Wir wählen links ein Projekt aus öffnen dann unser Projekt und dort das Programm intro.c

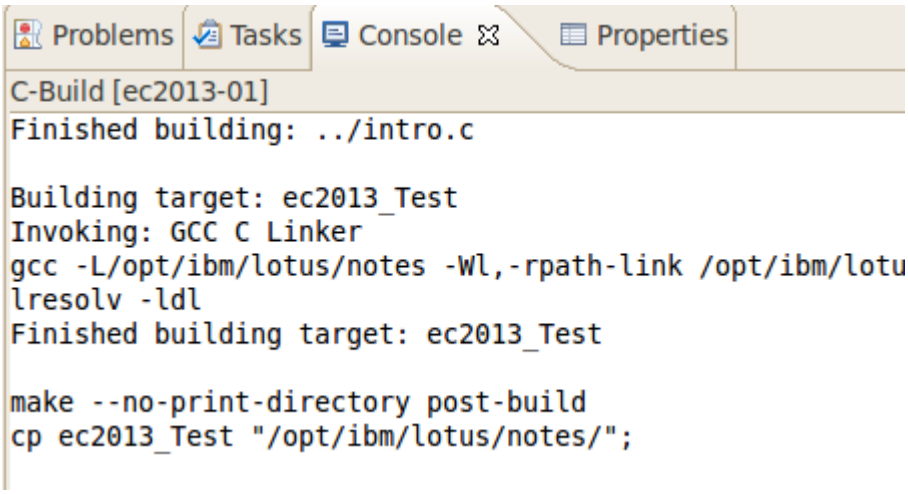


Sicherheitshalber löschen wir alle ggf vorhandenen Zwischendateien über das Projektmenü:

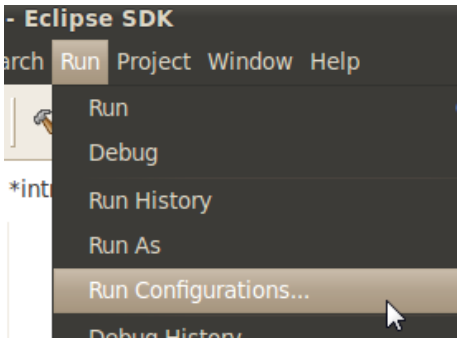


Das folgende Popup beantworten wir mit OK. Jetzt können wir das Projekt mit Build Project wieder neu aufbauen.

Das Ergebnis sehen wir dann unten in der Konsole:



Um das Programm zu testen wählen wir im Menü Run dann



Das erste Programm

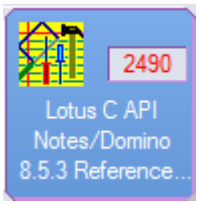
Zuerst kommen ein paar Standard Include-Dateien des C-Compilers. Hier sind die Definitionen der Funktionen der Standard-Runtine enthalten.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
```

In diesem Bereich finden wir dir Include-Dateien der Domino C-API, für die Definition der Konstanten, Datentypen und Funktionen usw….

```
#include "global.h"
#include "nsfdb.h"
#include "nsfdata.h"
#include "osmisc.h"
#include "osfile.h"
#include "lapiplat.h"
```

Welche Include-Dateien werden gebraucht? Immer wenn man auf eine externe Definition zurückgreift, muss die entsprechende Include-Datei vorher eingebunden werden, welche steht in der Dokumentation. Für die Domino C-API gibt es dafür die folgende Datenbank:



So fangen alle C-Programme an….

```
int main(int argc, char *argv[])
{
```

argc gibt an, wieviele Parameter übergeben wurden und argv ist ein Array mit den Übergabeparametern. Das erste Argument (argv[0)] ist der Programmaufruf mit dem das Proramm gestartet wurde, inklusive Pfad, die folgenden Parameter sind restlichen Optionen (wie z.B. =c:\...\notes.ini)

Hier kommen die Variablendeklarationen, zum Teil mit Vorgabewerten.

```
char        path_name[MAXPATH] = "log.nsf"; /* buffer path to database */
DBHANDLE    db_handle = NULLHANDLE;         /* database handle */
char        buffer[NSF_INFO_SIZE] = "";     /* database info buffer */
char        title[NSF_INFO_SIZE] = "";      /* database title */
STATUS      error = NOERROR;                /* error code API calls */
```

MAXPATH, NULLHANDLE, NSF_INFO_SIZE und NOERROR sind Konstanten, DBHANDLE und STATUS sind Datentypen aus der C-API.

Die Doku sagt darüber z.B.

MAXPATH - Maximum pathname.

#include <names.h>

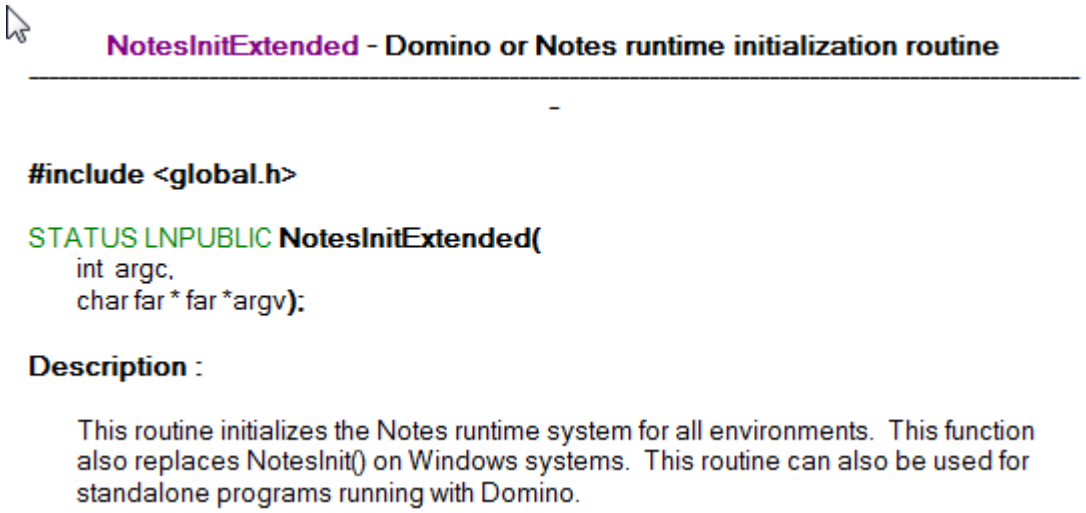
Description :

This length includes the terminating null so it may be used in data declarations. However, in places where a size is specified that indicates the maximum number of characters in the name, then the symbol minus 1 should be used.

Alle Strings werden in C mit einem Null-Byte terminiert. Wenn der Pathname also 256 Bytes groß sein darf, so sind nur 255 Buchstaben möglich. Dies ist eine beliebte Fehlerquelle!

error = **NotesInitExtended** (argc, argv);

NotesInitExtended initialisert das Notes-Laufzeitsystem. Es muss die NOTESINI und die ID-Datei finden, fragt das Kennwort ab.



NotesInitExtended - Domino or Notes runtime initialization routine

---

#include <global.h>

STATUS LNPUBLIC **NotesInitExtended(**
int argc,
char far * far *argv**):**

Description :

This routine initializes the Notes runtime system for all environments. This function also replaces NotesInit() on Windows systems. This routine can also be used for standalone programs running with Domino.

WICHTIG – Wenn NotesInitExtended ausgeführt wurde, muss am Ende UNBEDINGT NotesTerm() ausgeführt werden, weil sonst Speicherlecks entstehen und Notes oder Domino später abstürzen wird.

Hier wird geprüft, ob ein Fehler aufgetreten ist. NOERROR ist Null, in C bedeutet das false. Wenn ein Fehler zurückgegeben wurde, wird eine Statusmeldung ausgegeben und das Programm beendet.

```
if (error)
{
      fprintf (stderr, "\nError initializing Notes.\n");
      return (1);
}
```

Jetzt können wir die Datenbank öffnen – wieder mit Fehlerbehandlung ergänzt um NotesTerm().

```
if (error = NSFDbOpen ("log.nsf", &db_handle))
{
      fprintf (stderr, "\nError NSFDbOpen\n");
      NotesTerm();
      return (1);
}
```

NSFDbOpen - Opens an existing Domino database or database template or a directory.

```
#include <nsfdb.h>

STATUS LNPUBLIC NSFDbOpen(
    const char far *PathName,
    DBHANDLE far *rethDB);
```

Description :

This function takes a pathname to an existing Domino database or database template (whether on a Lotus Domino Server or a local database), opens the database, and returns a handle to it. All subsequent access to the database is carried out via this handle. Use NSFDbClose to close the database file handle and deallocate the memory associated with it.

Alle Datenbankaktionen erwarten einen sogenannte DBHANDLE, die Openfunktion setzt diesen Handle für uns. Die Variable haben wir oben deklariert. Damit die Funktion diesen Wert setzen kann, braucht sie einen Pointer auf diese Variable. Dazu dient in C das &-Zeichen.

Auch hier muss jedesml nach einem NSFDbOpen daran gedacht werden, die Datenbank mit NSFDbClose zu schliessen.

Mit diesem DBANDLE können wir nun die Datenbank-Informationen auslesen. Die Funktion schreibt sie in den oben deklarierten buffer.

```
if (error = NSFDbInfoGet (db_handle, buffer))
{
        fprintf (stderr, "\nError NSFDbInfoGet\n");
        NSFDbClose (db_handle);
        NotesTerm();
        return (1);
}
```

Exkurs:

Warum hat hier der buffer kein & davor? Buffer ist ein Array of char. Der Text "Hallo" entspricht z.B.:

```
buffer[0] = 'h'
buffer[1] = 'a'
      …
```

```
buffer[4] = 'o'
buffer[5] = '\0'   Null-Byte
```

Per Konvention ist buffer ein Pointer of Char (char *buffer), sodas buffer äquivalent zu &buffer[0] ist.

In den DB-Infos stecken viele Informationen über diese Datenbank. Um die einzelnen Elemente zu ermitteln, verwenden wir die Funktion NSFDBInfoParse.

```
NSFDbInfoParse (buffer, INFOPARSE_TITLE, title, NSF_INFO_SIZE - 1);
```

NSFDbInfoParse - Gets a specified piece of information from the database information buffer.

#include <nsfdb.h>

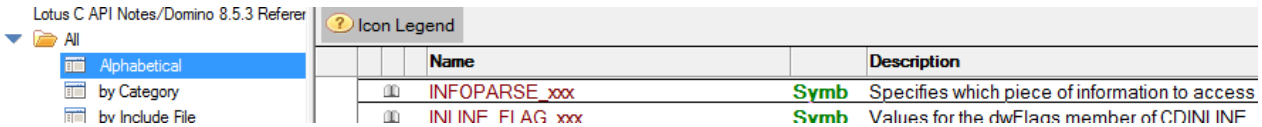
```
void LNPUBLIC NSFDbInfoParse(
    char far *Info,
    WORD What,
    char far *Buffer,
    WORD Length);
```

Description :

This function returns the specified piece of information from the given database information buffer. The database information buffer is obtained by a call to NSFDbInfoGet. This buffer contains the following pieces of information: database title, categories, class, and design class.

FRAGE: Welche Werte kann man mit dieser Funktion ermitteln?

TIPP: Schauen Sie hier:



Jetzt zeigt das Programm erstmal die Version der C-API:

```
/* Print out the Lotus C API for Domino and Notes Release Number */
printf ("\n\nLotus C API for %s Beispielprogramm EC2013",
        NOTESAPI_VERSION);
```

(Wo kommt den NOTESAPI_VERSION her? In der Doku haben sie es vergessen…..)

Um jetzt den oben ermittelten Titel der Datenbank auszugeben:

```
/* Print the title. */
printf ("\nDer Datenbanktitel für %s, lautet:\n\n%s\n\n", path_name, title);
```

Jetzt kommt das aufräumen, wir schließen die Datenbank.

```
/* Close the database. */
if (error = NSFDbClose (db_handle))
{
        fprintf (stderr, "\nError NSFDbClose.\n");
NotesTerm();
        return (1);
}
```

Und beenden die Laufzeitumgebung von Notes,

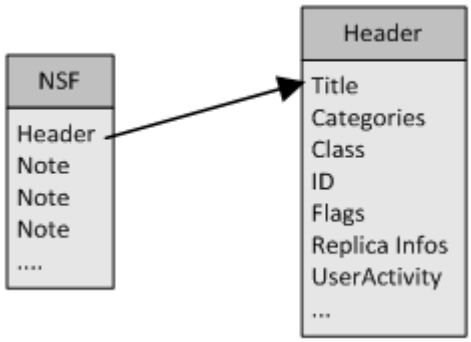
```
/* Terminate Domino and Notes. */
NotesTerm();

/* End of intro program. */
return (0);
```

AUFGABE 1: Verändern Sie das Programm so, dass die anderen Informationen ausgegeben werden.

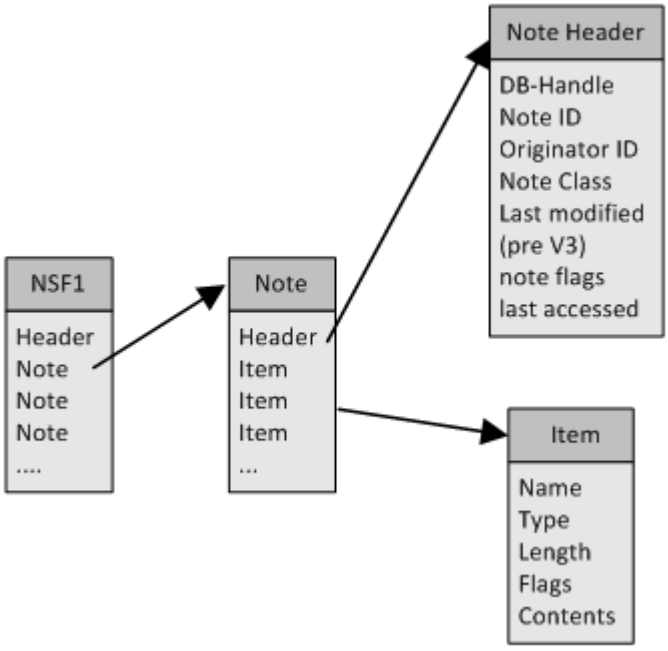
TIPP Die Funktionen um Datenbank-Eigenschaften zu manipulieren lauten fast alle NSFDb.....

Die Bestandteile der Datenbank:

Die Datenbank enthält einen Header, in dem verschiedene Informationen abgelegt werden.



Der Rest, z.B. Icon, Help-Dokumente, das Design (Views, Forms …) und die eigentlichen Dokumente werden in den Notes abgelegt.



AUFGABE 2: Verändern Sie den Datenbanktitel der Datenbank.
(NSFDbInfoModify und NSFDBInfoSet)

(Wenn es nur so einfach wäre, haben Sie die Bemerkung bei NSFDbInfoModify gesehen?)

Wir müssen also noch die NOTE mit dem Icon aktualisieren…..

NSFNoteOpen - Opens a note.

#include <nsfnote.h>

STATUS LNPUBLIC **NSFNoteOpen(**
DBHANDLE db_handle,
NOTEID note_id,
WORD open_flags,
NOTEHANDLE far *note_handle**);**

Description :

This function reads a note into memory and returns a handle to the in-memory copy. Its input is a database handle and a note ID within that database. This function only supports the set of 16-bit WORD options described in the entry OPEN_xxx; to use the extended 32-bit DWORD options, use the function NSFNoteOpenExt().

Jede mit nsfNoteOpen geöffnete Note muss mit nsfNoteClose wieder geschlossen warden

```
NOTEHANDLE note_handle;

nsfNoteOpen(db_handle, NOTE_ID_SPECIAL+NOTE_CLASS_ICON,0,
&note_handle);
```

Jetzt können wir testen, ob das Titelfeld da ist….

NSFItemIsPresent - Return TRUE if an Item is present in a note.

#include <nsfnote.h>

BOOL **NSFItemIsPresent(**
NOTEHANDLE note_handle,
char far *item_name,
WORD item_name_length**);**

Description :

This function takes a handle to an open note and the name for the Item whose presence you wish to confirm, and the length of that name. If the Item is present it returns TRUE.

If (NSFItemIsPresent(note_handle,FIELD_TITLE, (WORD) strlen(FIELDTITLE)))

Dann können wir den Wert ändern:

NSFItemSetText - Append an item of TYPE_TEXT to a note.

#include <nsfnote.h>

STATUS LNPUBLIC **NSFItemSetText(**
NOTEHANDLE hNote,
const char far *ItemName,
const char far *ItemText,
WORD TextLength**);**

Description :

This function takes a handle to an open note, the name of the item to append, and the text to store in the item. It appends a TYPE_TEXT item to the open note. If an item of that name already exists, it deletes the existing item first, then appends the new item.

NSFItemSetText(note_handle, FIELD_TITLE, „Neuer Titel“, MAXWORD);

Jetzt muss die Änderung noch gesichert werden (quasi das Save…)

NSFNoteUpdate - Write in-memory note to on-disk database.

```
#include <nsfnote.h>

STATUS LNPUBLIC NSFNoteUpdate(
    NOTEHANDLE note_handle,
    WORD update_flags);
```

Description :

This function writes the in-memory version of a note to its database. Prior to issuing this call, a new note (or changes to a note) are not a part of the on-disk database. This function only supports the set of 16-bit WORD options described in the entry UPDATE_xxx; to use the extended 32-bit DWORD options, use the function NSFNoteUpdateExtended().

NSFNoteUpdate(note_handle, 0);

Und nun schliessen wir die Note wieder..

NSFNoteClose - Closes an open note.

```
#include <nsfnote.h>

STATUS LNPUBLIC NSFNoteClose(
    NOTEHANDLE note_handle);
```

Description :

This function deallocates the memory associated with an open note. Closing a note does not write the contents of the note to disk. That function is performed by NSFNoteUpdate.

```
NSFNoteClose(note_handle);
```