Chapter 1 : HyperCard - Wikipedia

*The Educator's Guide to Using Hypercard and Hypertalk/Book and Disk [George H. Culp, G. Morgan Watkins] on calendrierdelascience.com *FREE* shipping on qualifying offers.*

A handler to export text from the background fields of a HyperCard stock to a text file. For the sake of convenience, we refer to them collectively as XCMDs. An XCMD is compiled code. Writing XCMDs is a task for programmers, not users. They are invoked by HyperTalk through its inheritance mechanism. If the appropriate resource name is found, the resource is loaded into memory and HyperCard jumps to the start of it. There are several reasons for using an XCMD: It is easy to control a video disk player through the built-in RS ports of the Mac. The video disk player controller would require three pieces of code. The driver for the video disk player would be a standard Macintosh driver. Many people have become proficient at writing drivers because a desk accessory is a driver. The second piece of code would be the XCMD. The XCMD would be small, and its purpose would be to simply convert HyperTalk messages to the appropriate driver calls. The third piece of code would be the HyperTalk scripts that call the XCMD with various parameters, asking it to ask the driver to ask the laser disk player to go to a certain frame, or to backup, for example. Using XCMDs as an interface to traditional drivers is a simple but powerful operation. If an XCMD gets too big in size, think about writing a driver. Keep in mind that XCMDs do have limitations. XCMDs are more like desk accessories than an application in this regard. Guidelines for writings XCMDs are as follows: Do not initialize the various Macintosh managers by calling their init traps: Do not rely upon having lots of RAM available for your code. Do not use register A5 of the 68xxx processor. Use STR resources or put the strings in a short assembly glue file. No A5 World means no jump table. No jump table means no code segments. No code segments means a 32K limit on code size for based machines. The supports longer branches. If your code allocated some memory in the heap, your code should also deallocate the memory. If an XCMD allocates a handle to save state information between invocations of the XCMD, then you must also use HyperTalk to store the handle somewhere in the current stack, perhaps in a hidden field. You will need to convert the handle from a LongInt to a string, as everything is treated as a string on the HyperTalk end of things. The link order is important. HyperTalk provides both conversion routines as well as stackaccess routines. The conversion routines provide support for converting various styles of strings and numbers into each other. For example, there are routines to convert C-style strings zero terminated back and forth to Pascal-style strings length byte followed by string. There are also routines to convert strings to and from integers. In addition, values of variables can be recalled and stored into, and contents of fields can be examined. The bulk of the code in the glue files is there to support these callback routines. However, because it is interpreted, it is also sometimes slow, especially when doing anything repetitive. For this reason, it is often wise to rewrite some operations as XCMDs, to benefit from the speed of compiled native 68xxx code. Documentation HyperCard ships with an excellent introductory manual that, unfortunately, does not include a description of HyperTalk. However, several good sources of information about using HyperTalk are available. The Help stack that ships with HyperCard contains a great deal of information about the language, and is an excellent resource. A good tutorial with pretty good reference material for HyperTalk has become an overnight phenomenon: This book, published by Bantam, is currently a bestseller. There is a rash of other books on the market, but most of them are highly inaccurate. For more information, contact: All software is shipped on K HFS disks. The disk includes the header and glue files shown in Listings Three page 68 and Four page Originally packaged with every Macintosh sold, the original versions of MacPaint 1. The latest version of MacPaint 2. QuickFile is a small publicdomain file manager that Bill Atkinson wrote in It uses a fast in-memory text search with a fixed-sized Rolodex card. Its search routine is completely different from that used in HyperCard. QuickFile was originally called Rolodex, but the name was changed due to copyright problems. Bill Atkinson originally wrote QuickDraw for the Lisa computer. QuickDraw was first written in Pascal, then rewritten many times, ending up finally as 24K of assembly code. Bill further enhanced QuickDraw for the Mac Plus by unwinding some of the loops, thus increasing the speed. Actually the inheritance path is slightly more complicated when the currently executing script goes to a different card or stack during its execution. The current script card or stack is searched as well as the current visible card or stack. The search order for XCMDs is: Desk accessories were originally meant to be a maximum of 8K. Today there are desk accessories over K!

Chapter 2 : A HyperCard Manual & Floppy Disks (Floppy Disk)

Like the Web, it also allowed for the connections of many different kinds of media. Each card contains a set of interactive objects, including text fields, check boxes, buttons, and similar common graphical user interface GUI elements. Users "browse" the stack by navigating from card to card, using built-in navigation features, a powerful search mechanism, or through user-created scripts. They place GUI objects on the cards using an interactive layout engine based on a simple drag-and-drop interface. This way, a stack of cards with a common layout and functionality can be created. The layout engine is similar in concept to a "form" as used in most rapid application development RAD environments such as Borland Delphi , and Microsoft Visual Basic and Visual Studio. The database features of the HyperCard system are based on the storage of the state of all of the objects on the cards in the physical file representing the stack. The database did not exist as a separate system within the HyperCard stack; no database engine or similar construct exists. Instead, the state of any object in the system was considered to be live and editable at any time. The system operates in a largely stateless fashion, with no need to save during operation. This is in common with many database-oriented systems, although somewhat different from document-based applications. The final key element in HyperCard was the script, a single code-carrying element of every object within the stack. The script was a text field which contents were interpreted in the HyperTalk language detailed below. When the user invokes actions in the GUI, like clicking on a button or typing into a field, these actions are translated into events by the HyperCard runtime. The runtime then examines the script of the object that was the target of the event, like a button, to see if its script object contains code for that event, code termed a handler. If it does, the HyperTalk engine runs the handler, if it does not, the runtime examines other objects in the visual hierarchy. External video "HyperCard Mania! Unlike the majority of RAD or database systems of the era, however, HyperCard combined all of these features, both user-facing and developer-facing, in a single application. This allowed rapid turnaround and immediate prototyping, allowing users to author custom solutions to problems with their own personalized interface. It was this combination of features that also made HyperCard a powerful hypermedia system. Users could build backgrounds to suit the needs of some system, say a rolodex , and use simple HyperTalk commands to provide buttons to move from place to place within the stack, or provide the same navigation system within the data elements of the UI, like text fields. Using these features, it is easy to build linked systems similar to hypertext links on the Web. Objects exist in a message path hierarchy and respond to messages generated by either the user or the system timers for instance. Objects inherit properties and attributes from those above them in the hierarchy. HyperTalk object classes are predetermined by the HyperCard environment, although others can be added by the use of externals see below. HyperTalk is verbose, hence its ease of use and readability. HyperTalk code segments are referred to as "scripts", a term that was considered less daunting to beginning programmers. Each HyperCard object class, contains a set of "properties". For example, buttons are a type of object, and come in standard styles. To determine, say, whether a checkbox style button is in fact checked, a script can simply call the highlight property, [10] which would return either true or false. In a similar way, objects can be analyzed via functions. This is very useful when performing a given action on each separate line of the field. The script that implements the action need only call the function to know exactly the number of lines it must process. Should the field data change, the already coded function call will still be accurate. HyperTalk is a weakly typed language. All variables, and in fact all values of any kind, are stored as typeless character strings handled by the interpreter as numbers or text based purely on context. This has a cost in speed. Variables need not be declared, but rather are created on the fly as they are required. For example, the following expression creates a variable named total, and sets its initial value: Then the expression add 3 to total would result in the string "18" being stored in that variable. Taking this further, a powerful and intuitive structure known as "chunking" allows precise manipulation of text and number strings. It is possible, for example, to have the second character of the value "" the 2 added to the last character of the value "", yielding "". For another example, word 3 of "life is cruel" cruel can be appended after the first word of "Hello world", yielding "Hello cruel world". It would then be possible to put "Goodbye" into the first word of that string, replacing the current value of that word to yield "Goodbye cruel world". The above mentioned terms: HyperTalk supports most standard programming structures such as "if-then" and "repeat". The "if-then" structure is so flexible that it even allows "case" structured code. HyperTalk scripting allows the system to be easily modified and extended. Unlike many procedural languages, and even many scripting languages, HyperTalk proved to be far more accessible to a wide range of users, partly because scripts were more or less readable as English. For instance, put the first word of the third line of field "hello" into field "goodbye" did exactly that. Referring to objects and the items on cards or backgrounds is easy. The example above shows how to access data within a field on a given card, but one can refer to any object in the same fashion, including the stack. All objects can be named or renamed, as in the example above. Adding scripts is also easy. The script may then be edited, saved, and used immediately. Also, HyperCard contains a Message Box, an interactive command-line in a floating window that can execute single lines of script. This also includes the find command, so it doubles as a search dialog. HyperTalk was sufficiently popular that one of its main uses was not as a database, but as a programming tool that empowered ordinary computer users. Thousands of stacks were written and distributed as stackware in the years when HyperCard was widely available. As stated above, programming "for the rest of us", that is, for non-professionals, allowed many thousands of personal applications to be created by individuals with a need for personal software solutions. Some are still in use today. Many hardware and software vendors provided their tutorials as HyperCard stacks, since the application was bundled with all Macs. These were code libraries packaged in a resource fork that integrated into either the system generally or the HyperTalk language specifically; this was an early example of the plug-in concept. Unlike conventional plug-ins, these did not require separate installation before they were available for use; they could be included in a stack, where they were directly available to scripts in that stack. BeeHive Technologies offered a hardware interface that allowed the computer to control external devices. Connected via the Apple Desktop Bus ADB , this instrument could read the state of connected external switches or write digital outputs to a multitude of devices. Externals allow access to the Macintosh Toolbox, which contained many lower level commands and functions not native to HyperTalk, such as control of the serial and ADB ports. It was initially released in August , with the understanding that Atkinson would give HyperCard to Apple only if the company promised to release it for free on all Macs. HyperCard was a huge hit almost instantly. Many people who thought they would never be able to program a computer started using HyperCard for many automation and prototyping tasks, a surprise even to its creator. Project managers found it was being used by a huge number of people, internally and externally. Bug reports and upgrade suggestions continued to flow in, demonstrating it had a wide variety of users. Since it was also free, it was difficult to justify dedicating engineering resources to improvements in the software. It was not lost on Apple or its mainstream developers that the power HyperCard gave to people could cut into the sales of ordinary shrink-wrapped products. This resulted in HyperCard 2. The new version included an on-the-fly compiler that greatly increased performance of computationally intensive code, a new debugger and many improvements to the underlying HyperTalk language. At the same time HyperCard 2. Although stacks HyperCard program documents were not binary-compatible, a translator program another HyperCard stack allowed stacks to be moved from one platform to the other. Then, Apple decided that most of its application software packages, including HyperCard, would be the property of a wholly owned subsidiary called Claris. Many of the HyperCard developers chose to stay at Apple rather than move to Claris, causing the development team to be split. Claris, in the business of selling software for a profit, attempted to create a business model where HyperCard could also generate revenues. At first the freely-distributed versions of

DOWNLOAD PDF THE EDUCATORS GUIDE TO USING HYPERCARD AND HYPERTALK/BOOK AND DISK

HyperCard shipped with authoring disabled. Many users were upset that they had to pay to use software that had traditionally been supplied free and which many considered a basic part of the Mac. Despite the new revenue stream, Claris did little to market HyperCard. Development continued with minor upgrades, and the first failed attempt to create a third generation of HyperCard. During this period, HyperCard began losing market share. Without several important, basic features, HyperCard authors began moving to systems such as SuperCard and Macromedia Authorware. Nonetheless, HyperCard continued to be popular and used for a widening range of applications, from the game The Manhole , an earlier effort by the creators of Myst , to corporate information services, and many thousands in between. In , Apple released the eagerly anticipated upgrade of HyperCard 2. However, these tools were limited and often cumbersome to use because HyperCard still lacked true, internal color support. The resulting HyperCard 3. Development of HyperCard 3. Calhoun and Crow both left Apple shortly after, in In the years that followed, the program saw no more support from Apple, which finally ceased selling HyperCard in March Applications[edit] HyperCard has been used for a range of hypertext and artistic purposes. Before the advent of PowerPoint , HyperCard was often used as a general-purpose presentation program. Examples of HyperCard applications include simple databases, " choose your own adventure "â€”type games, and educational teaching aids. Due to its rapid application design facilities, HyperCard was also often used for prototyping applications and sometimes even for version 1. HyperCard has lower hardware requirements than Macromedia Director.

Chapter 3 : An Introduction to Hypercard Programming | Dr Dobb's

Educators Guide To Using Hypercard And Hypertalk Book And Disk More references related to educators guide to using hypercard and hypertalk book and disk.

Chapter 4 : Results for George-Culp | Book Depository

This revised edition focuses on the newest version of HyperCard (). It explores all the new features, including the use of color. Developed specifically for educators, this well-illustrated, easy-to-follow, practical guide emphasizes a hands-on approach that stresses learning by doing.

Chapter 5 : George H. Culp | Open Library

Note: Citations are based on reference standards. However, formatting rules can vary widely between applications and fields of interest or study. The specific requirements or preferences of your reviewing publisher, classroom teacher, institution or organization should be applied.