Apple Events

Scripting mit dem MacOS

Nachdem ich beim letzten Mal das grundlegende Konzept, sowie die MacOS-Funktionen für Apple Events beschrieben habe, will ich nun ein funktionierendes Beispiel nachliefern und besprechen.

Um ein halbwegs sinnvolles, jedoch einfaches Beispiel zu haben, nehmen wir an, daß wir einen Text kodieren wollen. Wir haben also ein Hauptprogramm (Client), welches bei einem anderen Programm (Server) einen Text abliefert und den kodierten Text von diesem zurückerhält. Den Datenaustausch erledigen wir natürlich über Apple Events. Ferner soll der Server scriptable sein, d.h., auch fremde Programme sollen ihn über AppleScript problemlos nutzen können.

Das Client-Konzept

Neben den üblichen MacOS-Routinen zur Initialisierung sowie der Anmeldung einer Menüleiste brauchen wir eine Dialogbox, in der man einen Text eingeben kann, der zum Server geschickt wird, um ihn zu kodieren.

Der Server meldet einige zusätzliche Apple Events an, die das Empfangen eines Textes sowie das Kodieren und Abrufen ermöglichen. Er wird – zu Testzwecken – ebenfalls ein Fenster öffnen, welches den aktuellen Text darstellt.

Die Kodierung

Für die Kodierung wählen wir den in der DFÜ-Welt bekannten ROT13-Algorithmus. Er wird benutzt, um Texte zu verschleiern. Um eine echte Kodierung mit Paßwort handelt es sich hierbei nicht. Als Beispiel für einen möglichen sinnvollen Einsatz sei hier nur eine Gruppe mit Filmbesprechungen genannt, wo einige Leser über das Ende eines aktuellen Films sprechen wollen. Um nicht den zufälligen Lesern die Spannung an dem Film zu nehmen, kodieren sie ihre Textabschnitte mit ROT13. So liest niemand das Ende zufällig, sondern nur auf besonderen Wunsch! Die ROT13-Kodierung wird heute von den meisten Frontends unterstützt.

ROT13 hat seinen Namen aus der Art der Kodierung des Alphabets bekommen: zu den ASCII-Codes der einzelnen Buchstaben wird 13 hinzuaddiert. Bei einem Überlauf wird 26 abgezogen. Somit wird aus einem „A" ($41) ein „N" ($4E) und aus einem „N" wieder ein „A". Ziffern und Sonderzeichen werden nicht kodiert!

Hier ein Beispiel: Das Wort „Hallo" wird zu „Unyyb", und „Unyyb" wird beim zweiten Durchlauf wieder zu „Hallo".

Das Beispielprogramm

Um Platz zu sparen, habe ich den Sourcecode von Client und Server zusammengefaßt. Durch das Setzen der Variablen *COMPILE_SERVER* auf 0 erzeugen wir einen Client, durch Setzen auf 1 erzeugen wir einen Server. Das Listing selbst befindet sich aus Platzgründen auf der Mega-Disk zu dieser Ausgabe, ist aber auch in unserer Redaktionsmailbox erhältlich.

Gehen wir also die besonderen Routinen in diesem Programm durch:

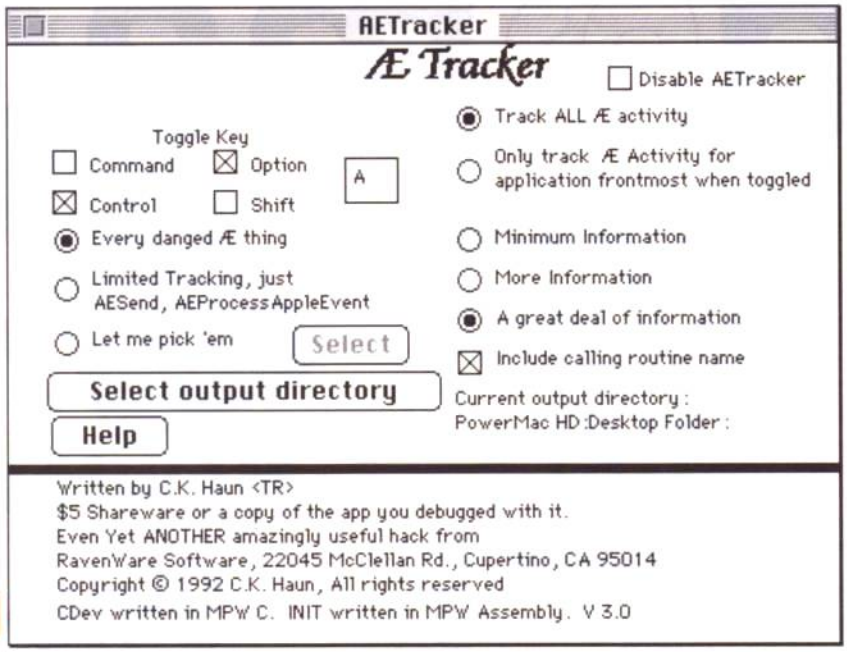
DoAppleEvent

Diese Routine erlaubt uns, das Versenden eines Apple Events an den Server. Der Server wird dabei anhand seines Creators gesucht. Dies setzt voraus, daß der Server auf dem gleichen Rechner bereits läuft. Falls er nicht läuft, so wird der Fehler *connectionInvalid* von der Routine zurückgegeben.

Ferner können wir diesem Event einen Textparameter mit übergeben. Und wenn der Event einen Reply zurückmeldet, so wird ein Text abgeholt. Dies reicht für unsere drei Events aus:

1. Text zum Server senden. Hier übergeben wir als Parameter unseren zu sendenden Text.
2. Text auf dem Server kodieren. Hier sind keine Parameter nötig.
3. Text vom Server abfragen. Hier bekommen wir vom Server den Text zurückgemeldet.

Natürlich könnte man die drei Events auch zu einem „Senden, Kodieren und Antwort zurücksenden"-Event zusam-



Der AE-Tracker ist nützlich, um Programme nach Apple Events zu „belauschen".

menfassen, aber wir wollen ja hier auch nur üben.

Achtung: Durch dieses Konzept der drei Events kann es passieren, daß uns ein zweiter Client in die Quere kommt! Wenn wir unseren Text gesendet haben, dann könnte ein anderer laufender Client ebenfalls einen (anderen) Text senden. Rufen wir nun später den Text wieder ab, erleben wir eine Überraschung ... Lösen könnte man dies durch Transaktionen. Die habe ich im ersten Teil ja bereits erwähnt.

AEOpenApp, AEQuitApp

Dies sind zwei der vier required Apple Events. Bei *AEOpenApp* tun wir gar nichts, bei *AEQuitApp* beenden wir unser Programm. Diese Events werden sowohl vom Server als auch vom Client verstanden.

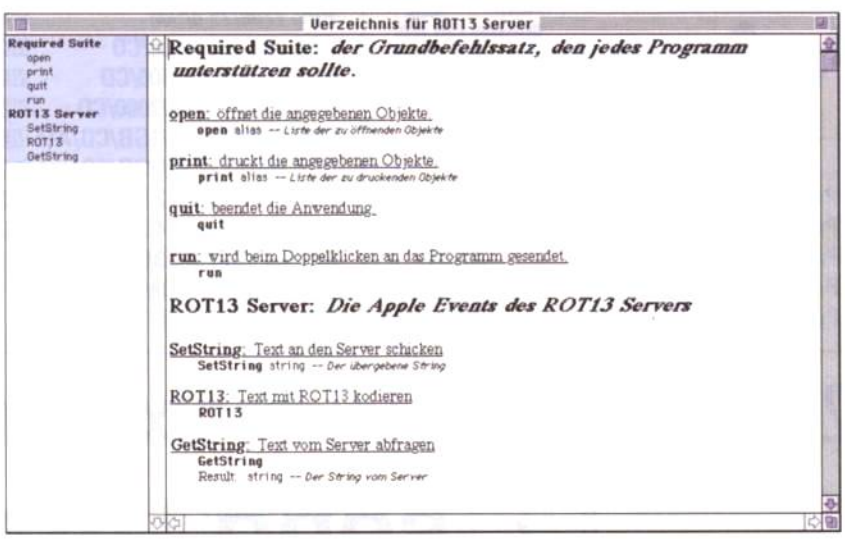
Die „Open Documents" und „Print Documents" unterstützen wir nicht, da wir gar keine Dokumente erzeugen können.

Der Server kennt zudem noch drei weitere Events:

AESetText
Dieser Event erwartet einen Textparameter und setzt ihn im Server.

AEGetText
Hiermit kann man den Text im Server abfragen.

AERot13
Diese Routine kodiert bzw. dekodiert einen Text im Server nach ROT13.



Diese Befehle versteht der ROT13 Server.

Der Rest ...

Viel mehr Highlights sind im Sourcecode nicht zu finden: ein ganz einfaches MacOS-Programm mit einer Dialogbox und etwas Menü. Ich hoffe, das Programm ist ansonsten soweit selbsterklärend.

Die Compilierung

Das Compilieren ist – wie immer – nicht ganz einfach. Am einfachsten sollte es wie folgt gehen:

Zuerst legt man zwei Ordner an: „ROT13 Client Projekt" und „ROT13 Server Projekt". In diese Ordner kopiert man den Sourcecode (je einmal in jeden Ordner). Im Source im Client-Ordner setzt man den *#define COMPILE_SERVER* auf 0. Im Server-Ordner muß er auf 1 stehen. Nun braucht man noch zwei Resourcen – für Client und Server. Diese gibt es – wie immer – auf der Mega-Disk. Zu guter Letzt muß man zwei Projektdateien anlegen und dort die „MacOS.lib" sowie den „ROT13.c" und die Resource-Datei hinzufügen. Beim Server muß man den Filecreator unbedingt auf „Rs13" setzen, damit er vom Client gefunden werden kann!

Scriptable!

Nun ist es nur noch ein winziger Schritt, um den Server auch mit AppleScript ansprechen zu können! Dazu müssen wir eine *aete*-Resource (mit der ID 0) für den Server erzeugen. Für das Erzeugen der *aete*-Resource gibt es einige Möglichkeiten – gute und schlechte:

Das Erzeugen mit dem kommerziellen Resource-Editor „Resorcerer"
Dies ist die beste Möglichkeit, *aete*-Resourcen zu erzeugen. Leider ist der Editor vergleichsweise teuer. Dafür ist er dem ResEdit in nahezu allen Belangen um Lichtjahre überlegen!

Das Schreiben mit dem „Aete Editor Stack 1.0b3"
Hierbei handelt es sich um einen HyperCard-Stack, der nicht nur in der Lage ist, *aete*-Resourcen zu erzeugen, sondern der sogar fertige Programme (Client und Server!) in Pascal und C erzeugen kann. Leider sind die erzeug-

Unser Beispielscript, um den String „Hallo" zu kodieren.

ten Sourcecodes nicht fehlerfrei ... trotzdem kann man einen sehr schönen Überblick gewinnen, und die paar Fehler kann man normalerweise schnell selbst beheben.

Das Schreiben mit dem Rez-Compiler

Hier schreibt man die *aete*-Resource mit einem Text-Editor und kompiliert sie anschließend mit dem Rez-Compiler zu einer Resource-Datei. Für das Listing müssen wir die *aete*-Resource übrigens als Rez-Listing mitliefern.

Der ResEdit mit *aete*-Editor-Plug-In

Nun ja, das Teil wird in Zukunft vielleicht mal interessant werden. Bis jetzt sieht es nicht sonderlich brauchbar aus.

Mit dem Skript-Editor können wir uns unsere Resource im Klartext anzeigen lassen. Die sieht beim Server z.B. so aus:

```
ROT13 Server: Die Apple Events des ROT13-Servers

SetString: Text an den Server schicken
    SetString string -- Der übergebene String

ROT13: Text mit ROT13 kodieren
    ROT13

GetString: Text vom Server abfragen
    GetString
    Result: string -- Der String vom Server
```

Ein kleines Beispielprogramm zeigt, daß der Server auch von AppleScript aus funktioniert. Am Ende sollte der Skript-Editor ein Fenster geöffnet haben, in dem er den kodierten String von „Hallo" darstellt. Im Fenster des Servers kann man kontrollieren, ob alles geklappt hat.

```
tell application "ROT13 Server"
    SetString "Hallo"
    ROT13
    GetString
end tell
```

Debugging

Das Debugging von über Apple Events miteinander kommunizierenden Programmen ist leider nicht ganz einfach. Deswegen noch drei hilfreiche Programme:

AETracker 3.0

Funktioniert ähnlich dem SysMon: dieses Kontrollfeld kann die Kommunikation zwischen Programmen abhören und in einer Datei protokollieren. Hiermit sollte man die meisten Fehler dann recht problemlos finden können.

Die 5 US$ Shareware-Gebühr ist das Kontrollfeld mehr als wert!

SendAE Test Tool 0.3

... ermöglicht es, einem Apple Events recht einfach zusammenzustellen und an ein Server-Programm zu schicken. Falls man sich nicht sicher ist, ob die Abfrage oder der Empfang klappt, kann man hiermit den Server gut testen.

MacsBug

Der bekannte Low-Level-Debugger kann mit entsprechenden Erweiterungen auch Apple Events etc. darstellen.

Ausblick

Der Einstieg ist geschafft! Wer noch mehr über Apple Events wissen will, sollte sich neben dem bereits genannten NIM „Interapplication Communication" von Apple noch das „Ultimate Mac Programming"-Buch von Dave Mark besorgen. Dieses knapp 600 Seiten dicke Buch ist ein hervorragender Einstieg in die Materie und bietet zusätzlich noch einige weitere interessante Themen (Sound, Trap-Patching u.v.a.m.) – zudem liegt eine CD-ROM bei, welche neben allen Listings auch noch eine benutzbare Demoversion vom CodeWarrior enthält sowie Demoversionen von vielen anderen Programmen zum Thema Software-Entwicklung.

MFR

Literatur:

NIM „Interapplication Communication" von Apple Computer, Inc., Addison-Wesley Publishing Company ISBN 0-201-62200-9 $36.95

„Ultimate Mac Programming " von Dave Mark, IDG Books ISBN 1-56884-195-7 $39.95