MFS Applications

An MFS is a likely solution anytime a developer wishes to monitor or control all activity in a file. Because the MFS sits on top of the filing system itself, it can oversee all operations in the file, regardless of what means a user might find to access the file (except a direct call to the BFS -- see "Programming an MFS"). Excellent candidates for MFS implementation include the following functions:

Application	Description
Encryption and Compression	An MFS can be used to monitor all reads and writes to a file, and pass the record through appropriate encryption or compression routines before the data is filed away. When the data is read later, the MFS can call decryption or decompression routines to return the data to its original form. At the BASIC+ level, all file I/O appears normal.
Audit Trails and Indexing	Because an MFS can detect when there is an attempt to read, write or delete records in a file, it can maintain an audit trail of all such activity, or can update indexes based on changes to the record. An audit trail or index MFS is an example of an MFS that does not actually modify the data for the filing system, but simply tracks its usage.
Security (Record and Field)	An MFS can use system security information (user names and privilege levels) to grant or deny access to data in the file. For example, the MFS can monitor all reads to a file and deny access to a user who does not have sufficient access privileges. The MFS might also permit some users to write to the file, but not others. This security can be implemented for the record as a whole, or for individual fields. A possible implementation for the latter might null out fields during a read that are not authorized to a particular user, and replace them before writing the data back to file.

System MFSs in OpenInsight

MFS	Description
SI.MFS	All files that contain indexed fields are monitored by **SI.MFS**. If this MFS detects that a change has been made to an indexed field, it creates a transaction that is later used to update the appropriate index. **SI.MFS** also traps certain retrieval calls such as READNEXT and fulfills them from an index, rather than directly from the file.
QUICKDEX.MFS and RIGHTDEX.MFS	If installed, **QUICKDEX.MFS** and **RIGHTDEX.MFS** monitor all writes, selects, and deletes against a file. During a write, these MFSs update a hidden record in the file, maintaining a sorted list of keys for records in the file. During a SELECT and READNEXT, **QUICKDEX.MFS** and **RIGHTDEX.MFS** simply read this hidden record, providing almost instantly a sorted list of record keys. At the same time, the MFSs hide the record by removing its name from any select lists generated by the user.
DICT.MFS	Two major functions are rolled into **DICT.MFS**. First, the MFS monitors writes to any dictionary (any file with a name beginning with the characters "**DICT**."), and calls the dictionary compiler whenever an **F** or **S** type record is written to the file. Second, **DICT.MFS** examines all records being written to look for indexing flags. If any are found (for example, if the sixth field is set, a Btree index has been established for that field), **DICT.MFS** calls additional system routines used to create an index for that field. **DICT.MFS** differs from other MFSs in OpenInsight in that it is installed "on the fly." The mere presence of the trigger characters "DICT." at the front of a file name causes the ATTACH process to install **DICT.MFS** onto the file automatically. Other MFSs must be installed explicitly.