UOS Internals Manual

Table of contents

Title page

UOS Internals and Data Structures

November 2018

Preface

Preface

The UOS Internals Manual provides developers and maintenance programmers with a comprehensive technical overview of UOS. The manual presents the philosophy, functionality and structure of UOS.

The data structures presented herein are shown using Pascal syntax, but they are not inherent to the Pascal programming language. All structures are "packed", meaning that items are aligned on byte boundaries. For clarity, the data types used in these structures are defined as follows:

Type	**Description**
byte	unsigned 8-bit integer
word	unsigned 16-bit integer
longint	signed 32-bit 2's complement integer
cardinal	unsigned 32-bit integer
int64	signed 64-bit 2's complement integer
pchar	pointer to text
pointer	generic pointer
string[n]	A byte length, followed by an array of characters that is n characters in length. Unused characters are set to 0.

Note that the size of pointers is platform dependent.

This document is divided into several sections, each dealing with a functionally separate aspect of UOS.

Part I: Overview
Part II: Terminal Services
Part III: The UOS File System
Part IV: System Initialization

Part I: Overview

Part I
OVERVIEW OF UOS FUNCTIONALITY

This part of the UOS Internals Manual presents an overview of the whole UOS Operating System.

Introduction

Introduction

UOS is a multiuser, general-purpose timesharing system. It can serve as the controller for embedded controllers, web servers, timesharing, development, and other purposes. Although hardware may provide lower limitations, UOS can serve up to 65,536 network disks, and up to 1,507,328 directly-connected disks or other random-access storage devices.

UOS consists of the UOS Executive, the Init secondary bootstrap, standard device drivers, and system utilities (known as Commonly Used System Programs or CUSPs). Some of the major features of UOS include the following:

- Interactive timesharing
- Scalability
- Platform agnostic
- User privileges
- Dynamic allocation and sharing of system resources
- Multi-tasking
- File processing and sharing
- Interprocess communication
- Network support
- Multiple File Systems
- Mirrored disks
- Network storage
- Clusters
- Disk caching
- Multi-core
- CDROM/DVDROM support
- Magnetic tape support
- Multiple shells
- Shared common code
- Internationalization support
- Multi-lingual programming support
- RAM Disk support
- Operator services
- Hardware error detection and logging
- System maintenance tools
- System reliability features

Preliminary Concepts

Preliminary Concepts

This section describes concepts that apply to the following sections.

Exceptions

Exceptions

An exception, in UOS terms, is a report of an error, encapsulated in a class. A pointer to an instance of this class can be passed to other objects, including other exception instances. Chained exceptions are exception instances that contain references to other exception instances, which contain references to others, and so on. The last exception added to the chain is the one that is passed to other code. It provides an error report from the called code that represents that code's perception of what went wrong. If further information is needed about *why* something went wrong, the next exception in the chain can be examined to provide that information. The calling code does not know ahead of time if the exception being returned is single or chained. A nil exception is simply an indication of no error.

UOS Exceptions have no inherent relationship to hardware exceptions or exceptions handled by various languages, such as C++. Each UOS Exception has a facility ID which uniquely identifies the component that generated the error, and a code which uniquely identifies the type of error from that component.

Stores

Stores

A store is essentially an array of bytes. UOS uses 64-bit pointers to access any point on the store. These pointers are integer offsets from the start of the store. So, a 1 Gb disk will have offsets 0 through 1,073,741,823. Thus, a 64-bit address range allows for a store as large as 18 quintillion bytes.

Mirror Sets

Mirror Sets

A mirror set is a collection of stores using RAID 1 to provide data redundancy and performance benefits. Mirror sets are logical stores that are denoted by store controller 25 (Z).

A member store of the mirror set has a structure at the end of the store that indicates information about the mirror set member:

```
type TMirror_Header = packed record
                          Signature : int64 ;
                          Set_Name : array[ 0..63 ] of char ;
                          Set_GUID : TGUID ;
                          Flags : cardinal ;
                          Sequence : int64 ;
                          Timestamp : int64 ;
                          Reserved : array[ 0..147 ] of byte ;
                      end ;
```

The following table describes the contents of this structure:

Item	Description
Signature	A value indicating a mirror set header: `FF FF 8C 01 00 8C FF FF`
Set_Name	The name of the mirror set that this is a member of (null terminated)
Set_GUID	The GUID associated with this mirror set
Flags	0 = normal, 1 = applies to partition
Sequence	Sequence number (see below)
Timestamp	Time of last synchronization
Reserved	Reserved for future use. Should be zeroes.

When a member of a set is mounted into the mirror set, if it has a later timestamp and sequence, it becomes the reference store. Otherwise, it is brought into synchronization with the rest of mirror set - a process called reconciliation. This involves copying the data from the reference store to the new member. If data is written to the store before reconciliation is finished, the new data is also written to the new store so that it will not become unreconciled as the set is updated.
While a member is unreconciled, it is never read from since it may contain out-of-date data.

RAM Stores

RAM Stores

RAM Disks implement stores in contiguous stretches of memory. Since they are dynamically added and removed, they (along with other stores, such as network storage) are assigned to their own virtual controller, which is store controller 24 (Y).

Partitions

Partitions

A store may be subdivided into partitions. UOS supports MBR and GFT partition schemes on any store.

File Heaps

File Heaps

A file heap is a type of heap that exists in a file. The file is treated as a contiguous, extendable, store, with a default cluster size of 16 bytes. The lowest offset of the file contains a header structure which defines how the heap is accessed.

```
type TFH_Header = packed record
                      Prefix : byte ;
                      Facility : byte ;
                      Version : byte ;
                      Reservedb : byte ;
                      Resolution : longint ;
                      Flags : longint ;
                      Reserved : longint ;
                      AT_Offset : int64 ;
```

```
          Origin : int64 ;
     end ;
```

Item	Description
Prefix	A value, that with the Facility value, indicates that this is a File Heap: FF
Facility	Indicates a file heap: 144 (decimal).
Version	File heap format version: 0
Reservedb	Reserved for future use. Should be 0.
Resolution	Resolution, in bytes, of allocation table (i.e. Cluster size). Default is 16 (decimal).
Flags	Reserved for future use. Should be 0.
Reserved	Reserved for future use. Should be 0.
AT_Offset	Byte offset in the file of the allocation table.
Origin	Byte offset in the file of the origin of the data stored in the heap.

Optional File Heap structures

Many file heaps make use of additional structures besides the header. This section describes these optional structures.

Store Strings

Store strings are arbitrary-length buffers. Typically they are used to store text. Store strings consist of a header structure, which points to the actual data:

```
TStore_String_Header = packed record
                          Prefix : TPrefix ; // "STR"
                          Version : byte ;
                          Length : longword ;
                          Flags : longint ; // reserved
                          RefCount : longint ; // reserved
                          Data : int64 ;
                       end ;
```

Item	**Description**
Prefix	A prefix, indicating that this is a store string header: "S", "T", "R"
Version	Store string header version. Should be 0.
Length	Length of the actual string data, in bytes.
Flags	Reserved for future use. Should be 0.
RefCount	Reserved for future use. Should be 0.
Data	Pointer to the store offset where the string contents are stored.

Store Lists

A store list is a collection of 64-bit integer values. The list is managed via an Allocation Cluster Manager. The list consists of a header that describes the list and a pointer to the data that is managed by the Allocation Cluster Manager:

```
TStore_List_Header = packed record
                         Prefix : TPrefix ; // "LIS"
                         Version : byte ;
                         Delta : longint ;
                         Capacity : longint ;
                         Max : longint ;
                         Data : int64 ;
                      end ;
```

Item	Description
Prefix	A prefix, indicating that this is a store string header: "L", "I", "S"
Version	Store list header version. Should be 0.
Delta	The number of 64-bit integers to extend the list by when an extension is required.
Capacity	The maximum (physical) size of the list, in items.
Max	The maximum logical size of the list, in items. Can never be larger than Capacity.
Data	Pointer to the data

Store String Lists

Store String Lists

A store string list is a store list with two differences:

- the prefix is different
- the 64-bit integer values in the list are treated as pointers to store strings.

```
TStore_List_Header = packed record
                         Prefix : TPrefix ;
                         Version : byte ;
                         Delta : longint ;
                         Capacity : longint ;
                         Max : longint ;
                         Data : int64 ;
                      end ;
```

Item	Description
Prefix	A prefix, indicating that this is a store string header: "S", "L", "I"
Version	Store string list header version. Should be 0.
Delta	The number of strings to extend the list by when an extension is required.
Capacity	The maximum (physical) size of the list, in strings.
Max	The maximum logical size of the list, in strings. Can never be larger than Capacity.
Data	Pointer to the data

The HAL

The HAL

All hardware access is via the HAL (Hardware Abstraction Layer). Any hardware platform that has a HAL written for it can run UOS.

The HAL standardizes the way that various CPU hardware features are accessed. These features include:

I/O Ports
Memory
Interrupts
Ring (protection) levels
Hardware devices
Numeric coprocessing

The HAL maps the UOS requests into the platform-appropriate code.

The implementation of the HAL is beyond the scope of this manual. The standard interface is as follows:

```
type THAL = class
                public
                    function Allocate_RAM( Size : int64 ) : pointer ;
                    function Boot_Device : TDevice_Info ;
                    function Console : TTerminal ;
                    procedure Deallocate_RAM( Value : pointer ; Size : int64 )
```

```
;
                     function Device( Index : integer ) : TDevice_Info ;
                     function Disable( Device_Type, Controller, Device_Unit :
word ) : word ;
                     function Enable( Device_Type, Controller, Device_Unit :
word ) : word ;
                     procedure Device_Reset ;
                     procedure Halt ;
                     function Heap : PHAL_Heap ;
                     function Memory( Index : integer ) : TMemory_Info ;
                     function Radix : longint ;
                     function Store( Index : integer ) : TCOM_Store64 ;
                     procedure Set_Timestamp( Value : int64 ) ;
                     function Timestamp : int64 ;
                     function Use_Line_Clock : boolean ;
                     function Get_Line_Clock : int64 ;
                     procedure Set_Line_Clock( Value : int64 ) ;
                     function Valid_Time : boolean ;
                     function Set_Configuration( Address : pointer ; Size :
cardinal ) : cardinal ;
             end ;
```

Method	**Description**
`Allocate_RAM`	Allocates `Size` bytes of memory and returns a pointer to the first byte. Returns null if the memory couldn't be allocated.
Boot_Device	Returns information on the boot device. See TDevice_Info structure.
Console	Returns an instance of a TTerminal object used to access the system console. Returns null if there is no system console device.
Deallocate_RAM	Deallocates `Size` bytes of memory starting at `Value`.
Device	Returns information on the device at the passed `Index`.
Device_Reset	Resets all devices to initial state.
Disable	Disables the specified device.
Enable	Enables the specified device
Get_Line_Clock	Returns current line clock value.
Halt	Halts the system.
Heap	Returns a pointer to a HAL Heap instance.
Memory	Returns memory information for memory segment `Index`.
Radix	The base used for this hardware. For instance, Intel hardware would return 16 (hexadecimal).
Set_Configuration	Sets the startup configuration to the passed data.
Set_Line_Clock	Sets the line clock to `Value`.
Set_Timestamp	Sets the current clock to `Value`.
Store	Returns an instance of a TStore that corresponds to the device at index `Index`. Returns null if the index is invalid or the device is not a store.
Timestamp	Returns the current clock value.
Use_Line_Clock	Returns true if the system has a line clock.

Valid_Time	Returns true if the clock has a valid value.

Note that this is an abstract interface and all of the methods use stdcall calling standard.

Date/Time
Date and time stamps in UOS use the Sirius Timestamp standard which is the number of 100ns intervals since 12:00:00 am, Jan 1, year 0 (projecting the current calendar back).

Devices

Devices

UOS hardware devices consist of three items:

1. Type
2. Controller
3. Unit

Type indicates the type of device. Controller is a number between 0 and 25, indicating which hardware controller, of the type. Unit is the device index on that controller.

The HAL keeps a list of devices, and can be queried for information about these devices by using an index (index 0 is the first device). The TDevice_Info structure is how this information is communicated back from the HAL.

The TDevice_Info structure is used to return information about a given device:

```
type TDevice_Info = packed record
                      Device_Type : word ;
                      Controller : word ;
                      Device_Unit : word ;
                      Disabled : boolean ;
                      Media_Present : boolean ;
                  end ;
```

The items in this structure are described in this table:

Item	**Description**
Device_Type	Type of device. See Device types, below.
Controller	Controller index
Device_Unit	Unit number of the device
Disabled	True if the device is disabled
Media_Present	True if media is present in the device.

Device types have the following values:

Value	**Mnuemonic**	**Description**
0	DT_Non_Existant	Invalid device index
1	DT_Unknown	Type not otherwise included here
2	DT_Serial	Serial (stream) device

3	DT_Store	Data store

Terminals

A terminal is a serial non-store device. The TTerminal class descends from the TDevice class.

```
type TDevice = class
                  public
                     procedure Cancel_Input ;
                     procedure Cancel_Output ;
                     function Pending_Input : boolean ;
                     function Pending_Output : boolean ;
                     procedure Poll ;
               end ;

type TTerminal = class( TDevice )
                    public
                       procedure Output( S : PChar ) ;
                       function Input( var C : integer ) : boolean ;
                       procedure Clear_Typeahead ;
                       function Peek : char ;
                       function Video : boolean ;
                       function Get_OnNewChar : TCharNotify ;
                       procedure Set_OnNewChar( Value : TCharNotify ) ;
                 end ;
```

The following table describes the TDevice methods:

Method	**Description**
Cancel_Input	Cancels pending input or input operation.
Cancel_Output	Cancels pending output or output operation.
Pending_Input	Returns True if there is any pending input or input operation.
Pending_Output	Returns True if there is any pending output or output operation.
Poll	Polls the device for operation completion.

The following table describes the TTerminal methods:

Method	**Description**
Output	Transmits the passed data. The data ends with a null.
Input	Retrieves the next waiting character from the device buffer.
Clear_Typeahead	Clears any characters waiting in the device buffer.
Peek	Retrieves the next waiting character from the device buffer but doesn't remove it.
Video	Returns true if the terminal is capable of video operations. Otherwise it returns false.
Get_OnNewChar	Returns the pointer to the new character delegation.
Set_OnNewChar	Sets the new character delegation.

Memory

Memory

The HAL provides access to the memory-related features of the hardware (such as paging), and also some minimal memory management, mostly for the purpose of Init and initial UOS startup. Memory may consist of several segments of varying characteristics. For instance, most systems have some of their physical memory address range mapped to ROM (such as the BIOS on PCs), whereas other memory is read/write and available for general use. The HAL considers each contiguous segment of the memory address space that is of the same general type as a separate area, which is assigned a unique index.

The HAL passes information about memory back to UOS using the following structure:

```
type TMemory_Info = packed record
                        Memory_Type : word ;
                        Low : int64 ;
                        High : int64
                    end ;
```

This information is only provided on physical memory, and is described below:

Item	**Description**
Memory_Type	Type of memory. See Memory types, below.
Low	Lowest address of this segment
High	Highest address of this segment

Memory types have the following values:

Value	**Mnuemonic**	**Description**
0	MT_Non_Existant	Non-existant memory
1	MT_RAM	Random access read/write memory
2	MT_ROM	Read-only memory
3	MT_WOM	Write-only memory

HAL Heap

The HAL Heap

The HAL Heap manager is an object instance that can be used to allocate/deallocate memory for Init and other non-UOS programs. The UOS executive has its own memory management component that it uses for the Executive heap. It is very basic and doesn't support anything more than allocating/deallocating areas of physical memory.

```
type THAL_Heap = object
                     public
```

```
        function Getmem( Size : Integer ) : Pointer ;
        function Free( P : Pointer ) : Integer ;
        function Realloc( P : Pointer ; Size : Integer ) : Pointer ;
    end ;
```

This is an abstract class. All methods use the stdcall calling convention. Note that the class doesn't support allocations of more than 2 Gb.

Item	**Description**
Getmem	Allocate `Size` bytes of memory and return a pointer to the first byte, Returns null if there wasn't enough contiguous memory available.
Free	Frees the memory allocated at address P.
Realloc	Reallocates the memory allocated at address P, to be `Size` bytes long. The function attempts to reallocate in place, but if it cannot, it allocates a new buffer and copies the data. The function returns the address of the resized buffer (whether or not it changed).

The Executive

The Executive

The UOS executive is the core part of UOS. It is made up of several components that work together. The executive interfaces directly with the HAL.

The Kernel

The Kernel

The Kernel is the executive component responsible for marshaling the rest of the executive components. It is the coordinator of communications between the various components and the HAL.

The File Processor (FiP)

The File Processor (FiP)

The FiP is the executive component responsible for managing devices and files. All device, file, and file system access go through this component.

The Heap Manager (HMC)

The Heap Manager (HMC)

The HMC executive component provides a memory heap manager for the other executive components.

The Interrupt Manager (IMC)

The Interrupt Manager (IMC)

The IMC executive component is responsible for managing the handling of interrupts.

The Memory Manager (MMC)

The Memory Manager (MMC)

The MMC interfaces directly with the HAL to manage the computer's memory. The HAL provides a somewhat abstracted interface to the memory management hardware. The MMC provides a view of the HAL memory interface that is compatible with the way that UOS uses memory. There are many memory management schemes. Being platform-agnostic, UOS has to be able to handle any of them.

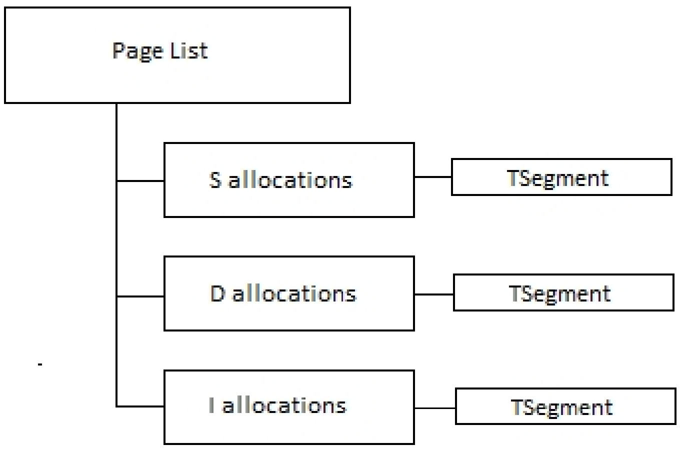
The purpose of the MMC is to manage memory by responding to requests to allocate RAM for a process.

MMC manages a master page table and a process page list. Since the executive does operations on behalf of the whole system, it operates as if it were its own process, separate from all other processes running on the system. A PID of 0 is used for executive-specific operations.

The Page_Table is a list of 32-bit integers, one per page (the HAL defines the size of a memory page). The low 16 bits of each value indicate the process ID that currently owns the page. The upper 16 bits are flags:

Mnuemonic	**Value (Hex)**	**Meaning**
Page_Flag_Locked	1000	Page is locked
Page_Flag_No_RAM	2000	Non-existent memory (unmapped in a paging scheme)
Page_Flag_Read_Only	4000	Physical read-only
Page_Flag_Write_Only	8000	Physical write-only
Page_Flag_Allow_Read	10000	Allow read access
Page_Flag_Allow_Write	20000	Allow write access
Page_Flag_Allow_Execute	40000	Allow execution

The MMC supports *allocation types*, which indicates the type of memory. The memory types are defined by the HAL. Each process has a page list that is a list of allocations per allocation type. On a typical modern CPU architecture, the allocation types are: Stack, Data, and Instruction (S, D, and I), and a page list would look like this:



Each set of allocations has one or more segments associated with it. If hardware requires contiguous segments, there will only ever be a single segment for each allocation type. Each segment is a represented with the following structure:

```
type TSegment = class
                    public // API...
                        Physical : int64 ;
                        Length : int64 ;
                        Flags : integer ;
                        Typ : char ;
                        Index : integer ;
                end ;
```

Name	Type	Description
Physical	int64	Starting physical address
Length	int64	Segment length, in bytes
Flags	integer	Page flags
Type	char	Allocation type
Index	integer	Type index

The System Services Component (SSC)

The System Services Component (SSC)

The SSC executive component provides various services for the other executive components. These services are provided as functions or classes.

The User and Security Component (USC)

The User and Security Component (USC)

The USC executive component manages users, processes, and security for UOS.

A user account is a uniquely-named set of attributes that define who is allowed to access the system and what they can do when logged in. Although each user has a name that is unique to the system, UOS uses an associated User ID Code (UIC) to identify the user. Unless otherwise specified, the term "user" herein will refer to a UIC and the associated attributes.

UICs are 4-byte unsigned integer values. UIC value 0 is used for the template user settings. That is, UIC 0 is not a user than can log in or be associated with ownership (a reference to UIC 0 in most places is an indication of an unknown, invalid, or unassigned, owner). However, within SysUAF, UIC 0 can be optionally defined. When it is defined, it serves as a template for all newly created user accounts. UIC 1 is always reserved for the Startup account. No user can ever log into the Startup account - its sole use is in starting up the system after the executive startup completes. In fact, that is the only way the Startup user is ever logged-in, via a call directly from the Kernel which forces the login. UIC 1 (and everything about it) is hard-coded. That way, a corrupted or missing SysUAF will never prevent system startup and/or recovery.
UICs 2 through 7 are reserved as special "system" accounts. Normally, when a new user is created in SysUAF, it is assigned an unused UIC greater than 7. Although a user with sufficient privileges can create a new user as a "system" account (UIC<8). System accounts are treated in special ways. For instance, a system user account can always log in, even if logins are disabled.

Privileges

Users are granted access to these features via privileges. Privileges are flags that indicate whether or not a user has a given ability to affect UOS. Most users have no privileges for security purposes - that way if the user runs a virus, it cannot harm the rest of the system. There are four privileges that affect the operation of protections:

- BYPASS - User has RWED access to all files, bypassing file protections
- READALL - User has read access to all files, bypassing file protections
- SYSPRV - User accesses all files via the file's system protection
- GRPPRV - User accesses all files via the file's group protection

The remaining privileges are:
- ACNT - Run processes with accounting disabled.
- ALLSPOOL - Allows user to allocate spooled devices.
- ALTPRI - Alter priorities.
- AUDIT - Allow audit records to be written.
- BUGCHK - Allow messages to error logger.
- CMEXEC - Allow calls to Change Mode to Supervisor system service.
- CMKRNL - Allow call to Change Mode to Kernel/Executive system service.
- DIAGNOSE - Allow user to run diagnostics and intercept error log messages.
- EXQUOTA - Allows user to exceed usage quotas.
- GROUP - Allows user to affect other processes belonging to common group.
- GRPNAM - Allows user to use /GROUP on mount and dismount operations.
- IMPERSONATE - Allows detached processes to be created with a different UIC.
- LOG_IO - Allow certain device control functions.
- MOUNT - Allows user to mount volumes.
- NETMBX - Allow network control operations.
- OPER - Allows use of OPCOM.
- PFNMAP - Allows unrestricted access to physical memory.
- PHY_IO - Allows physical I/O operations.
- PRMCEB - Allows creation/deletion of permanent common event flag clusters.

- PRMGBL - Allows creation/deletion of permanent global sections.
- PRMMBX - Allows creation/deletion of permanent mailboxes.
- PSWAPM - Allows control of swapping operations.
- SECURITY - Allow user to perform security-related functions.
- SETPRV - Allows user to create processes that have privileges greater than the user.
- SHARE - Allows user to open assigned devices or to assign nonshared devices.
- SHMEM - Allows user to create global sections and mailboxes in memory shared by multiple processors.
- SYSGBL - Allows user to create/delete system global sections.
- SYSLCK - Allows user to process locks.
- SYSNAM - Allows user to bypass access controls on system symbol tables.
- TMPMBX - Allows user to create temporary mailbox.
- VOLPRO - Allows user to override protections on volumes.
- WORLD - Allows user to control any/all other processes.

A user cannot grant privileges to any object that are greater than the privileges he has. For instance, a program can create sub-processes. These new processes can be granted any privileges that the user running the program has, but they cannot be granted privileges that the creator process doesn't have. The one exception to the rule is that the SETPRV privileges allows a process to grant additional privileges to itself or another process. It is possible, however, for a user to grant privileges to a exectuable such that any user that runs the executable will have those privileges while the program is running, even if the user running the program doesn't have those privileges. Certain CUSPs have such privileges in order to perform functions on behalf of a user without otherwise sufficient privileges.

A running process has four sets of privilege:

- Granted privileges: These are the privileges granted to the user account.
- Current privileges: This is a subset of the granted privileges that indicate which privileges are currently in effect.
- Program privileges: These are the privileges granted to the currently running program. If a program is not running, these flags are all cleared.
- Effective privileges: this is the same as Current privileges merged with any program privileges.

Whenever the user makes a request to UOS, the Effective privileges are checked against - regardless of the Granted privileges.

SysUAF

The System User Authorization File stores the user account information, including passwords. The file is named \uos\SysUAF.DAT on the boot device. SysUAF.DAT is a file heap store. This section describes the data structures stored within this file.

```
type Ptr = int64 ;
     TList_Ptr = Ptr ;
     TString_Ptr = Ptr ;
     TStringList_Ptr = Ptr ;
     TTimeStamp = Ptr ;

     TUAF_User = packed record
                     Name : TString_Ptr ;
```

```
        Flags : longint ;
        Authentication : TList_Ptr ;
        Access : TList_Ptr ;
        Shell : TString_Ptr ;
        LGICMD : TString_Ptr ;
        Home : TString_Ptr ;
        Privileges : int64 ;
        Auth_Privileges : int64 ;
        Expiration : TTimeStamp ;
        Owner : longint ;
        Priority : longint ;
        Quotas : TUAF_Quotas ;
        Last_Interactive_Login : TTimeStamp ;
        Last_Non_Interactive_Login : TTimeStamp ;
        Last_Login_Failure : TTimeStamp ;
        Login_Failures : longint ;
    end ;
```

Item	**Description**
Name	User's name.
Flags	User account flags. Valid flags are: UAF_Audit = If set, security auditing is enabled for the user. UAF_AutoLogin = If set, the user is restricted to the automatic login mechanism. UAF_Captive = If set, the user is prevented from changing any defaults at login, and cannot leave the LGICMD command procedure specified for the user. Further, Ctrl/Y interrupts are initially disabled. UAF_DefShell = If set, the user is restricted to UCL, the default UOS shell. UAF_DisCtlY = If set, Ctrl/Y interrupts are initially turned off. UAF_DisImage = If set, the user is prevented from executing the RUN command and any foreign commands. UAF_Disreconnect = If set, the user is disabled from automatic reconnection when an interactive session is interrupted. UAF_DisReport = If set, the login CUSP will not display the last login time, login failures, and other security reports. UAF_Disabled = If set, the account is disabled and logins are not allowed. UAF_DisWelcome = If set, the system welcome message is not displayed by the login CUSP. UAF_DisAuth = If set, the user is not required to provide authentication. UAF_Restricted = If set, the user is prevented from changing any defaults at login. Ctrl/Y is also initially disabled. Typically this is used to restrict a user to a specific application. UAF_Accounting = If set, user accounting information is written to accounting.dat file.
Authentication	Authentication schemes - list of pointers to TUAF_Authentication records
Access	Access list (see TUAF_Access record)
Shell	Default shell
LGICMD	Login command file name
Home	Default home folder
Privileges	Starting (default) privileges
Auth_Privileges	Authorized (allowed) privileges
Expiration	Account expiration
Owner	User to charge accounting to (0=this user)
Priority	Default priority
Quotas	A quota structure (see below)
Last_Interactive_Login	Timestamp of last interactive login.

Last_Non_Interactive_Login	Timestamp of last non-interactive login
Last_Login_Failure	Timestamp of last failed login attempt.
Login_Failures	Number of login failures since last successful login

Quota Structures

Quota Structures

This structure is used to store quota information for a user. It is embedded in the User header structure.

```
type TUAF_Quotas = packed record
                       ASTLM : cardinal ;
                       BIOLM : cardinal ;
                       BYTLM : cardinal ;
                       CPUTIM : cardinal ;
                       DIOLM : cardinal ;
                       ENQLM : cardinal ;
                       FILLM : cardinal ;
                       MAXACCTJOBS : cardinal ;
                       MAXJOBS : cardinal ;
                       PGFLQUOTA : cardinal ;
                       TQELM : cardinal ;
                       WSDEFAULT : cardinal ;
                       WSEXTENT : cardinal ;
                       PRCLM : cardinal ;
                       THREADLM : cardinal ;
                       ETIME : cardinal ;
                   end ;
```

Item	**Description**
ASTLM	Maximum number of simultaneous asynchronus I/O operations. When the quota is reached, asynchronous I/O requests are treated as synchronous I/Os.
BIOLM	Maximum number of I/Os that are buffered in system memory. I/Os can be written directly to user memory buffers, or to system I/O buffers and then copied to user buffers.
BYTLM	Maximum number of I/O bytes per session.
CPUTIM	Maximum CPU time per session.
DIOLM	Maximum number of simultaneous direct I/O requests. These are I/O requests that have user memory buffers waiting for the I/O completion. Reaching this limit will block the program until one or more direct I/Os complete.
ENQLM	Maximum number of locks that a process can simultaneously hold.
FILLM	Maximum simultaneous open files.
MAXACCTJOBS	Maximum number of non-network processes that can be active simultaneously.
MAXJOBS	Maximum total number of processes (interactive, batch, detached, and network) that can be active simultaneously. The first four network jobs are not counted.
PGFLQUOTA	Maximum number of pages in the page file that the process is allowed to use.
TQELM	Maximum simultaneous timed queue events (timers) that a process can have active.
WSDEFAULT	The number of memory pages that a process can use. Under some circumstances, WSEXTENT is used to limit process memory.

WSEXTENT	Maximum number of memory pages that a process can use.
PRCLM	Maximum number of simultaneous processes that a user can have running.
THREADLM	Maximum simultaneous threads per process.
ETIME	Amount of elapsed (connect) time per session.

Access Structures

Access Structures

Access structures in the access list each map onto 64-bit integers.

```
TUAF_Access = packed record
                  Typ : word ;
                  DOW : word ;
                  Starting : word ;
                  Ending : word ;
              end ;
```

Item	**Description**
Typ	Access type. One of the following values: UAT_Batch = Batch logins. UAT_Interactive = logins via physical connections (such as the monitor/keyboard on a PC) or via network (such as telnet). UAT_Network = Implicit logins that are used to authenticate the user when trying to access system resources via network connection. UAT_Remote = Logins through another UOS system serving as a gateway.
DOW	Day of the week. 0 = Sunday, 6 = Saturday.
Starting	Starting time of day: minutes after midnight.
Ending	Ending time of day: minutes after midnight.

Authentication Structures

Authentication Structures

Authentication structures are used to indicate what methods of authentication are used during logins.

```
TUAF_Authentication = packed record
                          Typ : longint ;
                          Expiration : int64 ;
                          Access_Type : longint ;
                          Auth : TString_Ptr ;
                          Flags : longint ;
                          Description : TString_Ptr ;

                          Lifetime : int64 ;
                          Last_Change : int64 ;
                      end ;
```

Item	Description

Typ	Authentication method:
	UAM_Password = Password
	UAM_Auth = External authentication

Expiration	If 0, the password doesn't expire. Otherwise, this is the timestamp of when the password expires.
Access_Type	Access type that this method applies to: UAT_Batch = Batch logins. UAT_Interactive = logins via physical connections (such as the monitor/keyboard on a PC) or via network (such as telnet). UAT_Network = Implicit logins that are used to authenticate the user when trying to access system resources via network connection. UAT_Remote = Logins through another UOS system serving as a gateway.
Auth	If this is a password authentication method, this is the password value. Note: this is a hashed value rather than the actual password text. If this is an external authentication method, then this is the name of the CUSP that is called to do the authentication. If Description is non-null, the user is prompted for a response before the CUSP is called. If Description is null, prompting is left up to the CUSP.
Flags	Flags. A combination of the following: UAMF_Generate = Automatically generate passwords
Description	If null, a default value is displayed to the user for authentication. For instance, for a password authentication method, the user would be prompted by the word "Password". Otherwise, this is the text that the user is prompted with.
Lifetime	This is ignored for non-password records. If 0, the password doesn't autoexpire. Otherwise, this is the length of time (in ns) until a new password expires. When a password is changed, the Expiration date is set to the current date/time plus this value.
Last_Change	Ignored except for password authentication records. Otherwise, this is the timestamp of the last password change.

Part II: Terminal Services

Part II
TERMINAL SERVICES

This part of the UOS Internals Manual describes the interface for terminal devices, associated structures, and classes. The hardware portion of the terminal interface and the HAL driver for that is not covered here. Terminal is a general term which indicates the means of interaction between the user and UOS.

Terminal Characteristics

Terminal Characteristics

Terminal devices have certain hardware characteristics that are strictly a function of that terminal. The hardware interface through which the terminal connects also has certain characteristics that are strictly under the control of the hardware. Much of this functionality happens outside of the knowledge and control of UOS. However, certain features of different terminals are supported by UOS so that many different forms of terminals and hardware interfaces are supported.

The following flags are used to inform UOS as to the terminal characteristics.

Value	Mnuemonic	Meaning
1	TF_Video	set to indicate that the terminal is a video terminal as opposed to a hardcopy (printer) terminal.
4	TF_Uppercase_Only	set to indicate that the terminal can only process uppercase characters. Otherwise, the terminal can process both uppercase and lowercase characters.

These flags have ramifications for how the input and output filters handle characters.

Output Filters

Output Filters

Output to terminals is cooked by an output filter which handles certain special conditions. Output filters have a flag which indicates the options related to outputting characters to the terminal.

Value	Mnuemonic	Description
1	TOFF_Null	Indicates that output is to be ignored when it is sent to the output filter. The user can toggle output on/off by means of the Control-O character. When the output is turned off, the TOFF_Null flag is set and any output sent to the filter is ignored and not sent to the terminal.
2	TOFF_Paused	Indicates that flow-control has paused output until the terminal or user is ready to view more output. When paused, the output filter buffers characters sent to it. Output can be paused/resumed by the terminal sending XON and OXFF when its buffer fills and empties. Most terminals also allow the user to use control characters to send XON/XOFF characters to manually pause or resume output. When an XOFF character is received by the terminal, the TOFF_Paused flag is set. When XON is received, the flag is cleared.
4	TOFF_Noformfeed	Indicates that the terminal doesn't process the form feed control character and would like the output filter to simulate it with line feeds.
8	TOFF_Notab	Indicates that the terminal doesn't process tabs characters. In this case, the filter simulates tab stops with spaces. By default, tab stops are considered to be every 8 columns.
16	TOFF_NoWrap	Indicates that the output filter will not output a CR/LF combination when the printing position reaches the terminal width. Normally, the output filter will do this to automatically wrap output.
32	TOFF_Noverticaltab	Indicates that the terminal doesn't process vertical tabs and wants the output filter to simulate it with line feeds.

The output filter is responsible for keeping track of the current position (column and line) on the terminal. By default, most control characters not mentioned above will be converted into a form of a circumflex followed by a letter. For instance, Control-B would be converted to "^B" on output.

For terminals that only support uppercase characters, the filter converts lowercase characters to uppercase.

XON/XOFF Flow Control

XON/XOFF Flow Control

Most terminals can use the XOFF/XON (transmit off/transmit on) protocol. This allows both the terminal and the computer to exercise some control over the data they receive by telling the other side when to start and stop transmitting characters. This is accomplished by sending XOFF (ASCII value 19) to stop the transmission and XON (ASCII value 17) to start it.

Most terminals have input buffers used to store received characters when they are received faster than the terminal can display/print. Likewise, some computer hardware used to receive terminal input also has a buffer to store received characters until the computer software has a chance to process them. XON/XOFF is used to prevent these buffers from filling and losing characters. Because hardware buffers tend to be relatively small, UOS also maintains buffers for both input and output to terminals. Thus, UOS will also send XON/XOFF to the terminal as its buffers fill and empty.

XON corresponds to the Control-Q and XOFF corresponds to Control-S. Because it is impossible for the computer to determine if an XON or XOFF was sent by the terminal hardware or by the user typing Control-Q/S, the user can use these key combinations to manually turn terminal reception of characters on and off.

If the terminal doesn't respond to XON/OFF, any characters received after the UOS input buffer is filled are ignored (lost). UOS has input flags that can allow UOS to ignore XON/XOFF in terms of flow-control and treat them as normal character input. Even if the terminal is in binary mode, XON/XOFF is treated as flow-control, unless the ignore-XON/OFF flag is set.

Note that XON/XOFF flow control is one-way, so that transmission can be paused in one direction but not in the other. Only if both ends (computer and terminal) send XOFF is transmission paused in both directions.

Fill counts

Fill counts

Some older terminal hardware requires time to complete a physical action initiated by a control character. For instance, after receiving a carriage return (CR) character, the DEC LA30 terminal needed time before the print head returned to the beginning of the next line. If characters were received before this, they would display at whatever point the print head was as it returned. Such old terminals are very rare, but UOS provides backwards compatibility for them.

Input Filters

Input Filters

Input from terminals is cooked by an input filter which handles certain special conditions. Input filters have a flag which indicates the options related to inputting characters to the terminal.

Value	Mnuemonic	Description
1	TIFF_Binary	Don't cook input
2	TIFF_Noecho	Don't echo characters
4	TIFF_NoControl	Echo control codes exactly
8	TIFF_Notypea	Don't buffer unsolicited input (no input is buffered while this is set)

	head	
16	TIFF_NoTermXON	UOS treats XON/XOFF sent by terminal as data rather than start/stop output
32	TIFF_NoLineEdit	Don't support line editing.
64	TIFF_BSasDEL	Treat backspace as Delete
128	TIFF_NoControlT	Don't provide status for Control-T
256	TIFF_NoControlC	Don't interrupt for Control-C
512	TIFF_NoControlY	Don't interrupt for Control-Y

The input filter is responsible for keeping track of the current position (column and line) on the terminal.

Echo Control

Echo Control

Under normal circumstances, UOS echoes characters as they are typed - but only when the running program is requesting input. Characters typed when input isn't being requested are stored in the input buffer (also known as the "type-ahead" buffer). They are echoed when the input buffer is processed if the terminal hasn't been set to binary input mode. There are two reasons for not echoing characters unless a program has requested input: 1) echoing characters at other times may interfere with formatted output, and 2) the input may be a password which the program will not want echoed for security reasons. Certain control codes are processed immediately upon receipt, rather than storing them in the type-ahead buffer for later processing. Such characters are either echoed or otherwise processed immediately.

The echo ability can be turned off by setting the flag that prevents echo or by setting the terminal to binary mode. Turning echo off can be used to protect a password while it is being typed (or processed). Some terminals have a "local-echo" feature that automatically displays the typed characters as they are typed and sent to the computer. In such a case, if the computer also echoes the character, the characters can be doubled on the terminal. In such cases, the echo can be turned off to prevent the doubled characters.

Delimiters

Delimiters

A delimiter is a character used to indicate the end of a line of input. Delimiters are treated as normal characters if the terminal is in binary input mode. Standard delimiters include carriage return (CR - control-M - ASCII value 13), line feed (LF - control-J - ASCII value 10), control-Z (ASCII value 26), and Escape (ASCII Value 30).

Binary Input

Binary Input

When a terminal is in binary input mode, the input data is passed directly to the program without alteration or processing, except for XON/XOFF flow control (unless the XON/XOFF processing is disabled). Standard delimiters and other special characters are neither required nor recognized. This includes Control-C, Control-Y, and Control-T.

Video Terminals

Video Terminals

Video terminals are treated differently than hard-copy terminals when it comes to line-editing processing. Characters on video terminals can be edited in-place. For instance, a delete operation will output a backspace, a space, and another backspace in order to erase the character. On hard-copy terminals, by comparison, a delete operation will display deleted characters between backslashes.

Part III: The UOS File System

Part III
THE UOS FILE SYSTEM

This part of the UOS Internals Manual presents an overview of the UOS File System. UOS can support multiple file structures. Most of these are documented elsewhere and are beyond the scope of this manual. The native file system for UOS is called the UOS File System (or UOSFS).

Stores

Stores

The UOSFS is a complex data structure which exists on a store.

Disks are one of the type of stores that UOS uses. Disks can read and write entire sectors, the minimum writable number of bytes is called the cluster size.

Managed Stores
The File System will need to allocate chunks of the available storage space as required. Also, as things are deleted, it will need to deallocate those chunks so they can be reused for other purposes. The management of free and allocated space is the domain of the store itself - not the file system. A managed store keeps track of free and allocated space, which is typically done through an allocation table (a data structure that allows the store to manage the space). The UOS File System can deal with any cluster size of 256, or more, bytes.

The allocation table is a bit array, where each bit represents a cluster on the store. If the bit is set, the corresponding cluster is in use; if it is not set, the cluster is available.

UOSFS Root Store Layout

UOSFS Root Store Layout

A store that has been initialized to hold a UOS File System has a structure that allows UOS to locate information about the store, the file system, and where to find the location of the root of all files stored in the file system.

```
type FS_Boot_Record = packed record
                          Bootstrap1 : array[ 0..15 ] of byte ;
                          Header : int64 ;
                      end ;

     FS_Header = packed record
                     ID : smallint ;
                     ID1 : byte ;
                     Version : byte ;
                     AT_Offset : int64 ;
                     AT_Size : int64 ;
                     Flags : int64 ;
                     Clustersize : cardinal ;
                     Folder_Clustersize : cardinal ;
                     Root : int64 ;
                     Volume_Label : string[ 127 ] ;
                     Password : string[ 31 ] ;
                     OS_Position : int64 ;
                     OS_Length : cardinal ;
                     MMC_Position : int64 ;
                     MMC_Length : cardinal ;
                     HMC_Position : int64 ;
                     HMC_Length : cardinal ;
                 end ;
```

The first cluster on a store contains the boot block, which is the bootstrap program for the store. The first 16 bytes are reserved for the bootstrap, followed by an 8-byte pointer to the file system header, and then followed by the rest of the bootstrap code.

The File System header contains the size and location of the store's allocation table, information on the cluster sizes, flags, the location of the root folder, and other information.

Item name	**Description**
ID	255
ID1	135
Version	10 (indicates version 1.0)
AT_Offset	Offset of allocation table
AT_Size	Size of allocation table, in bytes
Flags	UOSFSF_Dirty: 0=clean, 1=dirty UOSFSF_Private: 0=private, 2=public
Clustersize	File system cluster size
Folder_Clustersize	Cluster size for folders
Root	Location of file system root
Volume_Label	Label for store
Password	Password for store
OS_Position	Offset of Kernel image on store
OS_Length	Length of Kernel image
MMC_Position	Offset of Memory Management component on store
MMC_Length	Length of Memory Management component
HMC_Position	Offset of Heap Management component on store

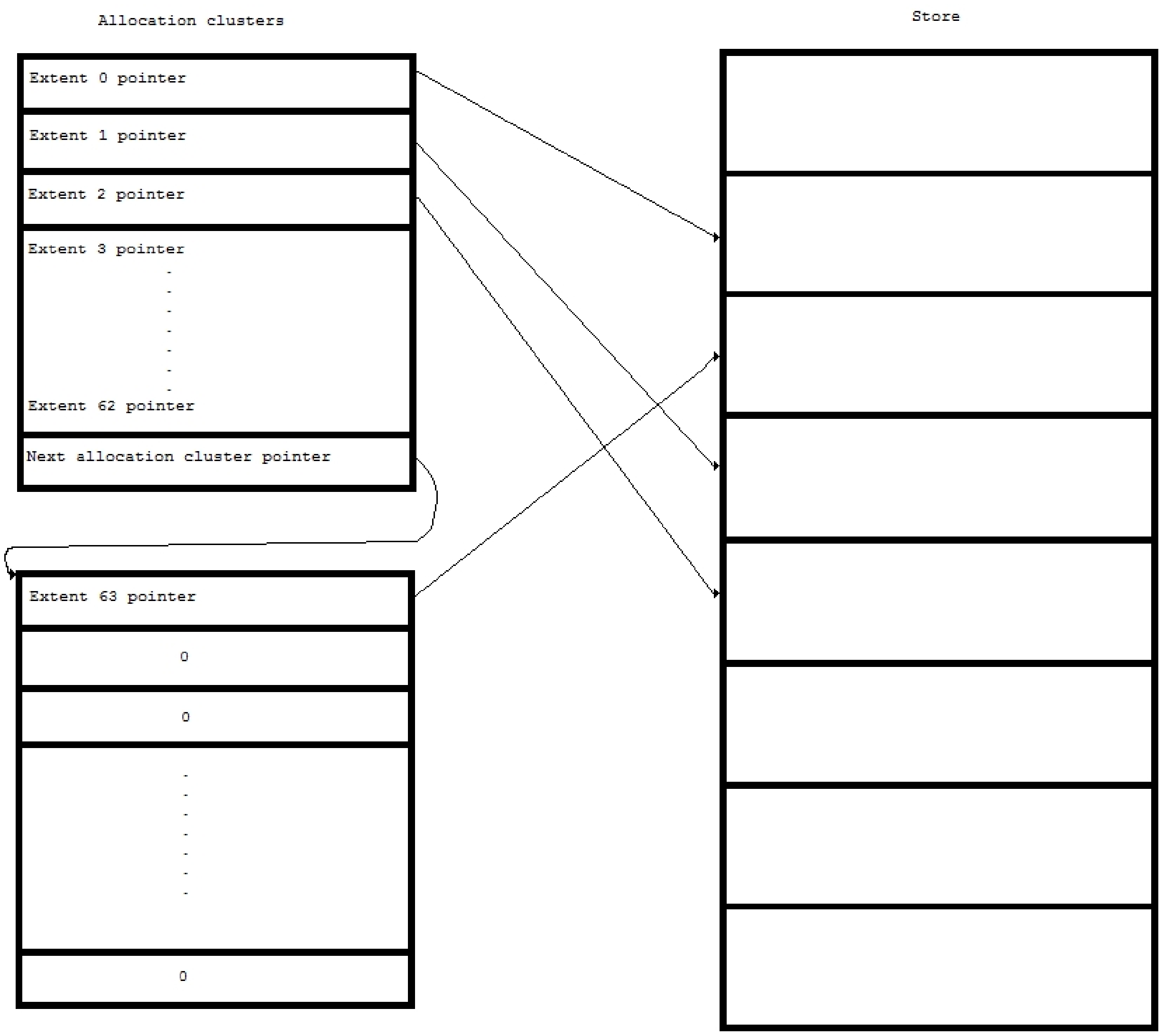
HMC_Length	Length of Heap Management component

Allocation Clusters

Allocation Clusters

Files in the UOSFS are stored in extents that can exist anywhere on the store. This allows an efficient use of the store's total capacity at the expense of some overhead. Each file, unless it has zero length, has an allocation cluster chain which is simply an array of 64-bit pointers. Each non-zero pointer indicates the location of an extent for the file on the store. An entire cluster is used to store a set of extent pointers, which means the number of pointers kept in a cluster is the cluster size divided by 8. However, the last pointer in the cluster is not a pointer to a file extent, but a pointer to the next cluster in the allocation cluster chain. In this way, a file can be of any size. The process of moving from one cluster to another while reading the allocation chain is called a "turn".

Here is a diagram of an allocation cluster chain on a store with a cluster size of 512:



Native Files

Native Files

Files on the UOSFS are tracked via a file header, which contains a pointer to an allocation cluster chain. The header holds five extent pointers so that small files don't use extra space for allocation clusters. Further, the number of allocation clusters required can be reduced by increasing the file's cluster size to some multiple of the store's cluster size.

The native file header is a structure matching the following definition:

```
type TData_Stream = packed record
                        Name : int64 ;
                        Pointer : int64 ;
                    end ;

     TUOS_File_Header = packed record
                                Name : longint ;
                                Size : int64 ;
                                EOF : int64 ;
                                Uncompressed_Size : int64 ;
                                Clustersize : cardinal ;
                                Record_Size : cardinal ;

                                // Dates...
                                Creation : int64 ;
                                Last_Modified : int64 ;
                                Last_Backup : int64 ;
                                Last_Access : int64 ;
                                Expiration : int64 ;

                                Creator : cardinal ;
                                Owner : cardinal ;
                                ACL : int64 ;

                                Flags : int64 ;
                                Version_Limit : longint ;
                                Extension : int64 ;

                                Streams : array[ 0..4 ] of TData_Stream ;
                                Data_Stream : int64 ;
                                Clusters : array[ 0..4 ] of int64 ;

                                Parent : int64 ;
                                File_System : int64 ;
                        end ;
```

The contents of the file header are described in the following table:

Header item	**Description**
Name	File name index
Size	File size on disk
EOF	Logical end of file offset
Uncompressed_Size	The size of the file when uncompressed
Record_Size	Size for record in record-oriented files
Creation	Creation date

Last_Modified	Date last modified
Last_Backup	Date last backed up
Last_Access	Date last accessed (read or write)
Expiration	Date of file expiration
Creator	ID of the user who created the file
Owner	ID of the user who owns the file
ACL	Pointer to Access Control List
Flags	A set of bit flags. **General:** FAF_DSM_MASK = Mask for data security mode FAF_PLACED = Placed at a specific location FAF_CONTIGUOUS = Contiguous file data FAF_DELETED = Deleted file FAF_READONLY = Read-only data FAF_LINK = Link to another file FAF_SYSTEM = System file FAF_HIDDEN = Hidden file FAF_DIRECTORY = Directory file FAF_PERMANENT = Non-deletable file FAF_COPYING = In the process of copying **Protections:** FAF_PROTECTION_OWNER_READ - Owner can read FAF_PROTECTION_OWNER_WRITE - Owner can write FAF_PROTECTION_OWNER_DELETE - Owner can delete FAF_PROTECTION_OWNER_EXECUTE - Owner can execute FAF_PROTECTION_GROUP_READ - Group members can read FAF_PROTECTION_GROUP_WRITE - Group members can write FAF_PROTECTION_GROUP_DELETE - Group members can delete FAF_PROTECTION_GROUP_EXECUTE - Group members can execute FAF_PROTECTION_SYSTEM_READ - System accounts can read FAF_PROTECTION_SYSTEM_WRITE - System accounts can write FAF_PROTECTION_SYSTEM_DELETE - System accounts can delete FAF_PROTECTION_SYSTEM_EXECUTE - System accounts can execute. FAF_PROTECTION_WORLD_READ - Everyone can read FAF_PROTECTION_WORLD_WRITE - Everyone can write FAF_PROTECTION_WORLD_DELETE - Everyone can delete FAF_PROTECTION_WORLD_EXECUTE - Everyone can execute
Version_Limit	Maximum version limit (0=no limit)
Extension	Pointer to header extension (reserved)
Streams	Meta-data stream pointers and names

Data_Stream	Data stream pointer
Clusters	Pointers to first 5 extents
Parent	Parent directory pointer
File_System	Parent directory offset

Data Streams
Files contain data. UOS file can contain meta-data in addition to data. Some meta data is in the file header, but UOSFS allows additional information to be stored in the file, but kept separate from the file's actual data. This is accomplished via "streams". A file can have any number of data streams. Data stream 0 is unnamed and reserved for the file's actual data. All other data streams can be assigned names that indicate what data is being stored in each stream.

Contiguous Files
Contiguous files (FAF_CONTIGUOUS flag) are stored in contiguous clusters. Therefore, allocation chains are not required. Instead, the first extent in Clusters points to the first cluster of the contiguous data.

String Tables

String Tables

File names and data stream names are stored in the file header as integer values, which are indexes into the store string table. This table is a group of three files in the \Store folder which contain all the names used on the store. The files are

1. An allocation table that tracks what space is used and available in the string data file. Each bit in the table maps 8 bytes in the string data file.
2. A string data file which contains the actual name text. This file can be up to 4 Gb in length.
3. An index table which contains pointers into the string data file, organized in a way that provides a sorted (ascending) list of the names

The string data file contains string records. Each record has the following format:

Byte Offset	**Description**
0-3	Reference count. Files with the same name share the same string table offset. This indicates how many references exist to this value on the store.
4	String length (up to 256 bytes). 0 represents a length of 256.
5-n	String contents.

Standard Files

Standard Files

The standard files and folders on a UOS File System are as follows (leftmost entries are top-level folders in the root folder):

Store - Store-related files exist in this folder
    AT.sys - string allocation table
    BadBlocks.sys - contains all bad clusters on the store
    Index.sys - string indexes

Strings.sys - string table data
System - System-wide files, such as installed products exist in this folder
UOS - Operating System files are kept in this folder
installed.sys - link to hooked UOS installation
default.sys - link to the default UOS installation
startup.dat - configuration information passed from Init to UOS
sysuaf.dat - system user authorization file (user accounts).
accounting.dat - user accounting file
all other folders are the UOS installations
Users - User home directories exist in this folder

Part IV: System Initialization

Part IV
SYSTEM INITIALIZATION

This part of the UOS Internals Manual describes the UOS initialization.

Introduction

Introduction

The process of booting UOS involves three steps of loading and executing:

1. Primary bootstrap
2. Init (secondary bootstrap)
3. UOS

The primary bootstrap is the normal bootstrapping procedure appropriate to the hardware.

Passing the Configuration to UOS

Passing the Configuration to UOS

Configuration information is passed to UOS in a file on the boot device. If the boot device is read-only, the information is written to a buffer. In either case, the configuration data contains a series of records with configuration data. Each record consists of a one byte record type, followed by one or more parameters appropriate to the type of record. The parameters consist of fixed and variable-length values. The fixed values vary in size from 1 to 8 bytes (1, 2, 4, and 8 byte values). The size is encoded in the record type.

Mask (Record type and 3)	**Description**
0	1 byte
1	2 bytes
2	4 bytes

3	8 bytes

Thus, record types 0-3 are the same record type, but with different sizes for fixed-size parameters. Here are the record types supported:

Record type(s)	Description	Parameters
0-3	Disabled device	type, controller, unit
4-7	RAM disk	address, size
8-11	Boot device	type, controller, unit

Note that any record types not shown are reserved for future use.

Kernel Startup

Kernel Startup

The kernel is responsible for preparing the UOS executive for use. Any errors encountered during this process will cause the startup to fail. The following steps are performed by the Startup method of the Kernel class instance:

1. Load file system component from the boot device
This is done by reading the boot block and extracting the file system position and length.

2. Load Memory Management component from boot device
3. Load Heap Management component from boot device
4. **Set up Memory Management component**
This is done by setting the MMC's kernel.

5. Set up Heap Management component
This is done by calling the HMC's End_Startup method.

6. Mount the boot device's file system
If the file system is dirty, it is rebuilt and then mounted.

7. Get path to installed UOS
This is done by getting the link from the \uos\installed.sys file.

8. Load Interrupt Manager component
9. Load File Processor component
10. Load System Services component
11. Dismount the file system and remount via the File Processor
12. Get startup configuration
If the store is read-only, the configuration is read from a buffer maintained by the HAL. Otherwise, it is read from \uos\startup.dat.

13. Process startup configuration
14. Add any RAM disks from the startup configuration to the File Processor
15. Load User Security component
16. Define sys$system: to point to booted installation
17. Create startup process and attach it to the system console
18. Log the startup process in to user 1
19. Run sys$system:startup.ucl