-OSTA-®
Optical Storage
Technology Association

UDF 2.60 approved errata

Base Document: OSTA Universal Disk Format Specification, revision 2.60
Document: UDF 2.60 approved errata
Date: December 16, 2005; last modified July 21, 2006

Purpose of this document:

This document contains the UDF Document Change Notices (DCNs) that were approved as UDF 2.60 errata by the OSTA UDF Committee.

Important notice: **UDF 2.50 rules identical to UDF 2.60 for non-POW.**

It is the intention to keep the UDF 2.50 and UDF 2.60 rules identical for non-POW media, see more detailed explanation in the UDF 2.50 errata document.
This assumes that non-POW DCNs that are approved as UDF 2.60 errata are also approved as UDF 2.50 errata and that POW and non-POW issues are dealt with in separate DCNs.

History of this document:

01-03-2005: Release of the approved UDF revision 2.60 document.
16-12-2005: Added DCN 5152 as approved on September 27, 2005.
11-01-2006: Added DCN 5151, 5153 and 5156 as approved on December 05, 2005.
There is an extra annex document to DCN 5156.
20-07-2006: Added DCN 5155, 5157, 5159, 5162 as approved on March 02, 2006.
21-07-2006: Added DCN 5154, 5160, 5161 as approved on June 12, 2006.

UDF 2.60 approved errata

Contents:

Addition to DCN history table in appendix 6.17 for next UDF revision x.yz

{editorial: x.yz is the next UDF revision after UDF 2.60}

{editorial: page number column is local for this document and will be removed for incorporation in next UDF revision appendix 6.17}

Description	DCN number	Updated in UDF Revision	Minimum UDF Read Revision	Minimum UDF Write Revision	Page nmb
DCNs approved after release of UDF 2.60					
Recommendations DVD-R DL LJR	5151	x.yz	1.02	1.02	3
Stream bit ZERO for main data stream	5152	x.yz	2.00	2.00	5
Relaxation of file timestamps relation rule	5153	x.yz	1.02	1.02	6
Requirements for HD DVD Disc	5154	x.yz	2.50	2.50	7
Add recommendations for DVD+R DL and DVD+RW DL	5155	x.yz	1.02	1.02	10
Macintosh OS X additions Annex to DCN-5156: Resulting C code of 6.7.2	5156 + annex	x.yz	1.02	1.02	14 25
Unicode Version and Unicode Normalization Form	5157	x.yz	1.02	1.02	31
Add additional recommendations for BD Read-only Disc	5159	x.yz	2.50	2.50	33
More prominent role for Extended File Entry	5160	x.yz	2.00	2.00	35
Treat Fixed Packets in the same way as ECC Blocks	5161	x.yz	1.50	1.50	38
Simplification of UDF Developer Registration Form	5162	x.yz	1.02	1.02	40

-OSTA®
Optical Storage
Technology Association

UDF 2.60 approved errata

Document:	**OSTA Universal Disk Format**	**DCN-5151**
Subject:	*Recommendations DVD-R DL LJR*	
Date:	June 15, 2005; last modified: December 05, 2005	
Status:	Approved December 05, 2005	

Description:

This DCN is for the next UDF revision after 2.60 **and as errata for all previous UDF revisions**.
DVD-R DL LJR introduces a new method of recording named Layer Jump Recording (LJR) as described in the MMC and Mt Fuji specifications. Although similar to incremental recording, this new recording is slightly different. Reserved R-Zones and LJBs (Layer Jump Block) of DVD-R DL LJR do not match the definition of a single UDF track, but two logical tracks. Consequently, Border does not match the UDF session definition. LJR also introduces the possibility to remap anchor point sectors.
UDF multi-session is not straightforward on DVD-R DL LJR, so this DCN describes how to perform multi-Border / multi-session recording on DVD-R DL LJR

Change:

Add:

6.xx Recommendations for DVD-R DL LJR (Multi-Border recording)

This appendix defines the recommendations on volume and file structures for DVD-R DL LJR, to support the interchange of information between users of computer systems.

1. The volume and file structure should comply with UDF 2.00.
2. The Minimum UDF Read Revision and Minimum UDF Write Revision should be 2.00.
3. The length of logical sector and logical block shall be 2048 bytes.

Additionally, the following recommendations are made for DVD-R DL LJR:

The DVD-R DL LJR does not follow the usual session rules. On DVD-R DL LJR the start of each Border corresponds with the start of a new session, as usual. However, the end of each session is always the end of the disc. This results in overlapping sessions, which is not strictly according to the session definition in 1.3.2.
DVD-R DL LJR is a fixed size medium. Each R-Zone contains one or more LJB (Layer Jump Block). For each R-Zone, READ TRACK INFORMATION returns 1 (physical) track but UDF implementations need to consider it as two logical tracks per LJB: one on layer 0, one on layer 1. Boundary of the logical track containing the current NWA for the R-Zone is indicated by the Next Layer Jump Address.

The formula to calculate the start address of the second logical track (on L1) can be found in the Mt. Fuji specification.
Files may start in a track of layer 0, respectively 1, and continue in a track of layer 1, respectively 0, so UDF implementations should take care to write corresponding file extents.

For DVD-ROM drive compatibility, UDF implementation should close the Border.

6.xx.0 DVD-R DL LJR Differences
DVD-R DL LJR with remapping slightly differs from recommendations in 6.11

Differences with 6.11.3 Multi-session Usage:

- After the first session, at least 2 of the AVDPs at the logical sector numbers 256, *N*-256 and *N* and at least the AVDPs remapped in the previous session, are *remapped* from the last session. The remapping requires writing in the last session AVDPs with location tags of 256, *N*-256 and/or *N*, then instruct the drive to remap with the Remapping Address (Format Code = 24h) of SEND DISC STRUCTURE command, using Anchor Point Number 2, 3 and/or 4 for respectively 256, *N*-256 and/or *N*.
- After the first session, at least 2 of the AVDPs at logical sector numbers S+256, C-256 and C are written, where C is the LRA of the last session.

Document: **OSTA Universal Disk Format** **DCN-5152**
Subject: *Stream bit ZERO for main data stream*
Date: September 8, 2005; last modified September 27, 2005
Status: Approved September 27, 2005

Description:

This DCN is meant for the next UDF revision after 2.60 **and as errata for UDF 2.00 till UDF 2.60 inclusive**.
There is confusion whether the ICBTag Flags Stream Bit of an Extended File Entry must be set to ONE if a Stream Directory is attached, because this EFE is referenced by the Parent FID in the Stream Directory. The confusion is raised by the unfortunate text of Note 24 in ECMA 4/14.6.8 bit 13. The Stream Bit is meant to distinguish between the main data stream and named data streams as defined by ECMA 4/8.8.3. E.g. if a repair utility finds a File Entry or Extended File Entry with the Stream Bit set, it knows that it must search for a reference in a Stream Directory instead of a normal directory. Note 24 of ECMA 4/14.6.8 in fact aims at a different situation, i.e. a hard link between a named data stream of a file and the main data stream of another file. This type of hard link is not allowed by any UDF revision.

Change:

In 2.3.5.4 replace: **NOTE:**

by: **NOTE 1:**

and at the end of 2.3.5.4 add:

Bit 13 (*Stream*):

~~Shall indicate (ONE) whether a File Entry or Extended File Entry defines a Named Stream or the main data stream of a file or directory, see ECMA 4/8.8.3 and UDF 3.3.5.~~

Shall be set to ONE for a FE or EFE defining a Named Stream. It shall be set to ZERO in all other cases.

NOTE 2: The Stream bit shall be set to ZERO for the FE or EFE of the main data stream of a file or directory and for the FE or EFE of the System Stream Directory. This is so in spite of the fact that such a FE or EFE may be referenced by the Parent FID in a Stream Directory, thus excluding the parent FID case from Note 24 in ECMA 4/14.6.8.

Document: **OSTA Universal Disk Format** **DCN-5153**
Subject: *Relaxation of file timestamps relation rule*
Date: November 04, 2005
Status: Approved December 05, 2005

Description:

This DCN is for the next UDF revision after 2.60 and as errata for all previous UDF revisions 1.02 thru 2.60.
Because of a different definition of the file creation time in different Operating Systems, it is difficult for UDF implementations to always ensure that the Modification, Access and Attribute Date and Times "shall not be earlier than the File Creation Date and Time", as required by ECMA. Therefore these rules will be changed from mandatory to a recommendation as decided in the September 27, 2005 UDF committee meeting.
Editorial: "Time" replaced by "Date and Time" to be consistent with ECMA. This will be changed for the whole UDF spec, see the editorial DCN-5150.

Change:

In 2.3.6 replace: struct timestamp AccessTime;
struct timestamp ModificationTime;
struct timestamp AttributeTime;

by: struct timestamp **AccessDateAndTime;**
struct timestamp **ModificationDateAndTime;**
struct timestamp **AttributeDateAndTime;**

after section 2.3.6.8 add:

2.3.6.9 Access, Modification, Creation and Attribute Timestamps

ECMA sections 4/14.9.12-14 state that the Access, Modification and Attribute Date and Time "shall not be earlier than the File Creation Date and Time …". Because some Operation Systems have a different notion of "Creation Time", UDF changes this ECMA rule from mandatory into a recommendation by reading "*should* not be earlier" instead of "*shall* not be earlier" in ECMA 4/14.9.12-14.

NOTE: ECMA 4/14.9.12-14 only refers to the File Creation Date and Time in a File Times Extended Attribute. However, the File Times EA File Creation Date and Time shall not be recorded for an Extended File Entry. An EFE has its own Creation Date and Time field that shall be used, see 3.3.4.3.1 and ECMA 4/14.17.13-16.

-OSTA-
Optical Storage
Technology Association

UDF 2.60 approved errata

Document:	**OSTA Universal Disk Format**	**DCN-5154**
Subject:	*Requirements for HD DVD Disc*	
Date:	December 5, 2005; last modified June 09, 2006	
Status:	Approved June 12, 2006	

Description:

This DCN is for the next UDF revision after 2.60 **and for the UDF 2.50 and UDF 2.60 errata**.
The High Density DVD (HD DVD) Format for consumer appliances uses UDF 2.50 as the file system for the High Density Read-Only disc (HD DVD-ROM), High Density Rewritable disc (HD DVD-RAM) and The High Density Recordable disc (HD DVD-R for SL/DL).
The purpose of this proposal is to provide enough information for the requirements in the HD DVD Format, and to support good interchangeability between both computer systems and consumer appliances using HD DVD.
The text in this DCN describes the requirements for HD DVD media, so all the HD DVD media that use UDF 2.50.

Change:

Insert a new section 6.z to describe the requirements for HD DVD Disc:

6.z Requirements for HD DVD Disc

This appendix defines the requirements and restrictions on volume and file structures for HD DVD media, including but not limited to HD DVD-ROM discs (6.z.1), HD DVD-RAM discs (6.z.2) and HD DVD-R for SL/DL discs (6.z.3), to support the interchange of information between users of both computer systems and consumer appliances. These requirements do not apply to the discs that are used in a computer system environment only and have no interchangeability with consumer appliances.
The common requirements for these HD DVD discs are summarized as follows:

1. The volume and file structure shall comply with UDF 2.50.
2. The length of logical sector and logical block shall be 2048 bytes.
3. ECC block size is 32 sectors (64 KB).
4. A Main Volume Descriptor Sequence and a Reserve Volume Descriptor Sequence shall be recorded.
5. A HD DVD disc shall have a single volume with a single Partition Descriptor per side.
   Therefore, the volume sequence number shall be 1, the maximum volume sequence number shall be 1 and the Primary Volume Descriptor Interchange Level shall be 2.
6. Only ICB Strategy type 4 shall be used.

6.z.1 Requirements for HD DVD-ROM

The volume and file structure is simplified as for Read-Only discs.

For Volume Structure:

1. A partition on a HD DVD-ROM disc shall be a read-only partition specified as access type 1.
2. One of <u>the</u> Anchor Volume Descriptor Pointers should be recorded in the logical sector 256.
3. The Terminating Descriptor shall be recorded to terminate an extent of a Volume Descriptor Sequence.
4. The Unallocated Space Table and the Unallocated Space Bitmap shall not be recorded.
5. Sparable Partition Map and Sparing Table shall not be recorded.
6. Virtual Partition Map shall not be recorded.
7. Metadata Partition Map, Metadata File and Metadata Mirror File shall be recorded. Metadata Bitmap File shall not be recorded.

For File Structure:

Common requirements for HD DVD shall be applied.

6.z.2 Requirements for HD DVD-RAM

The volume and file structure is simplified as for Overwritable discs using non-sequential recording.

For Volume Structure:

1. A partition on a HD DVD-RAM disc shall be an overwritable partition specified as access type 4.
2. Sparable Partition Map and Sparing Table shall not be recorded.
3. Virtual Partition Map shall not be recorded.
4. Metadata Partition Map, Metadata File and Metadata Bitmap File shall be recorded.

For File Structure:

5. Non-Allocatable Space Stream shall not be recorded.

6.z.3 Requirements for HD DVD-R for SL/DL

The requirements for HD DVD-R for SL/DL discs are under Data updatable structure (VAT) or under HD DVD-ROM compatible structure (read-only partition). The volume and file structure is simplified as for Write-Once discs using sequential recording.
In the case of HD DVD-ROM compatible structure, the requirements are the same as for HD DVD-ROM, refer to 6.z.1. HD DVD-R DL only supports single session.

In the case of Data updatable structure (VAT), following restriction shall be applied.

For Volume Structure:

1. A partition shall be a write-once partition specified as access type 2.
2. The Unallocated Space Table and the Unallocated Space Bitmap shall not be recorded.
3. Sparable Partition Map and Sparing Table shall not be recorded.
4. Only one Open Logical Volume Integrity Descriptor shall be recorded in the Logical Volume Integrity Sequence.
5. Virtual Partition Map shall be recorded. Therefore Metadata Partition Map shall not be recorded.

For File Structure:

6. Only one File Set Descriptor shall be recorded.
7. Non-Allocatable Space Stream shall not be recorded.
8. Virtual Allocation Table and VAT ICB shall be recorded.
9. Metadata File, Metadata Mirror File and Metadata Bitmap File shall not be recorded.

Add a new entry for this DCN to the UDF history table in section 6.17:

Description	**DCN number**	**Updated in UDF Revision**	**Minimum UDF Read Revision**	**Minimum UDF Write Revision**
Requirements for HD DVD Disc	5154	x.yz	2.50	2.50

(x.yz is the next UDF revision after 2.60)

-OSTA®
Optical Storage
Technology Association

Document:	**OSTA Universal Disk Format**	**DCN-5155**
Subject:	*Add recommendations for DVD+R DL and DVD+RW DL*	
Date:	November 29, 2005; last modified March 02, 2006; June 12, 2006: Editorial change referring to DCN-5163.	
Status:	Approved March 02, 2006	

Description:

This DCN is for the next UDF revision after 2.60 **and as errata for all previous UDF revisions**.
The recommendations for DVD+R and DVD+RW must be adapted to the Dual Layer versions that are available now. Further, the current text is improved. For choosing the Sparing Packet Length for UDF revisions 1.50 and 2.00, see also errata DCN-5163.

Change:

Replace the complete appendix 6.13 by:

6.13 Recommendations for DVD+R and DVD+RW Media

DVD+R and DVD+RW single layer and dual layer media require special consideration due to their nature. The following information and guidelines are established to ensure interchange.

- Logical Sector Size is 2048 Bytes
- 2048 Bytes user data transfer for read and write
- ECC Block Size is 32768 bytes (32KB) and the first sector number of an ECC block shall be an integral multiple of 16

Single layer DVD+R and DVD+RW media have a maximum capacity of 4.7 Gbytes. Dual layer DVD+R DL and DVD+RW DL media have a maximum capacity of 8.5 Gbytes. For more detailed information, see the SCSI-3 MMC command set specification and DVD+R and DVD+RW Basic Format Specification documents. For Mount Rainier formatted DVD+MRW media, see appendix 6.14.

Special care must be taken when UDF structures should be recorded ‘physically far apart’, see 2.2.3.2 and 2.2.13.1. For dual layer media, physically far apart is not the same as logically far apart.

6.13.1 Use of UDF on DVD+R media

DVD+R single layer and dual layer media can be used for disc at once, session at once and incremental recording. Multisession is supported. The general rules of appendix 6.11 apply.

6.13.2 Use of UDF on DVD+RW media

DVD+RW single layer and dual layer media are random readable and writable. Where needed, the DVD+RW drive performs Read-Modify-Write cycles to accomplish this. For DVD+RW media, the drive does not perform defect management. DVD+RW media provide the following features:

- Random read and write access
- Background physical formatting
- The Media Type is Overwritable (partition Access Type 4, overwritable)

Multisession is not supported for DVD+RW media.

6.13.2.1 Requirements

- Sparing shall be managed by the host via the Sparable Partition and a Sparing Table
- The sparing Packet Length shall be 16 logical blocks (32 KB, one ECC block). For UDF revisions 1.50 and 2.00, the sparing Packet Length may be 32 logical blocks, see errata DCN-5163.
- Defective packets known at logical format time shall be allocated by the Non-Allocatable Space Stream, see 3.3.7.2

Preparing a blank DVD+RW medium for UDF usage is done by physical formatting and logical formatting. Physical formatting is writing basic physical structures and writing data to all sectors once (de-icing for Read-Only device compatibility). Logical formatting is writing the mandatory basic UDF file system structures, see 6.13.2.3. Physical formatting can be done prior to logical formatting. This is called physical pre-formatting. However this will take much time. DVD+RW offers the possibility of background physical formatting, see 6.13.2.2. Logical formatting, writing of user data and eject and re-insert of the disc can be performed while background physical formatting is in progress. Physical formatting may be followed by a verification pass.

6.13.2.2 Background physical formatting

When background physical formatting is started, some minimal amount of formatting will be performed and then the de-icing operation will continue in background. From that moment, logical formatting and writing of user data can be performed. The disc can be ejected before background formatting has finished. For such an early eject, the background formatting process must be suspended, using a so-called compatibility stop or a quick stop. After a compatibility stop, the medium is compatible with Read-Only devices. For a compatibility stop, the drive must format all non-recorded areas in between recorded areas at the inner side of the disc. This could cause a significant delay in the early eject process. While background formatting is not yet complete, the delay for a compatibility stop can be reduced greatly by temporarily adapting the file system allocation strategy, see 6.13.2.4. When writing is resumed to a medium where background formatting was suspended, the background physical formatting process must be resumed too. Background physical formatting starts at the inner side of the disc. After a compatibility stop, an uninterrupted part at the inner side of the disc is recorded on layer L0 and for the dual layer disc also an equal part at the inner side of the disc on layer L1. These parts on L0 and L1 correspond to two equal portions, one at the beginning and one at the end of the UDF volume space respectively.

6.13.2.3 Logical formatting

Logical formatting is writing the mandatory basic UDF file system structures, such as VRS, AVDP, VDS, FSD, Root Directory and Sparing Tables.
An AVDP shall be recorded at two of the following locations: 256, N-256 and N, where N is the last valid address in the volume space. However, it is recommended to record an AVDP at all three locations, especially for the DVD+RW DL disc, where the regions for recording of the AVDPs are on both layers at the inner side of the disc, so physically not far apart. Allocation for sparing shall occur during the logical formatting process. The sparing allocation may be zero in length. Defective packets known at logical format time shall not be spared using the Sparing Table but added to the Non-Allocatable Space Stream. Not spared defective packets and packets used for a Sparing Table instance or as sparing area shall always be marked as allocated. Inside the UDF partition space, these packets shall be added to the Non-Allocatable Space Stream and consequently be marked as allocated in the partition Space Set, see 2.2.12 and 3.3.7.2. Outside the UDF partition space, these packets shall be marked as allocated by the Unallocated Space Descriptor. The background physical formatting status shall not influence recording of the LVID. At early eject, the LVID shall be recorded in the same way as it will be recorded on Overwritable media that do not support background physical formatting.

6.13.2.4 Restrictions for DVD-ROM compatibility during background formatting

The restrictions mentioned here are only recommended if DVD-ROM compatibility is required at an early eject while background physical formatting is not yet complete. When background physical formatting is complete, DVD-ROM compatibility is a fact and no restrictions are needed. The restrictions all aim at reduction of compatibility stop delay at an early-eject.
The restrictions during background physical formatting are:

- For single layer DVD+RW, only the first AVDP at 256 must be recorded after background physical formatting has been started. The second and third AVDP must be written after background formatting is complete. As long as there is only one AVDP recorded, the file system is in an intermediate state and is not strictly in compliance with ECMA 167. The dual layer DVD+RW DL does not have this restriction, because all AVDPs can be recorded immediately after background formatting has been started. This is possible because of physical formatting on both layers as described above in 6.13.2.2.
- In order to reduce delay in case of a compatibility stop at early eject, the allocation strategy must be restricted as long as background formatting is not yet complete. The lowest logical sector addresses at the beginning of the volume space and for dual layer DVD+RW DL also the highest logical sector addresses at the end of the volume space should be allocated and recorded first in order to reduce compatibility stop delay.

-OSTA-®
Optical Storage
Technology Association

UDF 2.60 approved errata

Document:	**OSTA Universal Disk Format**	**DCN-5156**
Subject:	*Macintosh OS X additions*	
Date:	November 30, 2005; last modified December 05, 2005	
Status:	Approved December 05, 2005	
	Editorial: Small C-code correction, January 11, 2006	

Description:

This DCN is meant for the next UDF revision after 2.60, and as errata for previous UDF revisions.
The changes defined in this DCN refer to the UDF 2.60 specification text. However, most of these changes are also relevant for the appropriate sections in previous UDF specifications starting with UDF 1.02. In UDF 2.50, an OS Class 3 with OS Identifier value 1 was introduced for Macintosh OS X, see 6.3.2. However, all references to "Macintosh" in the text of the UDF specifications 1.02 till 2.60 inclusive are in fact for "Macintosh OS 9 and older" and there are no specific rules for Macintosh OS X yet. This DCN wants to distinguish clearly between Macintosh OS X and Macintosh OS 9/older rules and will add Macintosh OS X specific rules where needed.

Changes:

In section 2.2.6.4 remove 2 occurrences of:

This information is needed by the Macintosh OS.

In the title of 3.3.1.1.1 replace: Macintosh
by: Macintosh OS 9/older, Macintosh OS X

Add at the end of section 3.3.1.1.1:

In Macintosh OS X, additional rules regarding the hidden bit are in section 3.3.4.5.4.2.

In the title of 3.3.2.1.2 replace: Macintosh
by: Macintosh OS 9/older

In the title of 3.3.2.1.3 replace: UNIX
by: UNIX, Macintosh OS X

In section 3.3.3.3, in the title of the "Default Permission Values table"
replace: Mac OS

by: Mac OS 9/older

replace: UNIX & OS/400
by: UNIX, OS/400, Mac OS X

add at the end of NOTE 1:

Under Macintosh OS X, the *attribute* permission shall either be treated in the same way as UNIX, or be specified by the user.

add at the end of NOTE 2:

Under Macintosh OS X, the *delete* permission shall either be treated in the same way as UNIX, or be specified by the user.

In the title of the "Processing Permissions table"
replace: Mac OS
by: Mac OS 9/older

Add a column at the Processing Permissions table with the following values:

Mac OS X
E
E
E
E
E
E
Note 4
Note 4
Note 4
Note 4

In the paragraph before NOTE 3
replace 2 occurrences of: Macintosh
by: Macintosh OS 9/older

add at the end of section 3.3.3.3:

NOTE 4: When processing permissions under Macintosh OS X, if an implementation chooses to treat the *attribute* permission the same way as UNIX, this permission shall be ignored; if an implementation allows the user to set the *attribute* permission, this permission shall be enforced. Similarly, if an implementation chooses to treat the *delete* permission the same way as UNIX, this permission shall be ignored; if an implementation allows the user to set the *delete* permission, this permission shall be enforced.

At the end of section 3.3.4.5.4.2 change: **NOTE:**
by: **NOTE 1:**
and add a second note:

NOTE 2: Macintosh OS X handles the invisible flag of the Finder Info specially. The invisible flag of the Finder Info is the 15th bit of the FdFlags of UDFFInfo for a file, or the 15th bit of the FrFlags of UDFDInfo for a directory.

- After the value of the Finder Info of a file or a directory is read, the value of the hidden bit in the File Characteristics of this file or directory's File Identifier Descriptor (FID) shall be copied to the invisible flag of the Finder Info returned to the application. If this file or directory does not have a Finder Info and the hidden bit in the FID is set, an all-zero Finder Info with only the invisible flag set shall be returned to the application. If more than one FID points to this file, the invisible flag of the Finder Info returned to the application shall be set to the same value as the value of the hidden bit of the last FID that was used to find this file. The on-disk data shall not be modified when reading.

- When updating the Finder Info on the media, the invisible flag of the Finder Info shall be copied to the hidden bit of the FID that points to this file or directory. If more than one FID points to the file, the hidden bit of at least one FID shall be updated according to the invisible flag of the Finder Info. Which FID is updated is up to the implementation.

This rule improves the interoperability of hidden files between Windows and Macintosh OS X so that hidden files on Windows are hidden on Macintosh OS X and vice versa. For files with hard links, the behavior of hidden files is undefined.

In the title and text of section 4.2.2.1.3 replace: Macintosh
by: Macintosh OS 9/older

Add section 4.2.2.1.7:

4.2.2.1.7 Macintosh OS X

Due to the restrictions imposed by the Mac OS X operating system environment, on the *FileIdentifier* associated with a file or a directory the following methodology shall be employed to handle *FileIdentifier(s)* under the above-mentioned operating system environment:

1. FileIdentifier Lookup: Upon request for a “lookup” of a *FileIdentifier*, a case-sensitive comparison shall be performed. If the case-sensitive comparison fails, a case-insensitive comparison may be performed.

2. Validate FileIdentifier: If the *FileIdentifier* is a valid Mac OS X file identifier for the current system interface, then do not apply the following steps.

3. Invalid Characters: A *FileIdentifier* that contains characters considered invalid within a Mac OS X file name shall have them translated into "_" (#005F). Multiple sequential invalid characters shall be translated into a single "_" (#005F) character. Reference appendix 6.7.2 on invalid characters for a complete list.

4. Long FileIdentifier: In the event that the name does not fit into the limit of the current system interface, the new *FileIdentifier* will consist of the first *N* characters of the *FileIdentifier* at this step in the process, where *N* is the largest possible value such that the first *N* characters of the *FileIdentifier* plus 5 characters (the size of the CRC) is valid in the current system interface.

5. FileIdentifier CRC: Since through the above step 3 and/or 4 process character information from the original *FileIdentifier* is lost, the chance of creating a duplicate *FileIdentifier* in the same directory increases. To greatly reduce the chance of having a duplicate *FileIdentifier* the *file name* shall be modified to contain a CRC of the original *FileIdentifier*.

The CRC has 5 characters. It starts with the separator '#', and followed by a 4 digit CS0 Hex representation of the 16-bit CRC of the original CS0 *FileIdentifier*.

If there is a file extension, the new *FileIdentifier* shall be transformed from the following:

(first *N* characters of the *FileIdentifier* obtained after step 3 without the file extension and the '.' before the file extension) '#' (four characters of CRC) '.' (file extension)

Otherwise if there is no file extension, the new *FileIdentifier* shall be transformed from the following:

(first *N* characters of the *FileIdentifier* obtained after step 3) '#' (four characters of CRC)

In both cases, *N* is the largest possible value such that the transformed *FileIdentifier* is valid in the current system interface.

In section 6.3.2
replace: Macintosh OS X and later releases.
by: Macintosh OS X - generic (includes Cheetah, Puma, Jaguar, Panther, Tiger, and later releases based on the same code base).

In the Appendix 6.18 Developers Registration Form,
replace the entry for: ***Macintosh***
by two entries for: ***Mac OS 9 and Mac OS X*** **respectively**

editorial note: This change may be overruled by a complete review of the Developer Registration Form as described in the separate DCN-5162.

In appendix 6.7.2 apply the following diff to the name conversion algorithm:

editorial note: The complete C code of appendix 6.7.2 as a result of this change is in a separate document named dcn-5156-annex.

```
*** nameconv-org.c  Thu Nov 17 13:59:25 2005
--- nameconv.c      Fri Dec  9 10:53:31 2005
***************
*** 21,30 ****
   *   Define WIN_NT
   *   Define MAXLEN = 255
   *
!  * Macintosh:
!  *   Define MAC.
   *   Define MAXLEN = 31.
   *
   * UNIX
   *   Define UNIX.
   *   Define MAXLEN as specified by unix version.
--- 21,34 ----
   *   Define WIN_NT
   *   Define MAXLEN = 255
   *
!  * Macintosh OS 9/older:
!  *   Define MAC9.
   *   Define MAXLEN = 31.
   *
+  * Macintosh OS X:
+  *   Define MACOSX
+  *   Define MAXLEN = 255
+  *
   * UNIX
   *   Define UNIX.
   *   Define MAXLEN as specified by unix version.
***************
```

```
*** 43,49 ****
   * byte needs to be unsigned 8-bit, and unicode_t needs to
   * be unsigned 16-bit.
   */
! typedef unsigned int unicode_t;
  typedef unsigned char byte;

  /*** PROTOTYPES ***/
--- 47,53 ----
   * byte needs to be unsigned 8-bit, and unicode_t needs to
   * be unsigned 16-bit.
   */
! typedef unsigned short unicode_t;
  typedef unsigned char byte;

  /*** PROTOTYPES ***/
***************
*** 54,59 ****
--- 58,82 ----
   * printable under your implementation.
   */
  int UnicodeIsPrint(unicode_t);
+
+ #ifdef MACOSX
+ size_t GetMaxUnicodeLen(unicode_t *name, /* the unicode name  */
+ size_t charcnt, /* number of unicode characters */
+ size_t maxUtf8Len); /* maximum size of the utf-8 buffer in bytes */
+
+
+ /***********************************************************************
+  * this function returns the number of bytes required to encode
+  * bytecnt/2 unicode characters into the encoding required by the
+  * current system.
+  *
+  * For example, in Mac OS X 10.4 (Tiger), this is UTF-8 encoding
+  * normalized to NFD (decomposed) from.
+  *
+  * The implementation of this function is not included in this standard.
+  */
+ size_t UTF8EncodeLength(unicode_t *str, size_t bytecnt, int flag);
+ #endif

  /***********************************************************************
   * Translates a long file name to one using a MAXLEN and an illegal
***************
*** 67,79 ****
  int UDFTransName(
  unicode_t *newName,/*(Output)Translated name. Must be of length MAXLEN*/
  unicode_t *udfName, /* (Input)  Name from UDF volume.*/
! int udfLen,         /* (Input)  Length of UDF Name.  */
  {
     int index, newIndex = 0, needsCRC = FALSE;
!    int extIndex, newExtIndex = 0, hasExt = FALSE;
! #ifdef (OS2 | WIN_95 | WIN_NT)
     int trailIndex = 0;
  #endif
     unsigned short valueCRC;
     unicode_t current;
     const char hexChar[] = "0123456789ABCDEF";
--- 90,106 ----
```

```
  int UDFTransName(
  unicode_t *newName,/*(Output)Translated name. Must be of length MAXLEN*/
  unicode_t *udfName, /* (Input)  Name from UDF volume.*/
! int udfLen)         /* (Input)  Length of UDF Name.  */
  {
     int index, newIndex = 0, needsCRC = FALSE;
!    int extIndex = 0, newExtIndex = 0, hasExt = FALSE;
! #if defined(OS2) || defined(WIN_95) || defined(WIN_NT)
     int trailIndex = 0;
  #endif
+ #ifdef MACOSX
+    int decomposedUtf8len = 0;
+ #endif
+
     unsigned short valueCRC;
     unicode_t current;
     const char hexChar[] = "0123456789ABCDEF";
***************
*** 111,117 ****
           }
        }

! #ifdef (OS2 | WIN_95 | WIN_NT)
        /* Record position of last char which is NOT period or space. */
        else if (current != PERIOD && current != SPACE)
        {
--- 138,144 ----
           }
        }

! #if defined(OS2) || defined(WIN_95) || defined(WIN_NT)
        /* Record position of last char which is NOT period or space. */
        else if (current != PERIOD && current != SPACE)
        {
***************
*** 127,135 ****
        {
           needsCRC = TRUE;
        }
     }

! #ifdef (OS2 | WIN_95 | WIN_NT)
     /* For OS2, 95 & NT, truncate any trailing periods and\or spaces. */
     if (trailIndex != newIndex - 1)
     {
--- 154,168 ----
        {
           needsCRC = TRUE;
        }
+
+ #ifdef MACOSX
+       decomposedUtf8len += UTF8EncodeLength(&current, 2, UTF_DECOMPOSED);
+       if (decomposedUtf8len >= MAXLEN)
+          needsCRC = TRUE;
+ #endif
     }

! #if defined(OS2) || defined(WIN_95) || defined(WIN_NT)
     /* For OS2, 95 & NT, truncate any trailing periods and\or spaces. */
     if (trailIndex != newIndex - 1)
     {
***************
*** 153,159 ****

              if (IsIllegal(current) || !UnicodeIsPrint(current))
              {
-                needsCRC = 1;
                 /* Replace Illegal and non-displayable chars
                  * with underscore.
                  */
--- 186,191 ----
***************
*** 172,177 ****
--- 204,214 ----
           }

           /* Truncate filename to leave room for extension and CRC. */
+ #ifdef MACOSX
+          maxFilenameLen = (MAXLEN - 5) -
+              UTF8EncodeLength(ext, localExtIndex*2, UTF_DECOMPOSED) - 1;
+          newIndex = GetMaxUnicodeLen(newName, newExtIndex, maxFilenameLen);
+ #else
           maxFilenameLen = ((MAXLEN - 5) - localExtIndex - 1);
           if (newIndex > maxFilenameLen)
           {
***************
*** 181,196 ****
           {
              newIndex = newExtIndex;
           }
        }
        else if (newIndex > MAXLEN - 5)
        {
           /*If no extension, make sure to leave room for CRC. */
           newIndex = MAXLEN - 5;
        }
        newName[newIndex++] = CRC_MARK; /* Add mark for CRC. */

        /*Calculate CRC from original filename from FileIdentifier. */
!       valueCRC = unicode_cksum(udfName, udfLen);
        /* Convert 16-bits of CRC to hex characters. */
        newName[newIndex++] = hexChar[(valueCRC & 0xf000) >> 12];
        newName[newIndex++] = hexChar[(valueCRC & 0x0f00) >> 8];
--- 218,238 ----
           {
              newIndex = newExtIndex;
           }
+ #endif
        }
        else if (newIndex > MAXLEN - 5)
        {
           /*If no extension, make sure to leave room for CRC. */
+ #ifdef MACOSX
+          newIndex = GetMaxUnicodeLen(newName, newIndex, MAXLEN - 5);
+ #else
           newIndex = MAXLEN - 5;
+ #endif
        }
        newName[newIndex++] = CRC_MARK; /* Add mark for CRC. */
        /*Calculate CRC from original filename from FileIdentifier. */
!       valueCRC = unicode_cksum((unsigned short *)udfName, udfLen);
        /* Convert 16-bits of CRC to hex characters. */
        newName[newIndex++] = hexChar[(valueCRC & 0xf000) >> 12];
        newName[newIndex++] = hexChar[(valueCRC & 0x0f00) >> 8];
***************
*** 210,216 ****
     return(newIndex);
  }

! #ifdef (OS2 | WIN_95 | WIN_NT)
  /***********************************************************************
   * Decides if a Unicode character matches one of a list
   * of ASCII characters.
--- 252,258 ----
     return(newIndex);
  }

! #if defined(OS2) || defined(WIN_95) || defined(WIN_NT)
  /***********************************************************************
   * Decides if a Unicode character matches one of a list
   * of ASCII characters.
***************
*** 238,244 ****
     }
     return(found);
  }
! #endif /* OS2 */

  /***********************************************************************
   * Decides whether the given character is illegal for a given OS.
--- 280,286 ----
     }
     return(found);
  }
! #endif /* if defined(OS2) || defined(WIN_95) || defined(WIN_NT) */

  /***********************************************************************
   * Decides whether the given character is illegal for a given OS.
***************
*** 249,256 ****
   */
  int IsIllegal(unicode_t ch)
  {
! #ifdef MAC
!    /* Only illegal character on the MAC is the colon. */
     if (ch == 0x003A)
     {
        return(1);
--- 291,298 ----
   */
  int IsIllegal(unicode_t ch)
  {
! #ifdef MAC9
!    /* Only illegal character on the Mac OS 9/older is the colon. */
     if (ch == 0x003A)
     {
        return(1);
***************
*** 259,266 ****
```

```
     {
        return(0);
     }
! #elif defined UNIX
!    /* Illegal UNIX characters are NULL and slash. */
     if (ch == 0x0000 || ch == 0x002F)
     {
        return(1);
--- 301,308 ----
     {
        return(0);
     }
! #elif defined(UNIX) || defined(MACOSX)
!    /* Illegal UNIX and Mac OS X characters are NULL and slash. */
     if (ch == 0x0000 || ch == 0x002F)
     {
        return(1);
***************
*** 269,275 ****
     {
        return(0);
     }
! #elif defined (OS2 | WIN_95 | WIN_NT)
     /* Illegal char's for OS/2 according to WARP toolkit. */
     if (ch < 0x0020 || UnicodeInString("\\/:*?\"<>|", ch))
     {
--- 311,317 ----
     {
        return(0);
     }
! #elif defined(OS2) || defined(WIN_95) || defined(WIN_NT)
     /* Illegal char's for OS/2 according to WARP toolkit. */
     if (ch < 0x0020 || UnicodeInString("\\/:*?\"<>|", ch))
     {
***************
*** 279,283 ****
     {
        return(0);
     }
! #endif
  }
--- 321,356 ----
     {
        return(0);
     }
! #endif
!    return 1; // should never reach here
  }
+
+ #ifdef MACOSX
+
+ /***********************************************************************
+  * given the maximum size of the utf8 buffer, return the number of
+  * unicode characters that can fit in the utf8 buffer
+  */
+ size_t GetMaxUnicodeLen(
+ unicode_t *name, /* the unicode name  */
+ size_t charcnt, /* number of unicode characters */
+ size_t maxUtf8Len) /* maximum size of the utf-8 buffer in bytes */
+ {
```

```
+    size_t len, i;
+
+    len = 0;
+    for (i=0; i<charcnt; i++)
+    {
+       len += UTF8EncodeLength(name++, 2, UTF_DECOMPOSED);
+       if (len > maxUtf8Len)
+          break;
+    }
+    return i;
+ }
+
+ int UnicodeIsPrint(unicode_t ch)
+ {
+    return 1;
+ }
+
+ #endif
```

Document:	**OSTA Universal Disk Format**	**Annex to DCN-5156**
Subject:	*Annex to DCN-5156: Resulting C code of 6.7.2*	
Date:	December 05, 2005	
	Editorial: Small C-code correction, January 11, 2006	

Description:

This document is an annex to DCN-5156.

Resulting C code of appendix 6.7.2 after applying DCN-5156:

```
/***********************************************************************
 * OSTA UDF compliant file name translation routine for OS/2,
 * Windows 95, Windows NT, Macintosh and UNIX.
 * Copyright 1995 Micro Design International, Inc.
 * Written by Jason M. Rinn.
 * Micro Design International gives permission for the free use of the
 * following source code.
 ***********************************************************************
 *
 * To use these routines with different operating systems.
 *
 * OS/2
 *   Define OS2
 *   Define MAXLEN = 254
 *
 * Windows 95
 *   Define WIN_95
 *   Define MAXLEN = 255
 *
 * Windows NT
 *   Define WIN_NT
 *   Define MAXLEN = 255
 *
 * Macintosh OS 9/older:
 *   Define MAC9.
 *   Define MAXLEN = 31.
 *
 * Macintosh OS X:
 *   Define MACOSX
 *   Define MAXLEN = 255
 *
 * UNIX
 *   Define UNIX.
 *   Define MAXLEN as specified by unix version.
 */

#define ILLEGAL_CHAR_MARK 0x005F
#define CRC_MARK          0x0023
#define EXT_SIZE           5
#define TRUE               1
#define FALSE              0
#define PERIOD            0x002E
#define SPACE             0x0020

/***********************************************************************
 * The following two typedef's are to remove compiler dependancies.
 * byte needs to be unsigned 8-bit, and unicode_t needs to
 * be unsigned 16-bit.
 */
```

```
typedef unsigned short unicode_t;
typedef unsigned char byte;

/*** PROTOTYPES ***/
int IsIllegal(unicode_t ch);
unsigned short unicode_cksum(register unsigned short *s, register int n);

/* Define a function or macro which determines if a Unicode character is
 * printable under your implementation.
 */
int UnicodeIsPrint(unicode_t);

#ifdef MACOSX
size_t GetMaxUnicodeLen(unicode_t *name, /* the unicode name  */
size_t charcnt, /* number of unicode characters */
size_t maxUtf8Len); /* maximum size of the utf-8 buffer in bytes */


/***********************************************************************
 * this function returns the number of bytes required to encode
 * bytecnt/2 unicode characters into the encoding required by the
 * current system.
 *
 * For example, in Mac OS X 10.4 (Tiger), this is UTF-8 encoding
 * normalized to NFD (decomposed) from.
 *
 * The implementation of this function is not included in this standard.
 */
size_t UTF8EncodeLength(unicode_t *str, size_t bytecnt, int flag);
#endif

/***********************************************************************
 * Translates a long file name to one using a MAXLEN and an illegal
 * char set in accord with the OSTA requirements.  Assumes the name has
 * already been translated to Unicode.
 *
 * RETURN VALUE
 *
 *    Number of unicode characters in translated name.
 */
int UDFTransName(
unicode_t *newName,/*(Output)Translated name. Must be of length MAXLEN*/
unicode_t *udfName, /* (Input)  Name from UDF volume.*/
int udfLen)         /* (Input)  Length of UDF Name.  */
{
   int index, newIndex = 0, needsCRC = FALSE;
   int extIndex = 0, newExtIndex = 0, hasExt = FALSE;
#if defined(OS2) || defined(WIN_95) || defined(WIN_NT)
   int trailIndex = 0;
#endif
#ifdef MACOSX
   int decomposedUtf8len = 0;
#endif

   unsigned short valueCRC;
   unicode_t current;
   const char hexChar[] = "0123456789ABCDEF";

   for (index = 0; index < udfLen; index++)
   {
      current = udfName[index];

      if (IsIllegal(current) || !UnicodeIsPrint(current))
      {
         needsCRC = TRUE;
        /* Replace Illegal and non-displayable chars with underscore. */
         current = ILLEGAL_CHAR_MARK;
         /* Skip any other illegal or non-displayable characters. */
         while(index+1 < udfLen && (IsIllegal(udfName[index+1])
                     || !UnicodeIsPrint(udfName[index+1])))
         {
            index++;
         }
      }

      /* Record position of extension, if one is found. */
      if (current == PERIOD && (udfLen - index -1) <= EXT_SIZE)
      {
         if (udfLen == index + 1)
         {
            /* A trailing period is NOT an extension. */
            hasExt = FALSE;
         }
         else
         {
            hasExt = TRUE;
            extIndex = index;
            newExtIndex = newIndex;
         }
      }

#if defined(OS2) || defined(WIN_95) || defined(WIN_NT)
      /* Record position of last char which is NOT period or space. */
      else if (current != PERIOD && current != SPACE)
      {
         trailIndex = newIndex;
      }
#endif

      if (newIndex < MAXLEN)
      {
         newName[newIndex++] = current;
      }
      else
      {
         needsCRC = TRUE;
      }

#ifdef MACOSX
      decomposedUtf8len += UTF8EncodeLength(&current, 2, UTF_DECOMPOSED);
      if (decomposedUtf8len >= MAXLEN)
         needsCRC = TRUE;
#endif
   }

#if defined(OS2) || defined(WIN_95) || defined(WIN_NT)
   /* For OS2, 95 & NT, truncate any trailing periods and\or spaces. */
   if (trailIndex != newIndex - 1)
   {
      newIndex = trailIndex + 1;
      needsCRC = TRUE;
      hasExt = FALSE; /* Trailing period does not make an extension. */
   }
#endif

   if (needsCRC)
   {
      unicode_t ext[EXT_SIZE];
      int localExtIndex = 0;
      if (hasExt)
      {
         int maxFilenameLen;
         /* Translate extension, and store it in ext. */
         for(index = 0; index<EXT_SIZE && extIndex + index +1 < udfLen;
              index++ )
         {
            current = udfName[extIndex + index + 1];

            if (IsIllegal(current) || !UnicodeIsPrint(current))
            {
               /* Replace Illegal and non-displayable chars
                * with underscore.
                */
               current = ILLEGAL_CHAR_MARK;
               /* Skip any other illegal or non-displayable
                * characters.
                */
               while(index + 1 < EXT_SIZE
                        && (IsIllegal(udfName[extIndex + index + 2])
                        || !UnicodeIsPrint(udfName[extIndex + index + 2])))
               {
                  index++;
               }
            }
            ext[localExtIndex++] = current;
         }

         /* Truncate filename to leave room for extension and CRC. */
#ifdef MACOSX
         maxFilenameLen = (MAXLEN - 5) -
             UTF8EncodeLength(ext, localExtIndex*2, UTF_DECOMPOSED) - 1;
         newIndex = GetMaxUnicodeLen(newName, newExtIndex, maxFilenameLen);
#else
         maxFilenameLen = ((MAXLEN - 5) - localExtIndex - 1);
         if (newIndex > maxFilenameLen)
         {
            newIndex = maxFilenameLen;
         }
         else
         {
            newIndex = newExtIndex;
         }
#endif
      }
      else if (newIndex > MAXLEN - 5)
      {
         /*If no extension, make sure to leave room for CRC. */
#ifdef MACOSX
         newIndex = GetMaxUnicodeLen(newName, newIndex, MAXLEN - 5);
#else
         newIndex = MAXLEN - 5;
#endif
      }
      newName[newIndex++] = CRC_MARK; /* Add mark for CRC. */

      /*Calculate CRC from original filename from FileIdentifier. */
      valueCRC = unicode_cksum((unsigned short *)udfName, udfLen);
      /* Convert 16-bits of CRC to hex characters. */
      newName[newIndex++] = hexChar[(valueCRC & 0xf000) >> 12];
      newName[newIndex++] = hexChar[(valueCRC & 0x0f00) >> 8];
      newName[newIndex++] = hexChar[(valueCRC & 0x00f0) >> 4];
      newName[newIndex++] = hexChar[(valueCRC & 0x000f)];

      /* Place a translated extension at end, if found. */
      if (hasExt)
      {
         newName[newIndex++] = PERIOD;
         for (index = 0;index < localExtIndex ;index++ )
         {
            newName[newIndex++] = ext[index];
         }
      }
   }
   return(newIndex);
```

```
}

#if defined(OS2) || defined(WIN_95) || defined(WIN_NT)
/***********************************************************************
 * Decides if a Unicode character matches one of a list
 * of ASCII characters.
 * Used by OS2 version of IsIllegal for readability, since all of the
 * illegal characters above 0x0020 are in the ASCII subset of Unicode.
 * Works very similarly to the standard C function strchr().
 *
 * RETURN VALUE
 *
 *    Non-zero if the Unicode character is in the given ASCII string.
 */
int UnicodeInString(
unsigned char *string,  /* (Input) String to search through.   */
unicode_t ch)  /* (Input) Unicode char to search for. */
{
   int found = FALSE;
   while (*string != '\0' && found == FALSE)
   {
      /* These types should compare, since both are unsigned numbers. */
      if (*string == ch)
      {
         found = TRUE;
      }
      string++;
   }
   return(found);
}
#endif /* if defined(OS2) || defined(WIN_95) || defined(WIN_NT) */

/***********************************************************************
 * Decides whether the given character is illegal for a given OS.
 *
 * RETURN VALUE
 *
 *    Non-zero if char is illegal.
 */
int IsIllegal(unicode_t ch)
{
#ifdef MAC9
   /* Only illegal character on the Mac OS 9/older is the colon. */
   if (ch == 0x003A)
   {
      return(1);
   }
   else
   {
      return(0);
   }
#elif defined(UNIX) || defined(MACOSX)
   /* Illegal UNIX and Mac OS X characters are NULL and slash. */
   if (ch == 0x0000 || ch == 0x002F)
   {
      return(1);
   }
   else
   {
      return(0);
   }
#elif defined(OS2) || defined(WIN_95) || defined(WIN_NT)
   /* Illegal char's for OS/2 according to WARP toolkit. */
   if (ch < 0x0020 || UnicodeInString("\\/:*?\"<>|", ch))
   {
      return(1);
   }
   else
   {
      return(0);
   }
#endif
   return 1; // should never reach here
}

#ifdef MACOSX

/***********************************************************************
 * given the maximum size of the utf8 buffer, return the number of
 * unicode characters that can fit in the utf8 buffer
 */
size_t GetMaxUnicodeLen(
unicode_t *name, /* the unicode name  */
size_t charcnt, /* number of unicode characters */
size_t maxUtf8Len) /* maximum size of the utf-8 buffer in bytes */
{
   size_t len, i;

   len = 0;
   for (i=0; i<charcnt; i++)
   {
      len += UTF8EncodeLength(name++, 2, UTF_DECOMPOSED);
      if (len > maxUtf8Len)
         break;
   }
   return i;
}

int UnicodeIsPrint(unicode_t ch)
{
   return 1;
}

#endif
```

Optical Storage Technology Association

Document:	**OSTA Universal Disk Format**	**DCN-5157**
Subject:	*Unicode Version and Unicode Normalization Form*	
Date:	December 6, 2005; last modified December 14, 2005	
Status:	Approved March 02, 2006	

Description:

This DCN is meant for the next UDF revision after 2.60, and as errata for all UDF revisions 1.02 till 2.60 included.
This DCN enables the use of d-characters from newer Unicode Standard versions than strictly defined in UDF section 2.1.1.
Further, Unicode Normalization Form C (NFC), as used by Windows is recommended for recording of d-character identifiers on all UDF media. This also avoids e.g. file identifiers that are 'optically identical' but are not identical for UDF because they are represented in a different normalization form on the medium.
The changes proposed in this DCN are with respect to the current UDF 2.60 text. Note that UDF revisions 1.02 and 1.50 are currently referring to Unicode Standard Version 1.1 opposed to Unicode Standard Version 2.0 as currently for UDF 2.00 and higher revisions. It is now proposed to let all UDF revisions refer to The Unicode Standard 4.0 as a *reference version*.

Change:

In section 2.1.1

Replace:

OSTA CS0 shall consist of the d-characters specified in The Unicode Standard, Version 2.0 (ISBN 0-201-48345-9 from Addison-Wesley Publishing Company http://www.awl.com/, see also http://www.unicode.org/), excluding #FEFF and #FFFE, stored in the *OSTA Compressed Unicode* format which is defined as follows:

by:

OSTA CS0 shall consist of the d-characters specified in The Unicode Standard, excluding the characters #FEFF and #FFFE. The Unicode Standard reference version is Version 4.0 (ISBN 0-321-18578-1 from Addison-Wesley Publishing Company http://www.awl.com/, see also http://www.unicode.org/). Because of the stability policy defined in the Unicode Standard (http://www.unicode.org/standard/stability_policy.html), also older or newer Unicode versions can be used without expecting backward or forward compatibility problems.

To improve interoperability among different platforms, the Unicode d-character identifiers stored on UDF media should be normalized to Normalization Form C (NFC), see Unicode Standard Annex #15 (http://www.unicode.org/unicode/reports/tr15).

NOTE 1: Since Windows uses NFC form, most existing UDF media and UDF implementations on Windows (including those that are not aware of Unicode normalization) already follow this recommendation.
UDF implementations using a different Normalization Form should still write d-character identifiers in NFC form onto the UDF medium and perform conversion to or from that different Normalization Form when needed. An example of this is MAC OS using Normalization Form D (NFD).
Implementations must be aware that normalization conversions of d-character identifiers may increase or decrease the number of Unicode characters of the identifier.

Unicode characters are stored in the *OSTA Compressed Unicode* format, which is defined as follows:

replace (2 occurrences): Unicode 2.0
by: Unicode

replace: **NOTE:**
by: **NOTE 2:**

Document:	**OSTA Universal Disk Format**	**DCN-5159**
Subject:	*Add additional recommendations for BD Read-only Disc*	
Date:	January 24, 2006	
Status:	Approved March 02, 2006	

Description:

This DCN is for the next UDF revision after UDF 2.60 **and for the UDF 2.50 and UDF 2.60 errata**.

The purpose of this DCN is to provide additional information for the BD Read-Only Disc Format to support good interchangeability between both computer systems and consumer appliances using Blu-ray Read-Only Disc.

For BD Read-Only disc with "BDMV Application", there are two types of discs with an ECC Block Size of 64KB or 32KB. Also, "BDMV Application" has a new additional directory immediately under the root directory to certify interactive applications.

Change:

In the second paragraph of section 6.16

replace: • Blu-ray Disc Read-Only Format (BD-ROM)
by: • Blu-ray Disc Read-Only Format (BD-ROM), see note below

and replace: 2. ECC Block Size is 65536 bytes (64KB)
by: 2. ECC Block Size is 65536 bytes (64KB), see note below

Add a following note at the end of section 6.16:

NOTE: There is a Blu-ray Read Only Format with the "BDMV Application" specified on a disc with a capacity of 4.7 Gbytes or 8.5 Gbytes. Its ECC Block Size is 32768 bytes (32KB). All other requirements for this format are the same as for BD-ROM.

In the third paragraph of section 6.16.4 replace:

The "BDMV Application" is a Video Application Format for BD-ROM discs, including AV Stream and database for playback the AV Stream.
The "BDMV" directory immediately under the root directory is reserved for the BDMV application.

_by:_

The ”BDMV Application” is a Video Application Format for BD-ROM discs, including AV Stream and database for playback of the AV Stream. It also supports certification of interactive applications.
The “BDMV” and “CERTIFICATE” directories immediately under the root directory are reserved for the BDMV application.

Document:	**OSTA Universal Disk Format**	**DCN-5160**
Subject:	*More prominent role for Extended File Entry*	
Date:	January 27, 2006; last modified March 16, 2006	
Status:	Approved June 12, 2006	

Description:

This DCN is for the next UDF revision after 2.60 and as clarification for UDF 2.00 and higher revisions.
Since UDF 2.00, the Extended File Entry descriptor should be used instead of the File Entry descriptor, see 3.3.5. However, the sections 2.3.6 and 3.3.3 are only about FE, no trace of EFE and there are no specific EFE sections. The result is that in most cases FE is used by implementations.
Sections 2.3.6 and 3.3.3 are adapted in such a way that it covers both EFE and FE with a more prominent role for EFE. No rule changes in this DCN.
Editorial: Mind that the approved DCN-5153 updates the same sections.

Changes:

Add a new entry for this DCN to the UDF history table in section 6.17:

Description	DCN number	Updated in UDF Revision	Minimum UDF Read Revision	Minimum UDF Write Revision
More prominent role for Extended File Entry	5160	x.yz	2.00	2.00

(editorial: x.yz is the next UDF revision after 2.60)

In 3.3.5.1 replace:

File Entries and Extended File Entries may be freely mixed. In particular, compatibility with old reader implementations can be maintained for certain files.

by:

File Entries and Extended File Entries may be freely mixed. In particular, compatibility with old reader implementations can be maintained for certain files. However, the use of an Extended File Entry instead of a File Entry is recommended, see 3.3.5.

Replace section 2.3.6 by:

2.3.6 Extended File Entry and File Entry

```
struct ExtendedFileEntry {                  /* ECMA 167 4/14.17 and 4/14.9 */
        struct tag          DescriptorTag;
        struct icbtag       ICBTag;
        Uint32              Uid;
        Uint32              Gid;
        Uint32              Permissions;
        Uint16              FileLinkCount;
        Uint8               RecordFormat;
        Uint8               RecordDisplayAttributes;
        Uint32              RecordLength;
        Uint64              InformationLength;
        Uint64              ObjectSize;                 /* EFE only */
        Uint64              LogicalBlocksRecorded;
        struct timestamp    AccessDateAndTime;
        struct timestamp    ModificationDateAndTime;
        struct timestamp    CreationDateAndTime;        /* EFE only */
        struct timestamp    AttributeDateAndTime;
        Uint32              Checkpoint;
        byte                Reserved[4];                /* EFE only */
        struct long_ad      ExtendedAttributeICB;
        struct long_ad      StreamDirectoryICB;         /* EFE only */
        struct EntityID     ImplementationIdentifier;
        Uint64              UniqueID,
        Uint32              LengthofExtendedAttributes;
        Uint32              LengthofAllocationDescriptors;
        byte                ExtendedAttributes[];
        byte                AllocationDescriptors[];
}
```

The total length of an *Extended File Entry* (*EFE)* or *File Entry (FE)* shall not exceed the size of one logical block. It is recommended to use an *EFE* instead of an *FE* for all cases.

An *EFE* is a superset of an *FE*, which means that an *EFE* has all fields of an *FE* with interleaved some extra fields that are marked in the structure above with "**EFE only**". Note that the offsets of identical fields may be different for *EFE* and *FE*. Generally, "*Extended File Entry*" can replace *"File Entry*" throughout the text of this specification.

If a Metadata Partition Map is recorded on a volume, then all (*Extended) File Entries*, Allocation Descriptor Extents and directory data shall be recorded in the Metadata Partition - i.e. in logical blocks allocated to the Metadata and/or Metadata Mirror File.
For details including exceptions see section 2.2.13.

Replace section 3.3.3 by:

3.3.3 Extended File Entry and File Entry

```
struct ExtendedFileEntry {          /* ECMA 167 4/14.17 and 4/14.9 */
        struct tag          DescriptorTag;
        struct icbtag       ICBTag;
        Uint32              Uid;
        Uint32              Gid;
        Uint32              Permissions;
        Uint16              FileLinkCount;
        Uint8               RecordFormat;
        Uint8               RecordDisplayAttributes;
        Uint32              RecordLength;
        Uint64              InformationLength;
        Uint64              ObjectSize;                 /* EFE only */
        Uint64              LogicalBlocksRecorded;
        struct timestamp    AccessDateAndTime;
        struct timestamp    ModificationDateAndTime;
        struct timestamp    CreationDateAndTime;        /* EFE only */
        struct timestamp    AttributeDateAndTime;
        Uint32              Checkpoint;
        byte                Reserved[4];                /* EFE only */
        struct long_ad      ExtendedAttributeICB;
        struct long_ad      StreamDirectoryICB;         /* EFE only */
        struct EntityID     ImplementationIdentifier;
        Uint64              UniqueID,
        Uint32              LengthofExtendedAttributes;
        Uint32              LengthofAllocationDescriptors;
        byte                ExtendedAttributes[];
        byte                AllocationDescriptors[];
}
```

See section 2.3.6 for rules and distinction between Extended File Entry (EFE) and File Entry (FE).

UDF 2.60 approved errata

Document:	**OSTA Universal Disk Format**	**DCN-5161**
Subject:	*Treat Fixed Packets in the same way as ECC Blocks*	
Date:	February 21, 2006; last modified March 27, 2006	
Status:	Approved June 12, 2006	

Description:

This DCN is for the next UDF revision after 2.60 **and as errata for the UDF revisions 1.50 till 2.60 inclusive**.
UDF rules for ECC blocks, like alignment etc., must also apply for fixed packet media like CD-RW. The easiest way to accomplish this is to add a remark to the ECC Block and Fixed Packet definitions. It would e.g. be strange not to align Metadata Partition extents on fixed packet boundaries for CD-RW when there is no Sparable Partition. Further, it seems that it is not clearly defined that the logical sector address of the first sector of a fixed packet must be an integer multiple of the packet length.

Changes:

In 1.3.2 replace:

ECC Block Size (bytes)	This term refers to values defined in relevant device and/or media specifications. The reader should consult the appropriate document – for example, the “MMC” or “Mt. Fuji” specifications for CD/DVD class media. For media exposing no such concept externally (e.g. hard disc) this term shall be interpreted to mean the sector size of the media.
Fixed Packet	An incremental recording method in which all packets in a given track are of a length specified in the Track Descriptor Block. Addresses presented to a CD drive are translated according to the Method 2 addressing specified in Orange Book parts-II and -III.

by:

ECC Block Size (bytes)	This term refers to values defined in relevant device and/or media specifications. The reader should consult the appropriate document – for example, the “MMC” or “Mt. Fuji” specifications for CD/DVD class media. For media exposing no such concept externally (e.g. hard disc) this term shall be interpreted to mean the sector size of the media. Although not strictly the same, media with fixed packets, like CD-RW, also have to apply to the ECC block rules in this specification, where a fixed packet is assumed to be equal to an ECC Block.
Fixed Packet	An incremental recording method in which all packets in a given track are of a length specified in the Track Descriptor Block. Addresses presented to a CD drive are translated according to the Method 2 addressing specified in Orange Book parts-II and -III. On a fixed packet medium with a UDF file system, the packets shall be equal in size for all tracks of the medium. The logical sector address of the first sector of each packet shall be an integer multiple of the number of logical sectors per Fixed Packet. Fixed Packets media must also obey to ECC Block rules, see the ECC Block Size definition above.

In 6.10.2.5 replace:

Note that packets may not be aligned to 32 sector boundaries.

by:

Note that packets and tracks shall be aligned to 32 sector boundaries, see the Fixed Packet definition in 1.3.2.

Add a new entry for this DCN to the UDF history table in section 6.17:

Description	**DCN number**	**Updated in UDF Revision**	**Minimum UDF Read Revision**	**Minimum UDF Write Revision**
Treat Fixed Packets in the same way as ECC Blocks	5161	x.yz	1.50	1.50

(editorial: x.yz is the next UDF revision after UDF 2.60)

Document:	**OSTA Universal Disk Format**	**DCN-5162**
Subject:	*Simplification of UDF Developer Registration Form*	
Date:	February 27, 2006	
Status:	Approved March 02, 2006	

Description:

This DCN is for the next UDF revision after 2.60, all previous UDF revisions and for the "Registered UDF Developers" section of the OSTA UDF web page.
On the December 2005 UDF committee meeting it was decided to simplify the UDF Developer Registration Form on the OSTA UDF web page section "Registered UDF Developers" and in appendix 6.18 of the UDF specification. Instead of detailed choices about the support of UDF revisions, media types and Operating Systems, each developer company is offered a text box where this information can be described in short.

Changes:

Add a new entry for this DCN to the UDF history table in section 6.17:

Description	**DCN number**	**Updated in UDF Revision**	**Minimum UDF Read Revision**	**Minimum UDF Write Revision**
Simplification of UDF Developer Registration Form	5162	x.yz	1.02	1.02

(x.yz is the next UDF revision after 2.60)

Replace the Developer Registration Form on the OSTA UDF web page and in appendix 6.18 by:

OSTA Universal Disk Format Specification
Developer Registration Form

Name: ______________________________

Company: ______________________________

Address: ______________________________

____

City: ______________________________

State/Province: ______________________________

Zip/Postal Code: ______________________________

Country: ______________________________

Phone: ______________ FAX: ______________

Email: ______________________________

Please indicate what value you plan to use as EntityID "*Developer ID" to identify your implementation. Developer ID: "*______________"

The Developer ID should uniquely identify your company as well as your product, see note 2 of section 2.1.5.2 in the latest UDF specification. The Developer ID should not start with "*UDF". The registered developer id can be extended with a suffix containing e.g. version information, as long as the total Developer ID (including "*") does not exceed 23 characters.

Please indicate which UDF revisions you plan to support:

____

____

Please indicate which media types you plan to support:

____

____

Please indicate which Operating Systems you plan to support:

____

____

O Please add my email address to the OSTA UDF email reflector.

O Please send an OSTA Membership kit.

Send completed form to OSTA, see http://www.osta.org/osta/contact.htm