Tie Fighter, Part 1

by Wayne Sikes

Peer into the depths of the TIE Fighter game engine from LucasArts Entertainment and discover a world of complex data files, file storing, and file-naming conventions.

On the chopping block this month is TIE Fighter by LucasArts Entertainment Company. TIE Fighter is the second game in LucasArts' Star Wars series, X-Wing is the first. The TIE Fighter game engine is an improved version of the X-Wing engine. The most noticeable improvements were made in the graphics rendering, artificial intelligence, and cockpit data display areas.

The TIE Fighter engine and its associated data structures are somewhat complex; therefore, this review will span two editions of the Chopping Block. In this issue, I will broadly summarize the executable and data files and focus on the pilot data file. In the next issue, I will delve into mission construction. I used TIE Fighter version V1.0 (06/15/94) for this review.

TIE Fighter initially requires about 14MB of hard disk space. A minimum of 572k conventional RAM and 900k of expanded memory are required to run the game. Ideally, the game wants 2MB of expanded memory. During installation the \CP, \MISSION, and \RESOURCE subdirectories are created under the primary \TIE directory. The \CP subdirectory contains most of the data files for vehicles the player is allowed to fly. The \MISSION directory is loaded with battle and historic mission data, and \RESOURCE contains the "generic" pieces of the game, including battle summary information; music, sound effects, and speech data; menu screen graphics; the game logo and credits; and registration (copy protection) data. The primary game subdirec-

A defender training mission in TIE Fighter.

Listing 1. Data Storage Files

```
FILE NAME    DATA TYPE
SUFFIX

TIE          All Historical and Battle mission files.

LFD          The most common data storage format for various types of data.

PNL          Appears to contain raw graphics data.

INT          Contains data for vehicles that the player can fly.
             Primarily tabular in format.

TFR          Pilot data file.
```

Listing 2. LFD Storage File Structures

```
struct  LFDRECORDHEADER
   {
   char   RecordName[12];     // Record name string
   long   RecordSize;         // Size of Record data
   };

struct LFDRESOURCETAG
   {
   char   ResourceTag[12];    // "RMAPresource" string
   long   FirstRecordOffset;  // Offset of the first Record
   };
```

Listing 3. Spacecraft Name References

SHIP NAME	LETTER REFERENCE	NUMERICAL REFERENCE
TIE Fighter (T/F)	F	1
TIE Interceptor (T/I)	I	2
TIE Bomber (T/B)	B	3
TIE Advanced (T/A)	A	4
Assault Gunboat (GUN)	G	5
TIE Defender (T/D) (also known as the TIE Deluxe)	D	6

tory, \TIE, contains the game executable files, user configuration information, system setup help routines, display fonts, and various sound card drivers.

Tie Fighter Executables

TIE Fighter was written using the Microsoft C development system. The executable code consists of three files: FLIGHT.OVL, FRONT. OVL, and TIE.EXE. The game is started by running TIE.EXE, which subsequently loads the FRONT.OVL and FLIGHT. OVL overlay routines when required. The various game functions are logically organized into these three files.

The TIE.EXE routine performs game start up. An initial part of the start up includes memory allocation and data validity checks. Once memory has been verified, the iMUSE (LucasArts' proprietary Interactive Music and Sound System) engine is started. Another initialization function involves setting up the flight combat filming system. While examining the TIE.EXE code, I noticed the unusual character string "heidirobinyali" buried in the data. Purusal of the *Starfighter Pilot Manual's* list of game credits showed that this string was the first names of three of the "Invaluable Support" personnel. It is possible that this data string was used by programmers as a trigger for debug or developmental mode operation. Although TIE.EXE is 368365 bytes in length, only about 59943 bytes are used for code and data. The remaining 308422 bytes have a 0 value and are apparently used when the Microsoft C run-time system loads the FRONT.OVL or FLIGHT.OVL overlay routines.

The FRONT.OVL module provides the front end for the game and provides the overall, nonflight gaming environment. This environment includes most of the main menu Concourse functions, cut scenes and other battle-related animations, award functions, funeral scenes, Tech Room and Film Room operations, and the registration and copy protection modules. To avoid violating LucasArts' copyrights, I will not tell you how to override the game's copy protection. I can tell you that the copy protection consists of standard C, null-terminated character strings that begin at file offset 3E42A (hex) and the last string ends at offset 3E5F1 (hex).

The FLIGHT.OVL overlay module contains most of the flight data. This file appears to contain the artificial intelligence used during combat and other space flight sequences. There are a number of possibilities for most combat situations. For example, a cursory scan of the "intelligence" given to Rebel vehicles found references to over 70 presumed different actions or activities. Data structures that define the general layout (weapons, shields, hull strengths, etc.) of all game vehicles are located in FLIGHT.OVL.

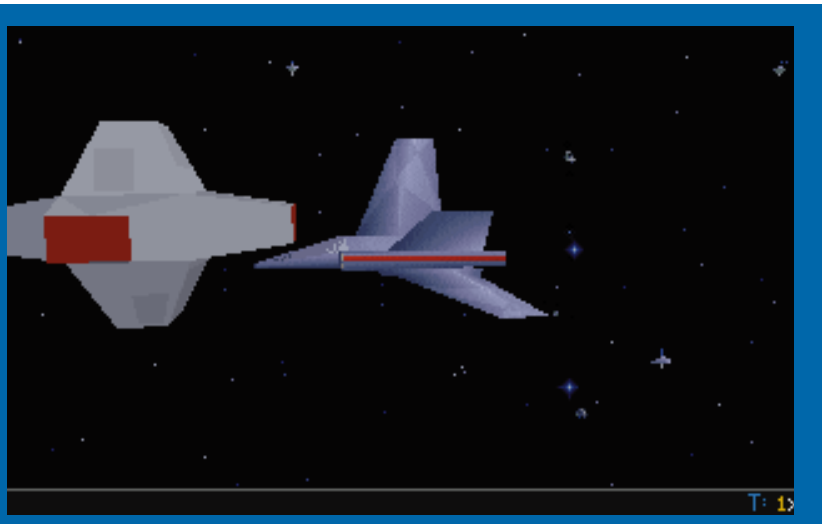
How Is the Data Stored?

TIE Fighter data is scattered among numerous files, and I observed several storage formats. Listing 1 gives a summary of several data storage file types. Most mission data is stored in .TIE files and the bulk of the general data is stored in .LFD files. (I speculate that the LFD suffix is a mnemonic for "Lucas File Data," and I will refer to the files suffixed with LFD as "LFD files.") Pilot data is stored in files suffixed with TFR.

The LFD file format allows for the storage of single or multiple Records. I observed two flavors of LFD files, but the Record data is stored basically in the same manner between the two. I will refer to these two types of files as Type I and Type II. The difference between the two LFD file types is that Type II has a tabular header at the top of the file, which indexes all of the Records in the file.

In both LFD file types, each Record consists of a 16-byte header followed by the Record data. The 16-byte header is composed of a 12-byte Record name followed by the size of the Record data expressed as a 4-byte (32-bit long) value. The Listing 2 structure, `LFDRECORDHEADER`, gives an example Record header. The actual size of the entire Record would be calculated by adding 10 (hex) to the `RecordSize` value.

The Type II tabular header consists of a 16-byte "Resource Tag" followed by 16-byte headers for all Records in the file. These 16-byte headers are duplicates of the Record headers previously discussed. The individual Records immediately follow the tabular header. The Resource Tag consists of a 12-byte character string, "`RMAPresource`", followed by the offset of the first Record expressed as a 4-byte value. The `LFDRESOURCETAG` structure in Listing 2 gives an example Resource Tag. (The actual offset of the first Record following the tabular header is calculated by adding 10 (hex) to the `FirstRecordOffset` value.)



An assault gunboat flies by a buoy, one of several space objects in TIE Fighter.

Mission File Naming Conventions

As previously mentioned, the mission files have a TIE suffix. Examination of the \TIE\MISSION subdirectory reveals many similar file names. Battle and Historical mission files are named using a standard convention. The Battle mission files are named using the format shown in Figure 1.

Using the diagram in Figure 1, the file named B7M3AW.TIE would reference Battle 7 Mission 3, where the player is flying a TIE Advanced vehicle in the Wingman position. Listing 3 shows the spacecraft name references. Historical mission file names are slightly different, as shown in Figure 2.

Using the Historical mission file naming diagram, a file named HG3M.TIE would reference Assault Gunboat Historical Mission 3, where the player flies an Assault Gunboat and is the

Figure 1. Battle Mission File Naming

```
B  [1-7]  M  [1-6]  V  [MW] . TIE
|     |     |     |     |   |    |W = player is the Wingman
|     |     |     |     |   |  M = player is the Flight Leader
|     |     |     |     |  Vehicle letter (see Listing 3)
|     |     |     |   Mission Number
|     |     |   M for Mission
|     |   Battle Number
|   B for Battle
```

Figure 2. Historical Mission File Naming

```
H  V  [1-4]  [MW] . TIE
|  |     |     |   |  W = player is the Wingman
|  |     |     |   M = player is the Flight Leader
|  |     |   Mission Number
|  |   Vehicle letter (see Listing 3)
|  H for Historical
```

Listing 4. Mission Files

```
BATTLE MISSIONS

BATTLE       FILE NAME (*.TIE)

Battle 1     B1M1FM B1M2FM B1M3BM B1M4IM B1M5GM B1M6GM
Battle 2     B2M1FW B2M2BW B2M3IW B2M4GW B2M5FW
Battle 3     B3M1BM B3M2BM B3M3FM B3M4GM B3M5BM B3M6GM
Battle 4     B4M1FM B4M2BM B4M3BM B4M4IM B4M5GM
Battle 5     B5M1IW B5M2GW B5M3GW B5M4AW B5M5GW
Battle 6     B6M1AW B6M2AW B6M3GW B6M4GW
Battle 7     B7M1AM B7M2AM B7M3AW B7M4DW B7M5DM


HISTORICAL MISSIONS

VEHICLE            FILE NAME (*.TIE)

TIE Advanced       HA1W   HA2W   HA3M   HA4M
TIE Bomber         HB1W   HB2W   HB3M   HB4M
TIE Defender       HD1W   HD2W   HD3M   HD4M
TIE Fighter        HF1W   HF2W   HF3M   HF4M
Assault Gunboat    HG1W   HG2W   HG3M   HG4M
TIE Interceptor    HI1W   HI2W   HI3M   HI4M
```

Flight Leader for the mission. Listing 4 gives the TIE Fighter mission file names.

Does Your Pilot Need Help?

If you are like me, your computer pilot needs help every now and then. Your pilot may die unexpectedly (the "I didn't see that one coming" scenario) and games such as TIE Fighter will penalize the player when his or her character is revived. In this section I discuss how some of the data in the pilot file is organized; you may find the information useful for pilot survival.

As mentioned previously, TIE Fighter stores pilot data in files suffixed with TFR. Pilot files are 3,856 bytes in length, but the size is somewhat misleading. During game play you can press the "escape" key to bring up the Personal Datapad, which can then backup your pilot file. The backup copy of your pilot data is stored in the same file as your primary pilot data. The first 1,928 bytes of the file are your active pilot data, and the last 1,928 bytes are backup data. When manually editing pilot files, avoid editing the backup data.

TIE defenders approaching a space platform.

Listing 5 gives a summary of several important data parameters and where they are located in your pilot file. (The FILE OFFSET references in the listing assume the first byte in the file is "byte 0".) The pilot file has open or add-on slots for future game expansion. There are slots for eight Historical missions for each vehicle you can fly, but the current game only has four Historical missions per vehicle. There are eight available mission slots for each Battle, but all Battles currently have less than eight missions.

Feel free to experiment with your pilot file (after backing it up, of course). It is very easy to change your skill level, scores, or reduce the number of vehicles you have lost. Personally, I usually edit the mission completion flags to mark a difficult mission as completed when my "computer pilot" gets a little too frustrated.

Until We Meet Again at the Imperial Starbase

As with many of LucasArts' gaming products, TIE Fighter is well written and executed. The improved graphic engine and artificial intelligence make the game a true leader in the space combat and simulator genre of games. The clean layout of the mission and pilot files makes tailoring the game much easier and lots of fun.

In the next Chopping Block column, we will dig into the TIE Fighter mission data structures. If you are interested in editing mission files or creating new missions, I would strongly urge you to pick up a copy of *The TIE Fighter Official Strategy Guide*. The mission statistics tables located in the back of the book were extracted directly from the mission file data. Once you understand how the mission

Getting ready to dock, cockpit view.

More TIE Fighter action.

data is stored (by reading the next Chopping Block column), the *Strategy Guide* will make mission file editing much smoother. ■

Wayne Sikes has been a computer hardware and software engineer for the last 10 years. He has an extensive background in C, C++, and assembly language programming. He also has several years experience as a computer systems intelligence analyst, where he specialized in deciphering and disassembling computer code while working on classified government projects. He has written numerous computer gaming help utilities. You can reach him via e-mail at 70733.1562@compuserve.com or through Game Developer.

Listing 5. Pilot File Data

FILE OFFSET (DECIMAL)	DATA TYPE*	DESCRIPTION
1	byte	Duty Status. 0=Alive, 1=Captured, 2=Killed
2	byte	Rank. 0=Cadet -> 5=General
3	byte	Difficulty Level. 0=easy -> 2=hard
4	long	Score.
8	word	Skill Level. 0=Novice -> 65535=Super Ace
10	byte	Secret Order Ranking. 0=None -> 6=Emporer´s Hand
29-34	byte	Next Training Level. off 29=T/F -> off 34=T/D **
42-62	long	Training Scores. off 42=T/F -> 62=T/D **
90-95	byte	Total Training Levels Completed. off 90=T/F -> 95=T/D **
136-164	long	T/F Historical Scores. 136=Mission1 -> 164=Mission8 ***
168-196	long	T/I Historical Scores. 168=Mission1 -> 196=Mission8 ***
200-228	long	T/B Historical Scores. 200=Mission1 -> 228=Mission8 ***
232-260	long	T/A Historical Scores. 232=Mission1 -> 260=Mission8 ***
264-292	long	GUN Historical Scores. 264=Mission1 -> 292=Mission8 ***
296-324	long	T/D Historical Scores. 296=Mission1 -> 324=Mission8 ***
520-527	byte	T/F Historical Completion Flags. 0=Not Done, 1=Done ***
528-535	byte	T/I Historical Completion Flags. 0=Not Done, 1=Done ***
536-543	byte	T/B Historical Completion Flags. 0=Not Done, 1=Done ***
544-551	byte	T/A Historical Completion Flags. 0=Not Done, 1=Done ***
552-559	byte	GUN Historical Completion Flags. 0=Not Done, 1=Done ***
560-567	byte	T/D Historical Completion Flags. 0=Not Done, 1=Done ***
617	byte	Battle 1 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
618	byte	Battle 2 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
619	byte	Battle 3 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
620	byte	Battle 4 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
621	byte	Battle 5 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
622	byte	Battle 6 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
623	byte	Battle 7 Status. 0=Inactive, 1=Active, 2=Pending, 3=Done
637	byte	Battle 1 Last Mission Completed. 0=None, 1=Mission1...
638	byte	Battle 2 Last Mission Completed. 0=None, 1=Mission1...
639	byte	Battle 3 Last Mission Completed. 0=None, 1=Mission1...
640	byte	Battle 4 Last Mission Completed. 0=None, 1=Mission1...
641	byte	Battle 5 Last Mission Completed. 0=None, 1=Mission1...
642	byte	Battle 6 Last Mission Completed. 0=None, 1=Mission1...
643	byte	Battle 7 Last Mission Completed. 0=None, 1=Mission1...
986-1014	long	Battle 1 Scores. off 986=Mission1 => 1014=Mission8 ****
1018-1046	long	Battle 2 Scores. off 1018=Mission1 => 1046=Mission8 ****
1050-1078	long	Battle 3 Scores. off 1050=Mission1 => 1078=Mission8 ****
1082-1110	long	Battle 4 Scores. off 1082=Mission1 => 1110=Mission8 ****
1114-1142	long	Battle 5 Scores. off 1114=Mission1 => 1142=Mission8 ****
1146-1174	long	Battle 6 Scores. off 1146=Mission1 => 1174=Mission8 ****
1178-1206	long	Battle 7 Scores. off 1178=Mission1 => 1206=Mission8 ****
1626	word	Total Kills.
1628	word	Total Captures.
1926	word	Number of Craft Lost.

* "byte" references an unsigned character.
"word" is a 16-bit unsigned value.
"long" is a 32-bit signed value.
** Vehicles are ordered as in Listing 3.
*** There are currently four historical missions for each flyable craft.
The pilot file has storage slots for eight historical missions per craft.
**** The pilot file has provisions for eight missions per battle.