OSTA
Optical Storage Technology Association

UDF Document Change Notice
DCN-5000

Document: OSTA Universal Disk Format DCN#5000
Subject: *Tag Serial Number and disaster recovery.*
Date: July 29, 1999

Description:

It appeared to be unclear how a Tag Serial Number shall be set in order to support disaster recovery. This DCN shall apply to all media types.

Change:

Section 2.2.1.1, on page 16 must be replaced by:

2.2.1.1 Uint16 TagSerialNumber

for read**:** Ignored. Intended for disaster recovery.

for write**:** Shall be set to the *TagSerialNumber* value of the Anchor Volume Descriptor Pointers on this volume.

In order to preserve disaster recovery support, the *TagSerialNumber* must be set to a value that differs from ones previously recorded, upon volume re-initialization. This value is determined at volume formatting time and may depend on the state of the volume prior to formatting. See <formatting_section> for further details.

Section 2.3.1.1, on page 34 must be replaced by:

2.3.1.1 Uint16 TagSerialNumber

for read**:** Ignored. Intended for disaster recovery.

for write**:** Shall be set to the *TagSerialNumber* value of the Anchor Volume Descriptor Pointers on this volume.

The same applies as for volume structure *TagSerialNumber* values, see 2.2.1.1 and < formatting_section>.

Add the following section as **2.1.6** :
(so replace all <formatting_section> references by 2.1.6)

<formatting_section> **Descriptor Tag Serial Number at formatting time.**

In order to support disaster recovery, the *TagSerialNumber* value of all UDF descriptors that will be recorded at formatting time, shall be set to a value that differs from ones previously recorded, upon volume re-initialization.
If no disaster recovery will be supported, a value zero (#0000) shall be used for the TagSerialNumber field of all UDF descriptors that will be recorded at formatting time, see ECMA 3/7.2.5 and 4/7.2.5.
If disaster recovery is supported, the value to be used depends on the state of the volume prior to formatting. There are only two states in which a volume can be formatted such that disaster recovery will be possible in the future. These states are:

1) The volume is completely erased. Only after this action, and where~~If~~ disaster recovery is to be supported then a value of one (#0001) ~~unequal to zero~~ shall be used as the TagSerialNumber value ~~for formatting~~. ~~It is recommended to use a value one (#0001) in this case.~~

2) The volume is a clean UDF volume that supports disaster recovery for TagSerialNumber values, and the TagSerialNumber values of at least two Anchor Volume Descriptor Pointers are both equal to X, where X is not equal to zero. If disaster recovery is to be supported then a value X+1 shall be used as the TagSerialNumber value ~~for formatting~~. If X+1 wraps to zero then keep it as zero to indicate that disaster recovery is not ~~support~~supported.

   NOTE: ~~is only possible if the volume is completely erased first, achieving state 1).~~ The reason for this is that if X+1 wraps to zero then the uniqueness of any TagSerialNumber value unequal to zero can no longer be guaranteed on the volume.

~~In all other cases, the volume has to be completely erased first, achieving state 1), in order to be able to support disaster recovery.~~

NOTE: By ‘erased’ in the above paragraphs, we mean that the sectors are made non-valid for UDF – for example by writing zeroes to the sectors.

Document Change Notice
UDF 2.01 DG-5002

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	Change to the DOS name transform algorithm
Date:	**October 20, 1998**

Description:

The DOS name transform introduced in UDF 1.50 generates insufficiently "implausible" names, and has been demonstrated to conflict with common user specified names in non-trivial situations. This change to use a base41 scheme and re-introduce the # separator character will dramatically lower the probability of conflict in normal operation.

The new routine also permits an implementation to account for native character sets with variable-length characters. Previous specifications for the name translation algorithm assume a character-for-character correspondence between the transformed Unicode output and the native representation of the string used by the operating system. This is not the case for the DOS namespace, where native characters (i.e. the OEM character set) can be one or two bytes in length, but the limit of the total byte length of the native string is fixed. This caused truncation problems when file names contained non-ASCII characters.

A name transform routine to synthesize DOS volume labels has also been added. The rules governing the volume label namespace are sufficiently different from that of the file namespace to necessitate a separate algorithm. This algorithm is not necessarily a drop-in for non-DOS systems.

Additionally, explicit leeway must be granted for an implementation to not use the DOS name transform so that the potential for conflict may be removed at the discretion of the implementor and/or user, recognizing the potential impact on 16-bit applications.

Change:

In "NOTE" two paragraphs before Definitions in section 4.2.2.1:

- Strike "In addition" from the second sentence.

- Add sentence: “In addition, the following algorithms reference “CS0 Base41 representation”, which corresponds to augmenting the CS0 Hex representation to use #0047 - #005A, #0023, #005F, #007E, #002D, and #0040 to represent digits 16-40 respectively.

Change:

In Section 4.2.2.1.1 (MS-DOS), add as the second paragraph:

Exception: Implementations on non-MS-DOS systems that may normally provide dual namespaces (8.3 and non-8.3) using this transformation may omit or provide a mechanism for disabling its use.

Change:

In Section 4.2.2.1.1 (MS-DOS), bullet 9:

- Strike the last two sentences (from the NOTE to the end).
- Substitute from the semi-colon in the last sentence onward: “, followed by the separator ‘#’ (#0023), followed by the CS0 Base41 representation of the 16-bit CRC of the UNICODE expansion of the original filename.”

Change:

In Section 6.7.1, replace code sample with the following:

```
/* OSTA UDF compliant file name translation routine for DOS and    */
/* Windows short namespaces.                                       */

/* Define constants for namespace translation                      */
#define DOS_NAME_LEN        8
#define DOS_EXT_LEN         3
#define DOS_LABEL_LEN      11
#define DOS_CRC_LEN         4
#define DOS_CRC_MODULUS    41

/* Define standard types used in example code.                     */
typedef BOOLEAN int;
typedef short INT16;
typedef unsigned short UINT16;
typedef UINT16 UNICODE_CHAR;

#define FALSE              0
#define TRUE               1

static char crcChar[] =
    "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ#_~-@";

/* FUNCTION PROTOTYPES                                             */
UNICODE_CHAR UnicodeToUpper(UNICODE_CHAR value);
```

```
BOOLEAN IsFileNameCharLegal(UNICODE_CHAR value);
BOOLEAN IsVolumeLabelCharLegal(UNICODE_CHAR value);
INT16 NativeCharLength(UNICODE_CHAR value);
BOOLEAN IsDeviceName(UNICODE_CHAR* name, UINT16 nameLen);

/*****************************************************************/
/* UDFDOSName()                                                  */
/* Translate udfName to dosName using OSTA compliant algorithm.  */
/* dosName must be a Unicode string buffer at least 12 characters*/
/* in length.                                                    */
/*****************************************************************/
UINT16 UDFDOSName(UNICODE_CHAR* dosName, UNICODE_CHAR* udfName,
                  UINT16 udfNameLen)
{
    INT16 index;
    INT16 targetIndex;
    INT16 crcIndex;
    INT16 extLen;
    INT16 nameLen;
    INT16 charLen;
    INT16 overlayBytes;
    INT16 bytesLeft;
    UNICODE_CHAR current;
    BOOLEAN needsCRC;
    UNICODE_CHAR ext[DOS_EXT_LEN];

    needsCRC = FALSE;

    /* Start at the end of the UDF file name and scan for a period */
    /* ('.').  This will be where the DOS extension starts (if     */
    /* any).                                                       */
    index = udfNameLen;
    while (index-- > 0) {
        if (udfName[index] == '.')
            break;
    }

    if (index < 0) {
        /* There name was scanned to the beginning of the buffer   */
        /* and no extension was found.                             */
        extLen = 0;
        nameLen = udfNameLen;
    }
    else {
        /* A DOS extension was found, process it first.            */
        extLen = udfNameLen - index - 1;
        nameLen = index;
        targetIndex = 0;
        bytesLeft = DOS_EXT_LEN;

        while (++index < udfNameLen && bytesLeft > 0) {
            /* Get the current character and convert it to upper   */
            /* case.                                               */
            current = UnicodeToUpper(udfName[index]);
            if (current == ' ') {
                /* If a space is found, a CRC must be appended to  */
                /* the mangled file name.                          */
                needsCRC = TRUE;
            }
            else {
                /* Determine if this is a valid file name char and */
```

```
                /* calculate its corresponding BCS character byte  */
                /* length (zero if the char is not legal or        */
                /* undisplayable on this system).                  */
                charLen = (IsFileNameCharLegal(current)) ?
                    NativeCharLength(current) : 0;

                /* If the char is larger than the available space  */
                /* in the buffer, pretend it is undisplayable.     */
                if (charLen > bytesLeft)
                    charLen = 0;

                if (charLen == 0) {
                    /* Undisplayable or illegal characters are     */
                    /* substituted with an underscore ("_"), and   */
                    /* required a CRC code appended to the mangled */
                    /* file name.                                  */
                    needsCRC = TRUE;
                    charLen = 1;
                    current = '_';

                    /* Skip over any following undiplayable or     */
                    /* illegal chars.                              */
                    while (index + 1 < udfNameLen &&
                        (!IsFileNameCharLegal(udfName[index + 1]) ||
                        NativeCharLength(udfName[index + 1]) == 0))
                        index++;
                }

                /* Assign the resulting char to the next index in  */
                /* the extension buffer and determine how many BCS */
                /* bytes are left.                                 */
                ext[targetIndex++] = current;
                bytesLeft -= charLen;
            }
        }

        /* Save the number of Unicode characters in the extension  */
        extLen = targetIndex;

        /* If the extension was too large, or it was zero length   */
        /* (i.e. the name ended in a period), a CRC code must be   */
        /* appended to the mangled name.                           */
        if (index < udfNameLen || extLen == 0)
            needsCRC = TRUE;
    }

    /* Now process the actual file name.                           */
    index = 0;
    targetIndex = 0;
    crcIndex = 0;
    overlayBytes = -1;
    bytesLeft = DOS_NAME_LEN;
    while (index < nameLen && bytesLeft > 0) {
        /* Get the current character and convert it to upper case. */
        current = UnicodeToUpper(udfName[index]);
        if (current == ' ' || current == '.') {
            /* Spaces and periods are just skipped, a CRC code     */
            /* must be added to the mangled file name.             */
            needsCRC = TRUE;
        }
        else {
```

```
/* Determine if this is a valid file name char and     */
/* calculate its corresponding BCS character byte      */
/* length (zero if the char is not legal or            */
/* undisplayable on this system).                      */
charLen = (IsFileNameCharLegal(current)) ?
    NativeCharLength(current) : 0;

/* If the char is larger than the available space in   */
/* the buffer, pretend it is undisplayable.            */
if (charLen > bytesLeft)
    charLen = 0;

if (charLen == 0) {
    /* Undisplayable or illegal characters are         */
    /* substituted with an underscore ("_"), and       */
    /* required a CRC code appended to the mangled     */
    /* file name.                                      */
    needsCRC = TRUE;
    charLen = 1;
    current = '_';

    /* Skip over any following undiplayable or illegal */
    /* chars.                                          */
    while (index + 1 < nameLen &&
        (!IsFileNameCharLegal(udfName[index + 1]) ||
        NativeCharLength(udfName[index + 1]) == 0))
        index++;

    /* Terminate loop if at the end of the file name.  */
    if (index >= nameLen)
        break;
}

/* Assign the resulting char to the next index in the  */
/* file name buffer and determine how many BCS bytes   */
/* are left.                                           */
dosName[targetIndex++] = current;
bytesLeft -= charLen;

/* This figures out where the CRC code needs to start  */
/* in the file name buffer.                            */
if (bytesLeft >= DOS_CRC_LEN) {
    /* If there is enough space left, just tack it     */
    /* onto the end.                                   */
    crcIndex = targetIndex;
}
else {
    /* If there is not enough space left, the CRC      */
    /* must overlay a character already in the file    */
    /* name buffer.  Once this condition has been      */
    /* met, the value will not change.                 */
    if (overlayBytes < 0) {
        /* Determine the index and save the length of  */
        /* the BCS character that is overlayed.  It    */
        /* is possible that the CRC might overlay      */
        /* half of a two-byte BCS character depending  */
        /* upon how the character boundaries line up.  */
        overlayBytes = (bytesLeft + charLen > DOS_CRC_LEN)
            ? 1 : 0;
        crcIndex = targetIndex - 1;
    }
```

```
            }
        }

        /* Advance to the next character.                          */
        index++;
    }

    /* If the scan did not reach the end of the file name, or the  */
    /* length of the file name is zero, a CRC code is needed.      */
    if (index < nameLen || index == 0)
        needsCRC = TRUE;

    /* If the name has illegal characters or and extension, it     */
    /* is not a DOS device name.                                   */
    if (needsCRC == FALSE && extLen == 0) {
        /* If this is the name of a DOS device, a CRC code should  */
        /* be appended to the file name.                           */
        if (IsDeviceName(udfName, udfNameLen))
            needsCRC = TRUE;
    }

    /* Append the CRC code to the file name, if needed.            */
    if (needsCRC) {
        /* Get the CRC value for the original Unicode string       */
        UINT16 udfCRCValue = CalculateCRC(udfName, udfNameLen);

        /* Determine the character index where the CRC should      */
        /* begin.                                                  */
        targetIndex = crcIndex;

        /* If the character being overlayed is a two-byte BCS      */
        /* character, replace the first byte with an underscore.   */
        if (overlayBytes > 0)
            dosName[targetIndex++] = '_';

        /* Append the encoded CRC value with delimiter.            */
        dosName[targetIndex++] = '#';
        dosName[targetIndex++] =
            crcChar[udfCRCValue / (DOS_CRC_MODULUS * DOS_CRC_MODULUS)];
        udfCRCValue %= DOS_CRC_MODULUS * DOS_CRC_MODULUS;
        dosName[targetIndex++] =
            crcChar[udfCRCValue / DOS_CRC_MODULUS];
        udfCRCValue %= DOS_CRC_MODULUS;
        dosName[targetIndex++] = crcChar[udfCRCValue];
    }

    /* Append the extension, if any.                               */
    if (extLen > 0) {
        /* Tack on a period and each successive byte in the        */
        /* extension buffer.                                       */
        dosName[targetIndex++] = '.';
        for (index = 0; index < extLen; index++)
            dosName[targetIndex++] = ext[index];
    }

    /* Return the length of the resulting Unicode string.          */
    return (UINT16)targetIndex;
}

/*****************************************************************/
/* UDFDOSVolumeLabel()                                           */
```

```
/* Translate udfLabel to dosLabel using OSTA compliant algorithm.  */
/* dosLabel must be a Unicode string buffer at least 11 characters */
/* in length.                                                      */
/*******************************************************************/
UINT16 UDFDOSVolumeLabel(UNICODE_CHAR* dosLabel, UNICODE_CHAR*
                         udfLabel, UINT16 udfLabelLen)
{
    INT16 index;
    INT16 targetIndex;
    INT16 crcIndex;
    INT16 charLen;
    INT16 overlayBytes;
    INT16 bytesLeft;
    UNICODE_CHAR current;
    BOOLEAN needsCRC;

    needsCRC = FALSE;

    /* Scan end of label to see if there are any trailing spaces.  */
    index = udfLabelLen;
    while (index-- > 0) {
        if (udfLabel[index] != ' ')
            break;
    }

    /* If there are trailing spaces, adjust the length of the      */
    /* string to exclude them and indicate that a CRC code is      */
    /* needed.                                                     */
    if (index + 1 != udfLabelLen) {
        udfLabelLen = index + 1;
        needsCRC = TRUE;
    }

    index = 0;
    targetIndex = 0;
    crcIndex = 0;
    overlayBytes = -1;
    bytesLeft = DOS_LABEL_LEN;
    while (index < udfLabelLen && bytesLeft > 0) {
        /* Get the current character and convert it to upper case. */
        current = UnicodeToUpper(udfLabel[index]);
        if (current == '.') {
            /* Periods are just skipped, a CRC code must be added  */
            /* to the mangled file name.                           */
            needsCRC = TRUE;
        }
        else {
            /* Determine if this is a valid file name char and     */
            /* calculate its corresponding BCS character byte      */
            /* length (zero if the char is not legal or            */
            /* undisplayable on this system).                      */
            charLen = (IsVolumeLabelCharLegal(current)) ?
                NativeCharLength(current) : 0;

            /* If the char is larger than the available space in   */
            /* the buffer, pretend it is undisplayable.            */
            if (charLen > bytesLeft)
                charLen = 0;

            if (charLen == 0) {
                /* Undisplayable or illegal characters are         */
```

```
            /* substituted with an underscore ("_"), and       */
            /* required a CRC code appended to the mangled     */
            /* file name.                                      */
            needsCRC = TRUE;
            charLen = 1;
            current = '_';

            /* Skip over any following undiplayable or illegal */
            /* chars.                                          */
            while (index + 1 < udfLabelLen &&
                (!IsVolumeLabelCharLegal(udfLabel[index + 1]) ||
                NativeCharLength(udfLabel[index + 1]) == 0))
                index++;

            /* Terminate loop if at the end of the file name.  */
            if (index >= udfLabelLen)
                break;
        }

        /* Assign the resulting char to the next index in the  */
        /* file name buffer and determine how many BCS bytes   */
        /* are left.                                           */
        dosLabel[targetIndex++] = current;
        bytesLeft -= charLen;

        /* This figures out where the CRC code needs to start  */
        /* in the file name buffer.                            */
        if (bytesLeft >= DOS_CRC_LEN) {
            /* If there is enough space left, just tack it     */
            /* onto the end.                                   */
            crcIndex = targetIndex;
        }
        else {
            /* If there is not enough space left, the CRC      */
            /* must overlay a character already in the file    */
            /* name buffer.  Once this condition has been      */
            /* met, the value will not change.                 */
            if (overlayBytes < 0) {
                /* Determine the index and save the length of  */
                /* the BCS character that is overlayed.  It    */
                /* is possible that the CRC might overlay      */
                /* half of a two-byte BCS character depending  */
                /* upon how the character boundaries line up.  */
                overlayBytes = (bytesLeft + charLen > DOS_CRC_LEN)
                    ? 1 : 0;
                crcIndex = targetIndex - 1;
            }
        }
    }

    /* Advance to the next character.                          */
    index++;
}

/* If the scan did not reach the end of the file name, or the  */
/* length of the file name is zero, a CRC code is needed.      */
if (index < udfLabelLen || index == 0)
    needsCRC = TRUE;

/* Append the CRC code to the file name, if needed.            */
if (needsCRC) {
```

```
        /* Get the CRC value for the original Unicode string       */
        UINT16 udfCRCValue = CalculateCRC(udfName, udfNameLen);

        /* Determine the character index where the CRC should      */
        /* begin.                                                  */
        targetIndex = crcIndex;

        /* If the character being overlayed is a two-byte BCS      */
        /* character, replace the first byte with an underscore.   */
        if (overlayBytes > 0)
            dosLabel[targetIndex++] = '_';

        /* Append the encoded CRC value with delimiter.            */
        dosLabel[targetIndex++] = '#';
        dosLabel[targetIndex++] =
            crcChar[udfCRCValue / (DOS_CRC_MODULUS * DOS_CRC_MODULUS)];
        udfCRCValue %= DOS_CRC_MODULUS * DOS_CRC_MODULUS;
        dosLabel[targetIndex++] =
            crcChar[udfCRCValue / DOS_CRC_MODULUS];
        udfCRCValue %= DOS_CRC_MODULUS;
        dosLabel[targetIndex++] = crcChar[udfCRCValue];
    }

    /* Return the length of the resulting Unicode string.          */
    return (UINT16)targetIndex;
}

/*****************************************************************/
/* UnicodeToUpper()                                              */
/* Convert the given character to upper-case Unicode.            */
/*****************************************************************/
UNICODE_CHAR UnicodeToUpper(UNICODE_CHAR value)
{
    /* Actual implementation will vary to accommodate the target   */
    /* operating system API services.                              */

    /* Just handle the ASCII range for the time being.             */
    return (value >= 'a' && value <= 'z') ?
        value - ('a' - 'A') : value;
}

/*****************************************************************/
/* IsFileNameCharLegal()                                         */
/* Determine if this is a legal file name id character.          */
/*****************************************************************/
BOOLEAN IsFileNameCharLegal(UNICODE_CHAR value)
{
    /* Control characters are illegal.                             */
    if (value < ' ')
        return FALSE;

    /* Test for illegal ASCII characters.                          */
    switch (value) {
        case '\\':
        case '/':
        case ':':
        case '*':
        case '?':
        case '\"':
        case '<':
        case '>':
```

```
        case '|':
        case ';':
        case '^':
        case ',':
        case '&':
        case '+':
        case '=':
        case '[':
        case ']':
            return FALSE;
        default:
            return TRUE;
    }
}

/*****************************************************************/
/* IsVolumeLabelCharLegal()                                      */
/* Determine if this is a legal volume label character.          */
/*****************************************************************/
BOOLEAN IsVolumeLabelCharLegal(UNICODE_CHAR value)
{
    /* Control characters are illegal.                           */
    if (value < ' ')
        return FALSE;

    /* Test for illegal ASCII characters.                        */
    switch (value) {
        case '\\':
        case '/':
        case ':':
        case '*':
        case '?':
        case '\"':
        case '<':
        case '>':
        case '|':
        case '.':
        case ';':
        case '^':
        case ',':
        case '&':
        case '+':
        case '=':
        case '[':
        case ']':
            return FALSE;
        default:
            return TRUE;
    }
}

/*****************************************************************/
/* NativeCharLength()                                            */
/* Determines the corresponding native length (in bytes) of the  */
/* given Unicode character.  Returns zero if the character is    */
/* undisplayable on the current system.                          */
/*****************************************************************/
INT16 NativeCharLength(UNICODE_CHAR value)
{
    /* Actual implementation will vary to accommodate the target */
    /* operating system API services.                            */
```

```
    /* This is an example of a conservative test. A better test    */
    /* will utilize the platform's language/codeset support to     */
    /* determine how wide this character is when converted to the  */
    /* active variable width character set.                        */
    return 1;
}

/*****************************************************************/
/* IsDeviceName()                                                */
/* Determine if the given Unicode string corresponds to a DOS    */
/* device name (e.g. "LPT1", "COM4", etc.).  Since the set of     */
/* valid device names with vary from system to system, and        */
/* a means for determining them might not be readily available,   */
/* this functionality is only suggested as an optional           */
/* implementation enhancement.                                   */
/*****************************************************************/
BOOLEAN IsDeviceName(UNICODE_CHAR* name, UINT16 nameLen)
{
    /* Actual implementation will vary to accommodate the target  */
    /* operating system API services.                             */

    /* Just return FALSE for the time being.                      */
    return FALSE;
}
```

Document Change Notice
UDF 2.01 DG-5004

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	Specify directory search order for dual namespaces
Date:	**October 29, 1999**

Description:

We must suggest a directory search order when generated names are visible alongside the original primary names in a directory to preserve expectations that the "visible" namespace of a directory is searched first.

For instance, the DOS transform results in a non-8.3 and 8.3 namespace existing side-by-side. Normal Win32 operations will see only the non-8.3 space, and expect to be matched against that space before the 8.3 space. It is easy to see that in a one-pass short circuited search it is possible that the match could occur in the 8.3 space that was also available (and expected) later in the non-8.3 space.

Change:

In section 4.2.2.1, immediately before "Definitions", add paragraph:

Some name transformations in section 4.2.2.1 result in two namespaces being visible at once in a given directory – the space of primary names, those which are physically recorded in a directory; and the space of generated names, those which are derived from the primary names. This is distinct from transformations that take an otherwise illegal name and render it into a legal form, the illegal name not being considered part of the namespace of the directory on that system. For UDF implementations using such transforms, the implementation should search a directory in two passes: pass one should match against the primary namespace and pass two should match against the generated namespace. A match in the primary namespace should be preferred to a match against the generated namespace.

Document Change Notice
UDF 2.01 DG-5006

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	Clarification of termination condition for Strategy 4096
Date:	**October 29, 1999**

Description:

The specification of strategy 4096 (UDF 2.00 6.6) is not exhaustively specific in regard to how the ICB hierarchy is to be terminated. While this is permissible since ECMA 167 4/8.10 specifies how ICB hierarchies are built, we have found that with respect to 4096 there exists confusion rooted in incorrect interpretation of ECMA 167 3/8.1.2.2, which specifies the determination of unrecorded sectors.

This clarification is with respect to the general notion of an unrecorded sector in the standard, and is of informational nature.

Change:

Rename existing section 5.4 to 5.5, create new section 5.4 “Clarification of Unrecorded Sectors” containing the following:

ECMA 167 section 3/8.1.2.2 states

> Any unrecorded constituent sector of a logical sector shall be interpreted as containing all #00 bytes. Within the sector containing the last byte of a logical sector, the interpretation of any bytes after that last byte is not specified by this Part.
>
> A logical sector is unrecorded if the standard for recording allows detection that a sector has been unrecorded and all of the logical sector's constituent sectors are unrecorded. A logical sector should either be completely recorded or unrecorded.

For the purposes of interchange, UDF must clarify the correct interpretation of this section.

This part specifies that an unrecorded sector logically contains #00 bytes. However, the converse argument that a sector containing only #00 bytes is unrecorded is not implied, and such a sector is not an “unrecorded” sector for the purposes of ECMA. Only the standard governing the recording of sectors on the store can provide the rule for

determining if a sector is unrecorded. For example, a blank check condition would provide correct determination for a WORM device.

The following additional ECMA 167 sections reference the rule defined 3/8.1.2.2: 3/8.4.2, 3/8.8.2, 4/3.1, 4/8.3.1 and 4/8.10. By derivation, UDF 6.6 (strategy 4096) is also affected. Since unrecorded sectors/blocks are terminating conditions for sequences of descriptors, an implementation must be careful to know that the underlying storage provides a notion of unrecorded sectors before assuming that not writing to a sector is detectable. Otherwise, reliance on the incorrect converse argument mentioned above may result. Explicit termination descriptors must be used when an appropriate unrecorded sector would be undetectable.

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5007

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	*Compression IDs 254 and 255.*
Date:	July 9, 1999 (updated October 29, 1999~~August 31, 1999~~)

Description:

Compression IDs 254 and 255 were introduced to ensure uniqueness of deleted entries in a directory, as required by ECMA-167, 4/8.6. The algorithm proposed in UDF 2.00 does not handle the special case where the FID contains an identifier with length of 1 byte. In general, the proposal and its intended use (which was not described in the standard) does not work and is excessively complex.

In the spirit of the original ECMA 167 work, we will fall back and clarify that deleted FIDs are not part of the pairwise unique space of the directory, and use the new compression IDs to allow implementations that believed that all FIDs were pairwise unique to function.

Change:

In chapter 2.1.1, add following text after the last paragraph (“A Compression ID … is set.”):

Compression IDs 254 and 255 shall only be used in FIDs where the deleted bit is set to ONE.

When uncompressing file identifiers with Compression IDs 254 and 255, the resulting name is to be considered empty and unique.

Change the table “Compression Algorithm”:

Entry 254: new text “Value indicates the CS0 expansion is empty and unique. Compression Algorithm 8 is used for compression.”
Entry 255: new text “Value indicates the CS0 expansion is empty and unique. Compression Algorithm 16 is used for compression.”

Remove the final paragraph “A Compression ID of 254 or 255 shall … when the Deleted bit is set.”

Document Change Notice
DCN-5007

Change:

In chapter 2.3.4.2, replace the two last paragraphs (“No two FIDs …” and “When deleting …”) with the following text:

ECMA 167 4/8.6 requires that the File Identifiers (and File Version Numbers, which shall always be 1) of all FIDs in a directory shall be unique. While the standard is silent on whether FIDs with the deleted bit set are subject to this requirement, the intent is that they are not. FIDs with the deleted bit set are not subject to the uniqueness requirement, as interpreted by UDF.

In order to assist a UDF implementation that may have read the standard without this interpretation, implementations shall follow these rules when a FID’s deleted bit is set:

If the compression ID of the File Identifier is 8, rewrite the compression ID to 254
If the compression ID of the File Identifier is 16, rewrite the compression ID to 255
Leave the remaining bytes of the File Identifier unchanged

In this way a utility wishing to undelete a file or directory can recover the original name by reversing the rewrite of the compression ID.

Note: Implementations should re-use FIDs that have the deleted bit set to one and ICBs set to zero in order to avoid growing the size of the directory unnecessarily.

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5008

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	*Mac Resource Fork clarification.*
Date:	May 23, 1999

Description:
Mac Resource Forks do only exist in data files, not in directories.

Change:
In the table of section 3.3.8, change the text behind “*UDF Macintosh Resource Fork” from “Any file or directory” to “Any file”.

OSTA
Optical Storage Technology Association

Document Change Notice
DCN-5009

Document: OSTA Universal Disk Format Specification
Revision 2.01

Subject: *Requirements for CD Media*
Date: September 27, 1998

Description:

The Multisession CD Specification contains the basic requirements for multisession CD Media.

Change:

Replace 6.10.1.1, first bullet, by the following text:

- Writing shall use Mode 1 or Mode 2 Form 1 sectors. On one disc, either Mode 1 or Mode 2 Form 1 shall be used; a mixture of Mode 1 and Mode 2 Form 1 sectors on one disc is not allowed.
  NOTE: According to the Multisession CD Specification, all data sessions on a disc must be of the same type (Mode 1, or Mode 2 Form 1).

Replace 6.10.2.1, second bullet, by the following text:

- Writing shall be performed using Mode 1 or Mode 2, Form 1 sectors. On one disc, either Mode 1 or Mode 2 Form 1 shall be used.
  NOTE: According to the Multisession CD Specification, all data sessions on a disc must be of the same type (Mode 1, or Mode 2 Form 1).

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5013

Document: OSTA Universal Disk Format Specification
Revision 2.01

Subject: *AVDP Placement*
Date: July 8, 1999

Description:

The UDF 2.00 specification is not clear with respect to the rules for placement of the AVDP on unclosed CD-R media. The proper specification is spread out over sections 2.2.3, 6.10.1 and 6.10.1.1. Also it is not immediately clear from the standard what to do in case there is a single AVDP at sector 256 *and* 512.

Change:

Change the note under the AVDP placement rules in section 2.2.3 into:

Note: As specified in section 6.10, unclosed CD-R media may have a single AVDP present at either sector 256 or 512. If on an unclosed disc a single AVDP is recorded on sector 256, any AVDP recorded on sector 512 must be ignored. Closed CD-R media shall conform to the above rules.

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5014

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	*Non-relocatable Bit clarification.*
Date:	May 23, 1999 (updated August 31, 1999)

Description:
In respect to future UDF revisions and content creators, the use of the Non-relocatable Bit in the ICB Tab needs to be specified more clearly.

Change:
In 2.3.5.4, replace the text at **Bit 4**, with the following text:

(read)
For OSTA UDF compliant media this bit shall indicate (ONE) ~~that~~ if the file is non-relocatable. If ONE, an implementation shall set the bit to ZERO if a modification will contravene the definition of this bit in ECMA 167-4/14.6.8

(write)
Should be set to ZERO unless required.

NOTE: This flag is **not** a lock on the file in any way. It is used to indicate that an implementation has arranged the allocation of the file to satisfy specific application requirements. In these cases, any remapping of a written block (see UDF sparable partitions) or defragmentation of the file might not be desired. If a file with this flag set to ONE is copied, then the new copy of the file should have this bit set to ZERO.

OSTA
Optical Storage Technology Association

Document Change Notice
~~XXXX~~DCN-5015

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	*Editorial corrections to several sections.*
Date:	May 24, 1999

Description:
Editorial corrections to several sections.

Change:
In the table of section 2., in the table entry for the File Set Descriptor, remove the sentences: “The *File Set Identifier* field contains a name that may be used as an alias name for identifying the Logical Volume to the user. See 2.3.2.7 for further details.”

Change:
In sections 3.3.4.5.1.1 and 3.3.4.6.1, change the Length in the table from “IU_L-1” to IU_L-2”.

Change:
In section 4.1, change the referenced section from “4.1.2.1” to “4.2.2.1”.

Document Change Notice
UDF 2.01 DCN-5018

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	Power Calibration Stream Fix
Date:	**October 20, 1998**

Description:

In section 3.3.7.3.1 the offset of the Power Calibration Table Records is given as 56 bytes. This is a typographic error. The offset should be 36 bytes.

Change:

Change the offset of the power calibration table records in the table in 3.3.7.3.1 to byte 36.

Document Change Notice
UDF 2.01 DCN-5019

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	**Parent of System Stream Directory**
Date:	**July 9, 1999**

Description:

ECMA 167 3rd Edition states in section 4/8.6, directory schema: "there shall be exactly one File Identifier Descriptor identifying the parent directory".

It is not clear whether the system stream directory in UDF 2.00 is considered a directory and, if it is, what its parent should be.

Change:

Add to section 3.3.5, Named Streams:

"The parent of the system stream directory shall be the system stream directory".

Add to section 3.3.5.1 Named Streams Restrictions:

Add to the 3rd restriction to make:

"….stream directories]. The sole exception is that the parent of the system stream directory shall be the system stream directory"

Add to the 9th restriction to make:

"….of the Extended File Entry. The sole exception is that the parent of the system stream directory shall be the system stream directory"

OSTA Optical Storage Technology Association

Document Change Notice
DCN-5020

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	IBM OS/400 UDF Information
DCN#:	*5020*
Date:	December 6, 1998 (Revised February 8, 2000)

This section includes the OS/400 specific information that would be contained in the OS specific areas of the UDF specification. The areas identified are based on current areas where other OS specific requirements are defined. References are to the UDF 2.0 specification.

Contents

OS/400 identification information

This section describes OS/400 specific information that is written in UDF structures where required. It includes implementation identification, operating system identification, and OS/400 specific names for implementation use extended attributes.

OS/400 specific information	
Name	**Description**
Developer Id (written in Identifier field of Implementation ID - UDF 2.1.5.2)	IBM OS/400 UDF (#49, #42, #4d, #20, #4f, #53, #2f, #34, #30, #30, #20, #55, #44, #46).
EA Identifier (written in Identifier field of EA Implementation ID - UDF 2.1.5.2)	*UDF OS/400 DirInfo (#2A, #55, #44, #46, #20, #4F, #53, #2F, #34, #30, #30, #20, #44, #69, #72, #49, #6E, #66, #6F). Needs to be added to UDF 6.1 and UDF 6.2
OS Class (written in Identifier suffix of Implementation ID - UDF 2.1.5.3)	0x07 - Assigned by OSTA (Arnold Jones 8/17/98). A new OS Class for OS/400 needs to be created in addition to DOS, OS/2, Macintosh, UNIX, Windows 9x, Windows NT, etc. Needs to be added to UDF 6.3.
OS Id (written in Identifier suffix of Implementation ID - UDF 2.1.5.3)	0x00 - Assigned by OSTA (Arnold Jones 8/17/98). An OS Identifier under the OS/400 OS Class -- only one OS/400 implementation, would be labelled "OS/400". Needs to be added to UDF 6.3.

OS/400 specific field requirements

This section outlines any OS/400 specific requirements for UDF fields. It corresponds with information that would be documented in Section 3 (System Dependent Requirements) of the UDF specification.

File Identifier Descriptor (UDF 3.3.1)

- **3.3.1.1 File Characteristics** - Handled the same as UNIX (i.e. the 3.3.1.1.2 heading could be changed to say UNIX, OS/400)

ICB Tag (UDF 3.3.2)

- **3.3.2.1 Flags** (may need to add a 3.3.2.1.x for OS/400)
    - Bits 6 and 7 (Setuid& Setgid), Bit 8 (Sticky) - handled same as MS-DOS, OS/2, Windows 95, Windows NT
    - Bit 10 (System) - handled same as UNIX

File Entry (UDF 3.3.3)

- **3.3.3.3 Permissions**- OS/400 setting and interpretation of permissions bits is defined in the following tables (could just add a column to existing table for OS/400). Note change to Note 1 to add OS/400.

OS/400 Setting of Permissions Bits (UDF 3.3.3.3)

The Permissions bits will be set as described in the table below by OS/400. "U" implies that the bit is user specified (controlled by user interface). This table is an extension of the one recorded in the UDF specification for all Operating Systems.

Permission	File/Directory	Description	OS/400
Read	File	The file may be read	U
Read	Directory	The directory may be read	U
Write	File	The file's contents may be modified	U
Write	Directory	Files or subdirectories may be created, deleted, or renamed	U
Execute	File	The file may be executed	U
Execute	Directory	The directory may be searched for a specific file or subdirectory	U
Attribute	File	The file's permissions may be changed	Note 1
Attribute	Directory	The directory's permissions may be changed.	Note 1
Delete	File	The file may be deleted	Note 2
Delete	Directory	The directory may be deleted	Note 2

Note 1: Under UNIX only the owner of a file/directory may change its attributes. Under OS/400 if a file or directory is marked as writable (Write permission set) then the Attribute permission bit should be set.
Note 2: The Delete permission bit should be set based upon the status of the Write permission bit.

I.

OS/400 Handling of Permissions Bits (UDF 3.3.3.3)			
OS/400 handling (enforcement vs. ignore) of Permissions bits will be as described in the table below. This table is an extension of the one recorded in the UDF specification for all Operating Systems.			
Permission	**File/Directory**	**Description**	**OS/400**
Read	File	The file may be read	Enforce
Read	Directory	The directory may be read	Enforce
Write	File	The file's contents may be modified	Enforce
Write	Directory	Files or subdirectories may be created, deleted, or renamed	Enforce
Execute	File	The file may be executed	Ignore
Execute	Directory	The directory may be searched for a specific file or subdirectory	Enforce
Attribute	File	The file's permissions may be changed	Ignore
Attribute	Directory	The directory's permissions may be changed.	Ignore
Delete	File	The file may be deleted	Ignore
Delete	Directory	The directory may be deleted	Ignore

Implementation Use Extended Attribute (UDF 3.3.4.5)

- **3.3.4.5.6 OS/400** Section for OS/400 Implementation Use Extended Attributes
- **3.3.4.5.6.1 OS400DirInfo** This attribute specifies the OS/400 extended directory information. Since this value needs to be reported back to OS/400 for normal directory information processing, for performance reasons it should be recorded in the ExtendedAttributes field of the FileEntry. This extended attribute shall be stored as an Implementation Use Extended Attribute whose ImplementationIdentifier shall be set to "*UDF OS/400 DirInfo".

***OS400DirInfo* format**

RBP	Length	Name	Contents
0	2	Header Checksum	Uint16
2	2	Reserved for padding	Uint16 = 0
4	44	DirectoryInfo	bytes

For complete information on the structure of the *DirectoryInfo* field recorded in the *OS400DirInfo* format, refer to the following IBM document:

IBM OS/400 UDF Implementation
Optical Storage Solutions, Department HTT
IBM
Rochester, Minnesota

File Identifier Translation Algorithm (UDF 4.2.2.1.6)

OS/400

Due to the restrictions imposed by OS/400 operating system environments, on the *FileIdentifier*

associated with a file 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 may be performed. If the case-sensitive comparison is not done or if it fails, a case-insensitive comparision shall be performed.

2. Validate *FileIdentifier*: If the *FileIdentifier* is a valid file identifier for OS/400 then do not apply the following steps.

3. Invalid Characters: A *FileIdentifier* that contains characters considered invalid within an OS/400 file name, or not displayable in the current environment shall have them translated into "_" (#005F). Multiple sequential invalid or non-displayable characters shall be translated into a single "_" (#005F) character.

4. Trailing Spaces: All trailing " "(#0020) shall be removed.

5. FileIdentifier CRC: Since through the above 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 filename shall be modified to contain a CRC of the original *FileIdentifier*.

   If there is a file extension then the new *FileIdentifier* shall be composed of up to the first (255 - (length of (new file extension) + 1 (for the '.')) - 5 (for the #CRC)) characters constituting the file name at this step in the process, followed by the separator "#" (#0023); followed by a 4 digit CS0 Hex representation of the 16 -bit CRC of the original CS0 *FileIdentifier*, followed by "." (#002E) and the file extension at this step in the process.

   Otherwise if there is no file extension the new *FileIdentifier* shall be composed of up to the first (255 - 5 (for the new #CRC)) characters constituting the file name at this step in the process. Followed by the separator "#" (#0023); followed by a 4 digit CS0 hex representation of the 16-bit CRC of the original CS0 *FileIdentifier*.

Note: Invalid characters for OS/400 are only the forward slash "/" (#002F) character. Non-displayable characters for OS/400 are any characters that do not translate to code page 500 (EBCDIC Multilingual).

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-50XX

Document: OSTA Universal Disk Format Specification
Revision 2.01

Subject: *MissingEntityID Suffixes*
Date: January 21, 2000

Description:

Not for all EntityIDs that are used and/or mentioned in the UDF standard proper suffixes have been defined.

Change:

Change in section 2.1.5:

UDF classifies *Entity Identifiers* into 4 separate types as follows:

- *Domain Entity Identifiers*
- *UDF Entity Identifiers*
- *Implementation Entity Identifiers*
- *Application Entity Identifiers*

Remove the following entries from table 2.1.5.2:

Entity Identifiers

Descriptor	Field	ID Value	Suffix Type
UDF Extended Attribute	Implementation ID	*See Appendix*	UDF Identifier Suffix
Non-UDF Extended Attribute	Implementation ID	*"*Developer ID"*	Implementation Identifier Suffix

Add the following entries to table 2.1.5.2:

Entity Identifiers

Descriptor	Field	ID Value	Suffix Type
Primary Volume Descriptor	Application Identifier	*"*Application ID"*	Application Identifier Suffix
Partition Descriptor	Partition Contents	*"+NSR03"*	Application Identifier Suffix
UDF Implementation Use Extended Attribute	Implementation Identifier	*See 3.3.4.5*	UDF Identifier Suffix
Non-UDF Implementation Use Extended Attribute	Implementation Identifier	*"*Developer ID"*	Implementation Identifier Suffix
UDF Application Use Extended Attribute	Application Identifier	*See 3.3.4.6*	UDF Identifier Suffix
Non-UDF Application Use Extended Attribute	Application Identifier	*"*Application ID"*	Application Identifier Suffix
UDF Unique ID Mapping Data	Implementation Identifier	*"*Developer ID"*	Implementation Identifier Suffix
Power Calibration Table Stream	Implementation Identifier	*"*Developer ID"*	Implementation Identifier Suffix

Add following note below the table:

In the *ID Value* column in the above table *"*Application ID"* refers to an identifier that uniquely identifies the writers application.

Add in section 2.1.5.3 follows:

For an *Application Entity Identifier* not defined by UDF, the *IdentifierSuffix* field shall be constructed as follows, unless specified otherwise.

Application *IdentifierSuffix*

RBP	Length	Name	Contents
0	8	Implementation Use Area	bytes

OSTA
Optical Storage Technology Association

Document Change Notice
DCN-5024

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	*Editorial corrections to several sections.*
Date:	July 8th 1999 (updated August 31, 1999)

Description:

Editorial corrections to several sections.

1.

now:

2.2.7.1 EntityID Implementation Identifier

This field shall specify "*UDF LV Info".

should be:

2.2.7.1 EntityID ImplementationIdentifier *(no space)*

The Identifier field of this EntityID shall specify "*UDF LV Info". Refer to the section on Entity Identifier.

.

now:

2.2.7.2 bytes Implementation Use

...

struct EntityID ImplementionID,

should be:

2.2.7.2 bytes ImplementationUse *(no space)*

...

struct EntityID ImplementationID, *(+ta)*

.

now:

2.2.7.2.4 struct EntityID ImplementionID

should be:

2.2.7.2.4 struct EntityID ImplementationID *(+ta)*

.

now:

In section 2. Basic Restriction & Requirements:
artition escritor: artition ccess e o

should be:

artition escritor: artition escritor ccess e o

.

problem:

2.3.10.1 struct ADImpUse definition printed a tiny font

solution:

2.3.10.1 font of struct ADImpUse definition should be as the font of struct long_ad

.

problem:

In 2.3.4.3 struct long_ad ICB, the definition of the Implementation Use bytes conflicts with the general definition of a long_ad in 2.3.10.1. The 2 bytes "Reserved = #00" conflict with the flags field of ADImpUse in 2.3.10.1.

solution:

Change in section 2.3.4.3 the name Reserved to the name Flags as in section 2.3.10.1

.

now:

2.2.4.3 struct EntityID DomainIdentifier
… Refer to 2.1.4.3.

should be:

2.2.4.3 struct EntityID DomainIdentifier
… Refer to 2.1.5.3.

.

In ECMA167-R3, 4/14.1.19, the File Set Descriptor's Reserved field is reported at BP464. It should correctly be at BP480. –

.

There is a textual error in UDF 2.00:

2.3.6.3 Uint8 RecordLength;
must be:
2.3.6.3 Uint32 RecordLength;

Document Change Notice
UDF 2.01 DG-5025-1

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	**New OS Classes and Identifiers**
Date:	**July 8, 1999, Revised December 1999**

Description:

OSTA should update OS class and identifier specifications for new operating systems. Parallel to the way we classify UNIX systems, use generic identifiers for the codebases involved in Windows systems.

Change:

In section 6.3, add:

Operating System Class: “Windows CE” as type 9, “BeOS” as type 8.

Operating System Identifier: “Windows CE – generic” as id 9/0, “BeOS – generic” as id 8/0

Change text for id 5/0 from “Windows 95” to “Windows 9x – generic (includes Windows 98)

Change text for id 6/0 from “Windows NT” to “Windows NT – generic (includes Windows 2000)

Change text for id 3/0 from “Macintosh OS System 7” to “Macintosh OS”

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5026

Document: OSTA Universal Disk Format Specification
Revision 2.01

Subject: *Explanation of Application Identifier field for the PVD*
Date: July 8, 1999, Revised February 3, 2000

Description:

The ECMA-167 definition of the Application Identifier field (BP 344) for the Primary Volume Descriptor seems to be a bit ambiguous or unclear to some people.

Change:

Insert as section 2.2.2.9 into the UDF specification the following text:

2.2.2.9 struct EntityID ApplicationIdentifier

- This field either specifies a valid Entity Identifier (section 2.1.5) identifying the implementation that last wrote this field, or the field is filled with all #00 bytes, meaning that no application is identified.

- Either all #00 bytes or a valid Entity Identifier (section 2.1.5) shall be recorded in this field.

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5027

Document: OSTA Universal Disk Format Specification
Revision 2.01

Subject: *Descriptor CRC Length*
Date: May 24, 1999

Description:

It is easy to misinterpret the UDF standard with respect to the Descriptor CRC Length (2.2.1.2) being equal to the length of the descriptor. Not in all cases the length of the descriptor matches the Descriptor CRC Length.

Change:

Add the following line to section 2.2.1.2:

Note: The Descriptor CRC Length field must not be used to determine the actual length of the descriptor or the number of bytes to read. These lengths do not match in all cases; there are exceptions in the standard where the Descriptor CRC Length need not match the length of the descriptor.

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5029

Document: OSTA Universal Disk Format Specification
Revision 2.00

Subject: *Correction for processing permissions.*

Date: July 9, 1999

Description:

The Attribute and Delete permissions should be changed from Enforce to Ignore for UNIX.

Change:

In section 3.3.3.3, replace

Attribute	directory	The file's permissions may be changed.	E	E	E	E	E	E
Attribute	directory	The directory's permissions may be changed.	E	E	E	E	E	E
Delete	file	The file may be deleted.	E	E	E	E	E	E
Delete	directory	The directory may be deleted.	E	E	E	E	E	E

With

Attribute	directory	The file's permissions may be changed.	E	E	E	E	E	I
Attribute	directory	The directory's permissions may be changed.	E	E	E	E	E	I
Delete	file	The file may be deleted.	E	E	E	E	E	I
Delete	directory	The directory may be deleted.	E	E	E	E	E	I

OSTA
Optical Storage
Technology Association

Document Change Notice
~~2-047~~DCN-5030

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	*Correction and clarification of 3.2.1.1.*
Date:	December 18, 1998

Description:

Correction and clarification of 3.2.1.1.

Change:

Part of section 3.2.1.1, on page 49 .
The references to 2.3.4.2 are wrong and "long_ad" should be more specific, so (each 2 times) replace :
"2.3.4.2" by "2.3.4.3"
and "long_ad" by "ICB field"
resulting in:

When a file or directory is created, this UniqueID is assigned to the UniqueID field of the File Entry/Extended File Entry, the lower 32-bits of UniqueID are assigned to UDFUniqueID in the Implementation Use bytes of the ICB field in the File Identifier Descriptor (see 2.3.4.3), and UniqueID is incremented by the policy described above.

When a name is linked to an existing file or directory, the lower 32-bits of NextUniqueID are assigned to UDFUniqueID in the Implementation Use bytes of the ICB field in the File Identifier Descriptor (see 2.3.4.3), and UniqueID is incremented by the policy described above.

OSTA
Optical Storage
Technology Association

Document Change Notice
dg-5031

Document: OSTA Universal Disk Format Specification
Revision 2.01

Subject: *Volume Recognition Sequence*
Date: July 1, 1999 (Update: September 2, 1999)

Description:

1) UDF (ECMA) does not specify number of Extended Area's in the VRS;
2) UDF (ECMA) does not specify that the place where to record the NSR0* and the BOOT* descriptors;
3) UDF (ECMA) does not put a limit on the number of NSR descriptors present;
4) UDF (ECMA) does not specify where the other descriptors can be located in the VRS;
5) UDF does not specify what should be the first block after VRS.

Change:

Add the following section to Chapter 2, just after the new section 2.1.6 on formatting:

2.1.7 Volume Recognition Sequence

The following rules shall apply when writing the volume recognition sequence:

(when writing)
The Volume Recognition Sequence as described in part 2 and part 3 of ECMA 167 shall be recorded. There shall be exactly one NSR descriptor in the VRS. The NSR and BOOT2 descriptors shall be in the Extended Area. There shall be only one Extended Area with one BEA01 and one TEA01. All other VSDs are only allowed before the Extended Area. The block after the VRS shall be unrecorded or contain all #00.

(when reading)
Implementers should expect that disks recorded by UDF 2.00 and earlier did not have this constraint, and should handle these cases accordingly.

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5032

Document: OSTA Universal Disk Format Specification
Revision 2.01

Subject: *Path Length*
Date: July 1, 1999

Description:

UDF 2.00 (also ECMA) does not properly define Path Size, however this is used in definitions in the standard (e.g. 2. Basic Restrictions & Requirements)

We are removing this because the pathsize is not practically measurable by an implementation, and because the 1023 value does not reflect any real application or OS limit.

Change:

Remove the following line in section 2. Basic Restrictions & Requirements:

Item	Restrictions & Requirements
Maximum Pathsize	Maximum of 1023 bytes

OSTA
Optical Storage Technology Association

UDF Document Change Notice
DCN-5034

Document: OSTA Universal Disk Format DCN-5034
Subject: *FID LengthofImplementationUse and CRC length*.
Date: August 3, 1999

Description:

Clarify obscured effects of 2.3.4.4 and 2.3.4.5 on each other and drop exception for FID CRC length.

Change:

Replace section 2.3.4.4 by:

2.3.4.4 Uint16 LengthofImplementationUse

for read : Shall specify the length of the ImplementationUse field.
for write: Shall specify the length of the ImplementationUse field. This field may contain zero, indicating that the ImplementationUse field has not been used. Otherwise, this field shall contain at least 32 as required by 2.3.4.5.

When writing a File Identifier Descriptor to write-once media, to ensure that the Descriptor Tag field of the next FID will never span a block boundary, if there are less than 16 bytes remaining in the current block after the FID, the length of the FID shall be increased (using the Implementation Use field) enough to prevent this. Remember that in the latter case, the Implementation Use field shall be at least 32 bytes.

OSTA
Optical Storage Technology Association

UDF Document Change Notice
DCN-5035

Document: OSTA Universal Disk Format DCN#5035
Subject: *Non-Allocatable Space Stream.*
Date: September 20, 1999

Description:

As a left-over from UDF 1.5, the UDF 2.0 specification contains in a few places "Non-Allocatable Space List" where "Non-Allocatable Space Stream" should have been used. This DCN applies to CD-RW media only.

Change:

Section 2.2.11, first paragraph on page 32:

Replace "shall be included in the Non-Allocatable Space List" into "shall be included in the Non-Allocatable Space Stream"

Section 2.2.11, first paragraph on page 33:

Replace "shall be part of the Non-Allocatable Space list" into "shall be part of the Non-Allocatable Space Stream"

Section 6.10.2.1, 3rd bullet on page 116:

Replace "using a Non-Allocatable Space List (see 3.3.7.1.2)" by "using a Non-Allocatable Space Stream (see 3.3.7.2)".

Section 6.10.2.2:

Replace "shall be enumerated in the Non-Allocatable Space List (see 3.3.7.1.2)" by "shall be enumerated in the Non-Allocatable Space Stream (see 3.3.7.2)".

OSTA Optical Storage Technology Association

UDF Document Change Notice
DCN-5036

Document: OSTA Universal Disk Format DCN-5036
Subject: *Allocation Extent Descriptor, UDF 2.3.11.*
Date: January 21, 2000 (replacing November 25, 1999)

Description:

It is a bit strange that according to ECMA 167, 4/14.5 the Allocation Descriptors field is no part of the Allocation Extent Descriptor as e.g. for the File Entry and for the Unallocated Space Entry. The most important effect is that the CRC is not calculated over the Allocation Descriptors as for the other descriptor mentioned above. It would be quite logical to do that, and some UDF implementations do that in fact. Strictly the CRC Length extents outside the descriptor which is illegal normally.
This DCN explains how UDF interprets ECMA 167, 4/14.5 regarding to the exact position of the Allocation Descriptors and improves the integrity of the Allocation Descriptors by giving the option of extending the DescriptorCRCLength of the Allocation Extent Descriptor to include the Allocation Descriptors. The fact that this is an option is because of backwards compatibility with former UDF revisions.

Change:

Replace section 2.3.11 and 2.3.11.1 by:

2.3.11 Allocation Extent Descriptor

```
struct AllocationExtentDescriptor {        /* ECMA 167 4/14.5 */
        struct tag      DescriptorTag;
        Uint32          PreviousAllocationExtentLocation;
        Uint32          LengthOfAllocationDescriptors;
}
```

The Allocation Extent Descriptor does not contain the Allocation Descriptors itself. UDF will interpret ECMA 167, 4/14.5 in such a way that the Allocation Descriptors will start on the first byte following the LengthOfAllocationDescriptors field of the Allocation Extent Descriptor. The Allocation Extent Descriptor together with its Allocation Descriptors constitute an extent of allocation descriptors. The length of an extent of allocation descriptors shall not exceed the logical block size.

Unused bytes following the Allocation Descriptors till the end of the logical block shall have a value #00.

2.3.11.1 struct tag DescriptorTag

The DescriptorCRCLength of the DescriptorTag should include the Allocation Descriptors following the Allocation Extent Descriptor. The DescriptorCRCLength value shall be either 8 or 8 + LengthOfAllocationDescriptors.

In **2. Basic Restrictions & Requirements**

on page 8 replace:

Allocation Extent Descriptors	| The length of any single Allocation | Extent Descriptor shall not exceed the | Logical Block Size.

by:

Allocation Extent Descriptors	| The length of any single extent of | allocation descriptors shall not exceed | the Logical Block Size.

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5037

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	*File Types of 248 to 255*
Date:	September 24, 1999, Revised December 6 1999

Description:

File Types of 248 to 255 are clarified and specified in section 2.3.5.2.

Change:

Replace section 2.3.5.2 by:

2.3.5.2 Uint8 FileType

As a point to clarification a value of 5 shall be used for a standard byte addressable file, not 0. The value of 248 shall be used for the VAT (refer to 2.2.10). The value of 249 shall be used to indicate a Real-Time file (refer to 6.x). Values of 250 to 255 shall not be used.

2.3.5.2.1 File Type 249

Files with FileType 249 require special commands to access the data space of this file. To avoid possible damage, if an implementation does not support these commands it shall not issue any command that would access or modify the data space of this file. This includes but is not limited to reading, writing and deleting the file.

OSTA
Optical Storage Technology Association

Document Change Notice
DCN-5038

Document:	OSTA Universal Disk Format Specification Revision 2.00
Subject:	*Requirements for Real-Time files*
Date:	November 25, 1999, Revised December 6, 1999

Description:

To specify general descriptions of Real-Time files, Appendix 6.X is added.

Change:

Add Appendix 6.X as:

6.X Real-Time files

A Real-Time file is a file that requires a minimum data-transfer rate when writing or reading, for example, audio and video data. For these files special read and write commands are needed. For example for CD and DVD devices these special commands can be found in the Mount Fuji 4 specification.

A Real-Time file shall be identified by file type 249 in the File Type field of the file's ICB Tag.

Optical Storage Technology Association

Document Change Notice
DCN-5039

Document: OSTA Universal Disk Format Specification
Revision 2.01

Subject: *Packet Length specification*
Date: September 29, 1999, Revised December 6, 1999

Description:

The UDF specification contains a specification for the value of Packet Length in two places. The first pace is the general definition of the Sparable partition map in 2.2.9, the second is in the CD-RW specific paragraph in 6.10.2.1.
To allow a packet size different from 32 sectors on non CD-RW media, the proposal is to remove the "shall be set to 32" from 2.2.9.
As a result, the packet length is not a fixed value any more for all media, and must be specified in the medium specific part of the UDF specification for rewritable media that do not support drive based defect management.

Change:

Replace in 2.2.9, table "Layout of Type 2 partition map for sparable partition":

	Packet Length	Uint16 = 32
by:	Packet Length	Uint16

Replace in 2.2.9:

- Packet Length = the number of user data blocks per fixed packet. Shall be set to 32.

by:

- Packet Length = the number of user data blocks per fixed packet. This value is specified in the medium specific section of Appendix 6.

OSTA
Optical Storage Technology Association

UDF Document Change Notice
DCN-5040

Document: OSTA Universal Disk Format DCN-5040
Subject: *Overlapping structures with conflicting field.*
Date: September 20, 1999

Description:

In 2.3.4.3 and 2.3.10.1, two overlapping structures are defined
of which one field causes a conflict.
2.3.4.3 defines a special struct for a FID field, while
2.3.10.1 is defined for all long_ad structures.
Field 'Reserved' of 2.3.4.3 conflicts with the 'flags' field in 2.3.10.1

Changes:

In:

2.3.4.3 struct long_ad ICB

add after: The *Implementation Use* … and directory namespace.

The *Implementation Use* bytes of a long_ad hold an ADImpUse structure as defined by 2.3.10.1. The four impUse bytes of that structure will be interpreted as a Uint32 holding the UDF Unique ID.

replace on top of the table: UDF Unique ID
by: ADImpUse structure holding UDF Unique ID

in the table replace: Reserved | bytes(=#00)
by: flags | Uint16 flags, see 2.3.10.1

further replace: Section 3.2.1Logical …
by: Section 3.2.1 Logical …
(editorial)

OSTA
Optical Storage
Technology Association

UDF Document Change Notice
DCN-5041

Document: OSTA Universal Disk Format DCN-5041
Subject: *Information Length reconstruction, 2.3.6.4.*
Date: November 10, 1999

Description:

2.3.6.4 describes how the Information Length of a File Entry can be reconstructed from the allocation descriptors. The way it is described is wrong.

Changes:

Change section 2.3.6.4 into:

2.3.6.4 Uint64 InformationLength

In most cases, the InformationLength can be reconstructed during a recovery operation by finding the sum of the lengths of each of the allocation descriptors. However, space may be allocated after the end of the file (identified as a "file tail."). As "unrecorded and allocated" space is a legal part of a file body, using the allocation descriptors to determine the information length is possible under the following conditions:

- if an allocation descriptor exists with an extent length that is no multiple of the block size
- if no such extent exists and the extent type of the last allocation descriptor with an extent length unequal to 0 is not equal to "unrecorded and allocated".

Only the last extent of the file body may have an extent length that is not a multiple of the block size, see ECMA 167 4/12.1 and 4/14.14.1.1.

Change table of section 2: basic restrictions and requirements

Extent Length:

Maximum extent length shall be 2^{30} –1 rounded down to the nearest integral multiple of the Logical Block Size.

…

OSTA
Optical Storage Technology Association

UDF Document Change Notice
DCN-5042

Document: OSTA Universal Disk Format DCN-5042
Subject: *Time zone interpretation, Section 2.1.4.*
Date: November 10, 1999

Description:

Section 2.1.4.1 describes the handling of time zone specifications within timestamps. When the time zone portion of the *TypeAndTimezone* field is set to - 2047, the time zone is considered unspecified, and the interpretation of the various time fields is left to the implementation. Although it is not intended that this should change, some clarification may provide guidance as to how best to handle unspecified time zones.

Changes:

Add the following note to Section 2.1.4.1:

2.1.4.1 Uint16 TypeAndTimezone

…

Note: Implementations on systems that support time zones should interpret unspecified time zones as Coordinated Universal Time. Although not a requirement, this interpretation has the advantage that files generated on systems that do not support time zones will always appear to have the same time stamps on systems that do support time zones, irrespective of the interpreting system's local time zone.

OSTA
Optical Storage
Technology Association

Document Change Notice
DCN-5044

Document: OSTA Universal Disk Format Specification
Revision 2.01

Subject: *Missing Partition Descriptor and Sparable Partition.*
Date: December 2 , 1999

This DCN is in fact DCN 5033 with some additions.
If accepted, this DCN will overrule DCN 5033.

Description:

In the UDF specification there is no section for the Partition Descriptor, like there is for any other descriptors. Further, rules for Sparable Partition boundaries have to be defined.

Change:

Add a note to the end of 2.2.3 indicating that the partition descriptor, described in 2.2.12, really should be described next, but was omitted from earlier versions of the standard and so is at 2.2.12 in order to preserve section numbering

Change:

Add new section 2.2.12:

2.2.12 Partition Descriptor

```
struct PartitionDescriptor {                          /* ECMA 167 3/10.5 */
        struct tag          DescriptorTag;
        Uint32              VolumeDescriptorSequenceNumber;
        Uint16              PartitionFlags;
        Uint16              PartitionNumber;
        struct EntityID     PartitionContents;
        byte                PartitionContentsUse[128];
        Uint32              AccessType;
        Uint32              PartitionStartingLocation;
        Uint32              PartitionLength;
        struct EntityID     ImplementationIdentifier;
        byte                ImplementationUse[128];
        byte                Reserved[156];
}
```

2.2.12.1 struct EntityID PartitionContents

For more information on the proper handling of this field see the section on *Entity Identifier.*

2.2.12.2 Uint32 PartitionStartingLocation;

For a Sparable Partition, the value of this field shall be an integral multiple of the Packet Length. The Packet Length is defined in the Sparable Partition Map.

2.2.12.3 Uint32 PartitionLength;

For a Sparable Partition, the value of this field shall be an integral multiple of the Packet Length. The Packet Length is defined in the Sparable Partition Map.

2.2.12.4 struct EntityID ImplementationIdentifier

For more information on the proper handling of this field see the section on *Entity Identifier.*

Change:

Add to section **2.2.11 Sparing Table**

In the paragraph at the bottom of page 31, change:

… is specified by a partition descriptor.
The sparing table further specifies an exception list of logical to physical mappings.

change into:

… is specified by a partition descriptor.
A sparable partition shall begin and end on a packet boundary.
The sparing table further specifies an exception list of logical to physical mappings.

Change:

Remove from section **6.10.2.6.1 Level 1**

The start of the partition shall be on a packet boundary. The partition length shall be an integral multiple of the packet size.

Optical Storage Technology Association

UDF Document Change Notice
DCN-5045

Document: OSTA Universal Disk Format DCN-5045
Subject: *Remove "type 1" from 2. Basic Requirements, Partition Descriptor.*
Date: November 29, 1999

Description:

In 2. Basic Requirements, Partition Descriptor, the "type 1" must be removed, Because it is wrong. E.g. a Sparable Partition does not have a type 1 Partition Map. Mind that this is correct for a Virtual Partition as well, because in that case there are 2 Partition Maps, referring to only one common Partition Descriptor.

Change:

Change in section 2. Basic Restrictions & Requirements"

Partition Descriptor | … There shall be exactly one type 1 prevailing Partition
| Descriptor recorded …

change into:

Partition Descriptor | … There shall be exactly one prevailing Partition
| Descriptor recorded …

OSTA
Optical Storage
Technology Association

UDF Document Change Notice
DCN-5046

Document: OSTA Universal Disk Format DCN#5046
Subject: *Primary Volume and Logical Volume Descriptors*
Date: Nov. 1, 1999

Description:

This DCN clarifies requirements for the *PrimaryVolumeDescriptor* and *LogicalVolumeDescriptor* that facilitates identification of logical volume completeness. This is done by adding a requirement that the media containing a 1 (one) in the *VolumeSequenceNumber* field of the *PrimaryVolumeDescriptor* must be a part of any logical volume defined in the set. Additionally, it adds a requirement that the governing instance of the *LogicalVolumeDescriptor* recorded on that volume must always represent the entire logical volume.

Change:

Section 2, Basic Restrictions and Requirements Table.

In the entry for Primary Volume Descriptor, add the following:

> The media where the *VolumeSequenceNumber* of this descriptor is equal to 1 (one) must be part of the logical volume defined by the prevailing Logical Volume Descriptor.

In the entry for Logical Volume Descriptor, add the following:

> The *LogicalVolumeDescriptor* recorded on the volume where the *PrimaryVolumeDescriptor*'s *VolumeSequenceNumber* field is equal to 1 (one) must have a *NumberofPartitionMaps* value and *PartitionMaps* structure(s) that represent the entire logical volume. For example, if a volume set is extended by adding partitions, then the updated *LogicalVolumeDescriptor* written to the last volume in the set must also be written (or rewritten) to the first volume of the set.

Optical Storage Technology Association

Document Change Notice
DCN-5047

Document: OSTA Universal Disk Format Specification
Revision 2.00
Subject: Correct Omissions in Informative Table
Date: November 30, 1999

Description:

The table of informative structure sizes in section 5.1 is missing some fixed size structures. To satisfy desires for completeness, add the rest. We also note the units of the numbers in this table.

Change:

In section 5.1, update the table with the following new lines:

Descriptor	Length
Primary Volume Descriptor	512
Extended Attribute Header Descriptor	24
Extended File Entry	Max of a logical block size

Additionally, change the heading of the Length column to read “Length in Bytes”.

OSTA Optical Storage Technology Association

UDF Document Change Notice
DCN-xxx2

Document: OSTA Universal Disk Format DCN-xxx2
Subject: *UniqueID for Extended Attribute space and Named Streams.*
Date: July 29January 24, 2000

Description:

In UDF 1.02 and 1.50 there was a rule in 2.3.6.5 on UniqueID for Extended Attribute space File Entries. This rule was accidentally dropped in UDF 2.00 when the rules for UniqueID were concentrated in section 3.2.1.
Further there is an inconsistency in UDF 2.00 regarding the rules for the UniqueID of Named Streams.

3.2.1.1: …the unique ID fields in these structures take their value from the UniqueID of the File Entry/Extended File Entry of the file/directory the streams are associated with.

3.3.5.1: … The Unique ID field of Named Streams and Stream Directories shall be set to zero and shall be ignored when read. The Unique ID of a Named Stream or Stream Directory shall be considered to be the same as the Unique ID of the main data stream.

Because of analogy with the Extended Attribute case and the fact that a UniqueID value zero is normally used for the Root Directory only, the interpretation of 3.2.1.1 is chosen to be the correct one.

Change:

at the end of the 2nd paragraph of:

3.2.1.1 Uint64 UniqueID

change:

… take their value from the UniqueID of the
File Entry/Extended File Entry of the file/directory the streams are associated with.

into:

… take their value from the UniqueID of the
File Entry/Extended File Entry of the file/directory they are associated with.
The same counts for File Entries/Extended File Entries used to define an Extended Attribute space.

in: **3.3.5.1 Named Streams Restrictions**

change:

The Unique ID field of Named Streams and Stream Directories shall be set to zero

and shall be ignored when read. The Unique ID of a Named Stream or Stream Directory shall be considered to be the same as the Unique ID of the main data stream.

into:

The Unique ID field of Named Streams and Stream Directories shall be the same as the Unique ID of the main data stream.