

# Memory File System

## *Features supported*

Thu, Jan 6, 2000

The IRM system code includes several areas of support for memory-based file system. This note describes most of these areas.

A file directory is maintained as a table of 32-byte records. This CODES table is system table #9. The contents of a CODES table entry include the file name, which is composed of two 4-character names, the first the type name and the second the file name. (There is some ambiguity here for the term "file name.") Values for type name are limited in practice. The type name PAGE is used for page application program files; the type name LOOP is used for local application program files; the type name DATA is used for arbitrary data files; and the type name HELP is used for the single file called HELPLOOP that supplies prompting parameter text for installing local applications instances. No other TYPE names have yet been used. The 4-character file name may be anything, but it has been conventionally chosen as a full 4-character upper-case alphanumeric name. Any name less than 4 characters in length should be blank-filled to a total of 4 characters.

Other fields in the CODES table entry include the size of the file in bytes, a checksum for the file contents, a pointer to its beginning address in non-volatile memory, a pointer to its beginning address in dynamic memory, if one is allocated. Finally, a version date is kept in a 6-byte BCD format of YrMoDaHrMnSc. (Only two-digit years are recorded, so that 00 represents the year 2000.)

A CODES table entry is installed whenever a file is written into the non-volatile memory. There is support via listype #76 for writing a file into this memory, which should be done all at the same time. (The file system is a read-only file system; a file's contents may be changed only by writing a new version of the file with the same 8-character name.) The ident for listype #76 consists of the node#, the 4-character type name, the 4-character file name, and a 4-byte offset. The procedure to write a new file consists of three parts. First, send the 4-byte total size of the file using the ident with the indicated type name and file name and 4-byte offset value -1. This causes any previous version of the same file to be freed and the indicated space reserved for the new file that is to follow. (If the size that is sent is 0, then the current version of the file is deleted, but no new file is allocated. This can be useful when nonvolatile memory is becoming full, as all free space segments are combined into one free segment at reset time.) Then send the file contents in pieces, using the offset field to indicate the base to target the data being sent. The final step is to send the 4-byte checksum value and optionally the time stamp, using the 4-byte offset value of +1. If the number of bytes of data sent is 4, it is the checksum, and a time stamp is automatically assigned by sampling the present time as known in the receiving node. If the number of bytes is 10, then a 6-byte time stamp follows the checksum. The 6-byte BCD time stamp is of the format YrMoDaHrMnSc. (It is accurate enough to time stamp the new file to one-second precision.) Note that the checksum is sent from the client system, not automatically generated according to what data segments arrive. The time stamp will be used to check against what is actually in the nonvolatile memory file, when it is copied into dynamic memory to be used, say, for execution. Because of this step—not actually using a file except as a copy from non-volatile memory—one can freely update the nonvolatile file without disturbing ongoing system operation.

The time stamp associated with a given file can have the significance of a version of the file contents, which is usually an executable program. The TFTP server supports the downloading of such files from a TFTP client. Since there is no way for a client to specify the time stamp of the file that is sent, the TFTP server performs a setting for the last step of only a checksum value, so that the local node assigns a time stamp. After transferring a new file to an initial node via TFTP, any subsequent copies of that file to other nodes use Classic Protocol to perform the transfer, which preserves the time stamp as recorded in the initial node. It is common to maintain a library node that houses the latest versions of all program files in this way.

The Page D application program can generate a listing of the contents of the CODES table, which

includes all the information in the CODES table entries, including a diagnostic count of the number of times the program version has been called up from nonvolatile memory since it was last downloaded.

The CODES table directory is also referenced by the Page E application, which permits installation of local application (LOOP-type) instances. It can thereby show the version time stamp of the program whose instance is currently displayed.

While not a feature, a special page program was written to look for problems in the nonvolatile memory region compared to what the CODES table indicates. It is vital that they correspond, as the CODES table is actually the memory file system directory. One might compare this to the Disk First Aid application on the Macintosh.

The overall objective fulfilled by the memory file system is one of housing programs or data that survive resets or power outages. When placed into service, each local application program becomes an extended part of the system code. The configuration of local applications is also maintained in nonvolatile memory so that every time the system is initialized, the same suite of local applications is placed into service automatically. Also, while the system is running, new versions can be downloaded into the nonvolatile area, so that the next time the program is called for, the new version is available. For a local application that is active, if a new version is downloaded, the old running version is terminated normally, and the new version is placed into execution automatically and initialized with the same parameters.

Local applications are merely procedures that are called every cycle, with up to ten 16-bit parameters passed to them each time they are invoked. This happens during data access table processing, triggered by one entry type that causes all enabled local application instances to be invoked. Each is expected to run only a limited time, as all must be invoked each cycle. In addition, a local application that performs the function of a network protocol server—either UDP port or Acnet task—can be invoked when a network message is received that is directed toward that server. Serial port input can also cause invocation of a local application that so declares its interest to the system.

Page applications are similar to local applications, but they support a user interface page display of 16 lines by 32 characters. Only one may be active at a time, and it is invoked once each cycle after all data access table work, routine fulfillment of data requests, and alarm scanning has completed. The actual user interface is supported either by a small console hardware box directly connected to the front end node, or by a "Page G" page display interface that emulates such a console on another platform, such as a Macintosh or PC or a Unix host. If multiple users attempt to operate the page display interface simultaneously, the actions of each is shared; what one user types onto the screen can be read by another user, or one can simply observe the actions of another all the while in phone communication.

Diagnostics are available to monitor the time spent executing each local application. This can be observed via Page E, updated each cycle. The total time spent on each cycle executing all active local applications during data access table processing is available internally in that data access table entry itself, where not only the time spent each cycle is shown, but also the maximum time ever spent in one cycle is given. The starting time for commencement of local application invocations each cycle is also available.

The diagnostic page application that supports dumping out some of the context memory from all instances of a given local application in a set of nodes makes use of local application access. When each local application instance is initialized, a block of context memory is allocated that serves to maintain that local application instance's context. A pointer to this memory, in addition to the ten parameters, is passed to the local application each time it is invoked. This is what makes it possible to support multiple instances of a local application.