Microsoft 365 PDF Accessibility

Article • 11/26/2024

Microsoft 365 Apps for Windows including Word, Excel, and PowerPoint allow users to export documents in PDF format. Furthermore, add-ins can use the object model to automate PDF export using either the exporter built in to each app or their own exporter that implements the IMsoDocExporter COM interface.

An important part of exporting to PDF is writing PDF/UA tags that provide the semantic information to preserve the accessibility of the content. This allows people with disabilities to consume the PDF using assistive technologies such as screen readers. This documentation provides details about the PDF/UA tags written by the exporter built in to Word, Excel, and PowerPoint as well as the APIs that add-ins need to implement to provide their own exporter.

Extending Office PDF Export

Extending Office PDF Export

Office 2024

Office 2024 PDF Accessibility Improvements

Excel

Excel PDF Accessibility

Excel.Workbook.ExportAsFixedFormat

PowerPoint

PowerPoint PDF Accessibility

PowerPoint.Presentation.ExportAsFixedFormat3

Word

Word PDF Accessibility

Word.Document.ExportAsFixedFormat2

Extending Office PDF Export

Article • 11/26/2024

Summary: Create a COM add-in for Office 2024, Office LTSC 2024, and Microsoft 365 Version 2408 and later applications with your own logic for exporting to PDF format. The technique described requires knowledge of C++ and COM.

Applies to: Excel, OneNote, PowerPoint, Publisher, Visio, and Word in Office 2024, Office LTSC 2024, Microsoft 365 Version 2408 and later.

Introduction to the Office (2024) Fixed-Format Export Feature

This article explains how third-party software developers can hook in to the fixed-format export feature available in the Office 2024, Office LTSC 2024, Microsoft 365 Version 2408 and later applications so that they can add their own exporter.

The applications include built in exporters for Microsoft XML Paper Specification (XPS) and Portable Document Format (PDF). Fixed-file formats expose the content of a document in a paginated form that is both application-independent and platform-independent.

Software developers can add their own exporter, by writing an Office add-in that implements the **IMsoDocExporter** COM interface. This article describes **IMsoDocExporter** and its interaction with a hosting Microsoft 365 application, such as Word.

Fixed-format export has been available since the Office 2007 release, and this article includes information on the features that are new in the Office 2024, Office LTSC 2024, Microsoft 365 Version 2408 releases.

Expand table

Important
The fixed-format export feature is available in all the applications listed in the preceding Applies to section. However, the discussion below uses Publisher as an example application, except in those cases where an explanation is more relevant to a different application.

Initializing Add-Ins

For the user to access add-in functionality, the add-in should add a new menu item or a new toolbar button to application. When the user selects this menu item or button, the add-in should use the Microsoft Office Object Model to obtain a pointer to the active document. It should then call the active document's **ExportAsFixedFormat** method with an **IUnknown** interface pointer that supports the **IMsoDocExporter** interface through a call to the **QueryInterface** method. The object model parameter for the interface pointer is a VARIANT with VT_UNKNOWN type.

Expand table

Note
For OneNote, the add-in calls the **Publish** method with a string parameter that is the class ID of the add-in's implementation of the **IMsoDocExporter** interface. OneNote then calls **CoCreateInstance** with the class ID to get an **IUnknown** interface pointer from the add-in's class factory.

After Publisher has a pointer to the **IMsoDocExporter** interface, it calls back the add-in through the methods exposed by **IMsoDocExporter**. Through these callbacks, Word provides the add-in with document content and other information about the document.

An excellent source of information about building COM add-ins for Microsoft Office applications is the codeproject.com article Building an Office2K COM Add-in with VC++/ATL.

IMsoDocExporter

The **IMsoDocExporter** interface exposes the following methods.

Table 1. Methods exposed by the IMsoDocExporter interface

Expand table

Method	Description
HrCreateDoc	Called at the start of the fixed-format export process.
HrAddPageFromEmf	Called to pass the add-in an enhanced metafile (EMF) that represents a rendered view of the content to export.
HrAddDocumentMetadataString	Called to specify string-format metadata for the document.
HrAddDocumentMetadataDate	Called to specify date-format metadata for the document.
HrSetDefaultLcid	Called to specify the default locale ID (LCID) for the content to

Method	Description
	export.
HrAddOutlineNode	Called to specify user-navigable document outline information.
HrGetPageBreaks	Called to obtain pagination information from the add-in.
HrSetPageHeightForPagination	Called to specify the page height to enable the add-in to paginate the document.
HrFinalize	Called at the end of the fixed-format export process. Allows the add-in to perform any final processing.
HrBeginStructNode	Called to pass the add-in the starting structure for a document-structure node that spans multiple pages.
HrEndStructNode	Called to pass the add-in the ending structure for a document-structure node that spans multiple pages.
EnableCancel	Called to pass the add-in a pointer to an **IDocExCancel** interface.
GetOutputOption	Called to retrieve fixed-format output options.
SetOutputOption	Called by Office to set fixed-format output options.
SetDocExporterSite	Called to provide the add-in with a pointer to an **IMsoDocExporterSite** interface for extended color support.

In addition, **IMsoDocExporter** also exposes the following methods that are inherited from the **IUnknown** interface.

Table 2. Methods inherited from the IUnknown interface

Expand table

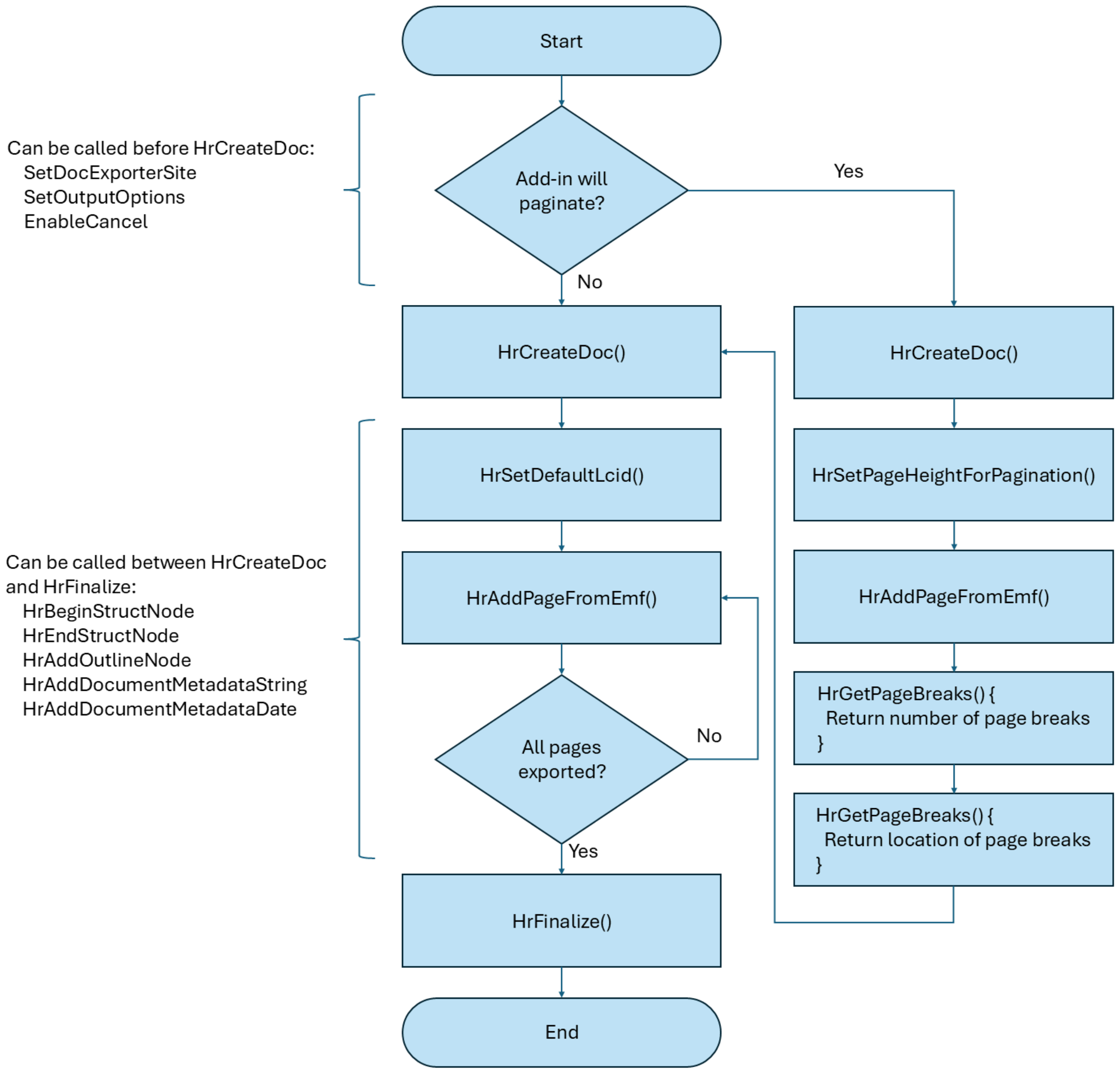
Method	Description
AddRef	Increments the reference count.
QueryInterface	Returns pointers to supported interfaces. The add-in's implementation of **QueryInterface** should support returning an **IMsoDocExporter** interface pointer from IID_IMsoPdfWriter.
Release	Decrements the reference count.

For information about implementing the **IUnknown** interface methods, see IUnknown (COM).

Call Flow

The following diagram shows the sequence in which Publisher calls the methods exposed in **IMsoDocExporter**. Not all of the methods are used by each Microsoft Office application and not all of the methods are used for every document that is exported.

Figure 1. Calling methods from the IMsoDocExporter interface


Start
Can be called before HrCreateDoc:
SetDocExporterSite
SetOutputOptions
EnableCancel
Add-in will paginate?
Yes
No
HrCreateDoc()
HrCreateDoc()
HrSetDefaultLcid()
HrSetPageHeightForPagination()
Can be called between HrCreateDoc and HrFinalize:
HrBeginStructNode
HrEndStructNode
HrAddOutlineNode
HrAddDocumentMetadataString
HrAddDocumentMetadataDate
HrAddPageFromEmf()
HrAddPageFromEmf()
HrGetPageBreaks() { Return number of page breaks }
All pages exported?
No
HrGetPageBreaks() { Return location of page breaks }
Yes
HrFinalize()
End

The following sections further describe the methods exposed by the **IMsoDocExporter** interface. The methods are described in approximately the order in which they would be called by Publisher.

GetOutputOption and SetOutputOption

Publisher calls the **GetOutputOption** and **SetOutputOption** methods to retrieve and set output options for the fixed-format export process.

```C++
void GetOutputOption(
    MSODOCEXOPTION docexoption,
    DWORD* pdwVal
);
void SetOutputOption(
    MSODOCEXOPTION docexoption,
    DWORD dwVal
);
```

The *docexoption* parameter specifies the output option and the *(p)dwVal* parameter specifies the value for the option.

While the built in exporter in Office uses **GetOutputOption** and **SetOutputOption**, an add-in can implement its own method of getting and setting options and it own user experience for the options.

Microsoft Office Calls GetOutputOption Only with msodocexOptionTargetDPIColor for Fixed-Format Add-Ins

For the implementation of fixed-format export in Office, Publisher calls the **GetOutputOption** method to retrieve output options for display to the user in the **Publish as PDF or XPS** dialog box. For add-ins developed by third-party software developers, Publisher calls **GetOutputOption** with only the msodocexOptionTargetDPIColor value. This is the only value that an add-in needs to support. If the add-in's implementation of **GetOutputOption** is called with this value, it should return the target dots-per-inch (DPI) for 3-D effect rasterization.

Microsoft Office Calls SetOutputOption for Fixed-Format Add-Ins

For both the implementation of fixed-format export in Office and for add-in implementations, Publisher calls **SetOutputOption** at the beginning of the fixed-format export process. In the implementation in Office, the parameter values passed in specify fixed-format output options. However, if the add-in implements its own set of options, the add-in can disregard the options passed to it by Publisher.

EnableCancel

Publisher calls the **EnableCancel** method to pass the add-in a pointer to an **IMsoDocExCancel** interface. The add-in can use this interface to query whether a user chooses to cancel a long document-export operation.

```C++
void EnableCancel(
    IMsoDocExCancel* pdec
);
```

HrBeginStructNode

Publisher calls the **HrBeginStructNode** method to specify the start of a document-structure node for content that encompasses multiple complete pages in the document. Document-structure nodes for elements of the document that reside entirely within a page (for example, paragraphs) are embedded by Publisher in the enhanced metafile (EMF) itself using the **DocExComment_BeginStructNode** and **DocExComment_EndStructNode** structures. For more information about document-structure nodes, see the sections HrAddPageFromEmf and DocExComment_BeginStructNode in this article.

```C++
HRESULT HrBeginStructNode(
    int idNodeParent,
    int iSortOrder,
    const MSODOCEXSTRUCTNODE* pnode,
    BOOL fNoEndNode
);
```

The *idNodeParent* parameter specifies the ID of the node that is the parent of the node being passed to the add-in. If this parameter is **0**, the node is located under the root of the document-structure tree. Multiple sibling nodes may be located under the root. If this parameter is **-1**, the node is located under the currently open node, that is, under the last node specified by **HrBeginStructNode** that has not been closed by a call to **HrEndStructNode**.

The *iSortOrder* parameter specifies the sort order of the structure node among its siblings. No two nodes can have the same sort order. However, the set of integers that constitute the sort order need not be contiguous. A value of **-1** indicates that the sibling sort order is the same order in which the nodes appear in the EMF comments.

The *pnode* parameter points to an **MSODOCEXSTRUCTNODE** structure, which has the following declaration:

```C++
```

```
typedef struct _MsoDocexStructNode
{
    int idNode;
    MSODOCEXSTRUCTTYPE nodetype;
    WCHAR* pwchAltText;
    union
    {
        int iHeadingLevel;
        ULONG idPara;
        ULONG idDropCap;
        int iPage;
        WCHAR* pwchActualText;
        MSODOCEXLINEBREAKTYPE bt;
        int iListLevel;
        MSODOCEXLISTTYPE listType;
        ULONG idAtn;
        long cpLim;
        int shapeProperty;
        MsoDocexTableAttr tableAttr;
        WCHAR* idTableHeader;
        int iTargetParentId;
    };
} MSODOCEXSTRUCTNODE;
```

The **idNode** member specifies the ID of the node being passed in the call to **HrBeginStructNode**. This member may not have a value of **0**. A value of **-1** indicates that child nodes do not use the *idNodeParent* parameter to specify this node as their parent. Instead, this node can be a parent only by enclosing child nodes in the EMF. Multiple nodes can have an ID of **-1**. If the ID is not **-1**, the value is unique across the document.

The embedded union at the end of the MSODOCEXSTRUCTNODE is interpreted differently depending on the type of node:

- **iHeadingLevel** is the heading level for an msodocexStructTypeHeading.
- **idPara** is the paragraph id for a P, TOCI, or ListBody.
- **idDropCap** is the id of an msodocexStructTypeDropCap.
- **iPage** is the page number for an msodocexStructTypePage.
- **bt** is the line break type for an msodocexStructTypeTextLine.
- **iListLevel** is the list level for an msodocexStructTypeList or msodocexStructTypeListItem.
- **listType** is the list type for an msodocexStructTypeListItem.
- **idAtn** is the id of an msodocexStructTypeAnnotationBegin or msodocexStructTypeAnnotationEnd.
- **cpLim** is used to determine the nesting order of tables within tables for an msodocexStructTypeTable, msodocexStructTypeTOC, or msodocexStructTypeListBody.

- **shapeProperty** is for a msodocexStructTypeFigure where the content is a shape, text box, or table cell and contains bit fields from the MSODOCEXSHAPEPROPERTY enumeration.
- **tableAttr** is the table cell attributes for a msodocexStructTypeTH or msodocexStructTypeTD.
- **idTableHeader** is the unique id for an msodocexStructTypeTH or msodocexStructTypeTD.
- **iTargetParentId** is the id of the node to reparent an msodocexStructTypeDiagram to.

Table 3. Enumerated values of MSODOCEXLINEBREAKTYPE

Expand table

Value	Description
msodocexLineBreakTypeNormal	Normal line break.
msodocexLineBreakTypeManual	Manual line break.
msodocexLineBreakTypeEOP	End of paragraph.

Table 4. Enumerated values of MSODOCEXLISTTYPE

Expand table

Value	Description
msodocexListTypeNone	No bullets or numbering.
msodocexListTypeBulletDisc	Disc-shaped bullets.
msodocexListTypeBulletCircle	Circle-shaped bullets.
msodocexListTypeBulletSquare	Square-shaped bullets.
msodocexListTypeBulletDecimal	Decimal numbering.
msodocexListTypeUpperRoman	Uppercase Roman numeral numbering.
msodocexListTypeLowerRoman	Lowercase Roman numberal numbering.
msodocexListTypeUpperAlpha	Uppercase alphabetic numbering.
msodocexListTypeLowerAlpha	Lowercase alphabetic numbering.

Table 5. Enumerated values of MSODOCEXSHAPEPROPERTY bit fields

Expand table

Value	Numeric Value	Description
msodocexShape	0x00000001	The object is a shape or text box.
msodocexShapeText	0x00000002	The object has non-whitespace text.
msodocexShapePath	0x00000004	The object has a fill and/or outline.
msodocexShapeAltText	0x00000008	The object has Alt Text.
msodocexShapeEquation	0x00000010	The object has text that contains an equation.
msodocexShapeTabelCell	0x00000020	The object is a cell in a table.

MsoDocexTableAttr

The **MsoDocexTableAttr** structure fits in 32 bits and includes the row and column span and header scope information for a table cell.

```C++
struct MsoDocexTableAttr
{
    static constexpr unsigned int MaxSpanBits = sizeof(unsigned int) * 8 / 2
- 1;
    static constexpr unsigned int MaxSpanValue = (1u << MaxSpanBits) - 1;

    unsigned int rowSpan : MaxSpanBits;
    unsigned int fRowScope : 1;
    unsigned int colSpan : MaxSpanBits;
    unsigned int fColScope : 1;
};
```

The members of **MsoDocexTableAttr** structure are as follows:

- **MaxSpanBits** Specifies the number of bits available for the rowSpan and colSpan values, which is 15.
- **MaxSpanValue** Specifies the maximum value that can be specified for the rowSpan and colSpan.
- **rowSpan** Specifies the number of rows that a table cell spans.
- **fRowScope** Specifies whether the header is Row/Both or Column.
- **colSpan** Specifies the number of columns that a table cell spans.

- **fColScope** Specifies whether the header is Column/Both or Row.

For table structure nodes, the union is interpreted as an ordering of the table ends relative to other tables by using **cpLim**, which can be used to determine the nesting order of tables within tables.

In the context of the **DocExComment_BeginStructNode**, the add-in can ignore the **pwchActualText** member of this union.

The **pwchAltText** member specifies alternate text for the structure node.

The *fNoEndNode* parameter to **HrBeginStructNode** specifies whether Publisher calls the **HrEndStructNode** method to mark the end of the structure node. If *fNoEndNode* is false, then Publisher calls **HrEndStructNode** to close off the content bounded by the node. If this parameter has a **true** value, then the node does not bound any content.

The *fNoEndNode* parameter affects the interpretation of the parent ID value of subsequent nodes. If *fNoEndNode* is **false**, nodes inserted between this call to **HrBeginStructNode** and the subsequent call to **HrEndStructNode**, and that have a parent ID of -1, are children of this node. However, if *fNoEndNode* is **true**, then nodes inserted after this call to **HrBeginStructNode**, and that have a parent ID of **-1**, are not children of this node but are children of the next-most-recently specified node that has *fNoEndNode* equal to **false**.

Document structure nodes can be nested to arbitrary depth.

The nodes specified by **HrBeginStructNode** and those specified by **DocExComment_BeginStructNode** share the same ID space and exist in the same document structure tree. **HrBeginStructNode** and **DocExComment_BeginStructNode** are two alternative ways of adding nodes to this tree. For example, if the most recently opened node was opened by **HrBeginStructNode** and the next node encountered is from a **DocExComment_BeginStructNode** EMFcommentrecord with *idNodeParent* equal to **-1**, it means that the node from **HrBeginStructNode** is the parent of the node from the **DocExComment_BeginStructNode** record.

HrEndStructNode

Publisher calls the **HrEndStructNode** method to specify the end of a document-structure node for content that encompasses multiple pages in the document. The structure node ended by the **HrEndStructNode** was begun previously by a call to the **HrBeginStructNode** method. For more information, see HrBeginStructNode in this article.

```C++
HRESULT HrEndStructNode();
```

HrCreateDoc

Publisher calls the **HrCreateDoc** method to specify the creation of a new, empty fixed-format document.

```C++
HRESULT HrCreateDoc(
    const WCHAR* wzDocExFile
);
```

Publisher calls the **HrCreateDoc** method at the beginning of the fixed-format export process to specify the creation of an empty fixed-format document. The *wzDocExFile* parameter specifies a name for the output file to which to write the fixed-format document.

For an add-in implementation, Publisher calls **HrCreateDoc** with the file name that the add-in provided in the call to the **ExportToFixedFormat** method in the Microsoft Office object model. However, because add-ins typically provide configuration UI to allow the user to specify an output file name, the add-in could disregard this file name during the export process.

For Microsoft Office applications that require the add-in to paginate the document, **HrCreateDoc** is called twice, once at the start of the pagination-calling sequence, and again after the add-in has paginated the document. For more information, see the descriptions for the HrSetPageHeightForPagination method and the HrGetPageBreaks method.

HrSetDefaultLcid

Publisher calls the **HrSetDefaultLcid** method to specify the default locale ID (LCID) for the content to be exported.

```C++
HRESULT HrSetDefaultLcid(
    DWORD lcid
);
```

For a list of valid LCIDs, see List of Locale ID (LCID) Values as Assigned by Microsoft.

HrAddPageFromEmf

Publisher calls the **HrAddPageFromEmf** method to pass the add-in a handle to an in-memory EMF that represents the content in the document to export.

```C++
HRESULT HrAddPageFromEmf(
    HENHMETAFILE hemf
);
```

The EMF passed by Microsoft Office to the add-in is the primary source of the content that the add-in exports as a fixed-format file. Microsoft Office calls **HrAddPageFromEmf** once for each page of content in the application's source document.

EMF Comments Convey Semantic Information

An EMF is a sequence of drawing commands (GDI and GDI+ commands) that specify how to render the visual elements of the document. The EMF does not contain any information beyond these commands (for example, "draw an image here," or "draw a line over there"). In particular, conventional EMF do not support semantic aspects of the document, such as hyperlinks, locale information, and accessibility information. To preserve semantic information in the exported document, Publisher injects special records in the EMF. These records contain the semantic information.

The records that represent the semantic information are implemented as special-formatted EMF comments. The EMF format allows for comment record types that are ignored by the rendering engine for Graphics Device Interface (GDI), but can contain arbitrary information.

As an example, consider a document that contains alternate text. (Alternate text is used by document readers to describe images for users with sight impairments.) Publisher injects EMF comments before and after rendering the image, and these EMF comments specify the alternate text for the image. The add-in interprets the comments and writes the information to the fixed-format export file.

The following table shows the semantic records types supported by the Microsoft Office fixed-format export feature. These types are enumerated by the **MSODOCEXSTRUCTTYPE** enumeration. Each type corresponds to a structure type that describes the format for the record.

Table 6. Semantic record types supported by fixed-format export

Expand table

Comment Value	Structure Type
msodocexcommentExternalHyperlink	DocExComment_ExternalHyperlink
msodocexcommentExternalHyperlinkRctfv	DocExComment_ExternalHyperlink
msodocexcommentInternalHyperlink	DocExComment_InternalHyperlink
msodocexcommentInternalHyperlinkRctfv	DocExComment_InternalHyperlink
msodocexcommentColorInfo	DocExComment_ColorInfo
msodocexcommentColorMapEnable	DocExComment_ColorEnable
msodocexcommentBeginTextRun	DocExComment_BeginTextRun
msodocexcommentBeginTextRunRTL	DocExComment_BeginTextRun
msodocexcommentEndTextRun	DocExComment_EndTextRun
msodocexcommentBeginStructNode	DocExComment_BeginStructNode
msodocexcommentEndStructNode	DocExComment_EndStructNode
msodocexcommentUnicodeForNextTextOut	DocExComment_UnicodeForNextTextOut
msodocexcommentUnicodeForNextTextOutRTL	DocExComment_UnicodeForNextTextOut
msodocexcommentEPSColor	DocExComment_EPSColor
msodocexcommentEPSCMYKJPEG	DocExComment_EPSColorCMYKJPEG
msodocexcommentEPSSpotImage	DocExComment_EPSColorSpotImage
msodocexcommentEPSStart	DocExComment_EPSStart
msodocexcommentPageName	DocExComment_PageName
msodocexcommentTransparent	DocExComment_Transparent

DocExComment_ExternalHyperlink(Rctfv)

The **DocExComment_ExternalHyperlink(Rctfv)** structure describes a hyperlink that links to outside of the document, for example to a Web site on the Internet.

```
C++
struct DocExComment_ExternalHyperlink
{
    DWORD ident {};
    DWORD iComment {};
    union
    {
        RECT  rcdvRegion;
        struct
        {
            float xLeft;
            float yTop;
            float dxWidth;
            float dyHeight;
        } rctfvRegion;
    };
    WCHAR wzLink[MAX_PATH];
};
```

The members of **DocExComment_ExternalHyperlink(Rctfv)** structure are as follows:

- **ident**   Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.

- **iComment**   Specifies the MSODOCEXCOMMENT value, msodocexcommentExternalHyperlink or msodocexcommentExternalHyperlinkRctfv.

- **rcdvRegion** and **rctfvRegion**   A union that specifies the region of the page that is the source location of the hyperlink. The region can be represented as a RECT type (rcdvRegion) that uses device pixels as the unit of measure, or as a structure that contains floating-point coordinates (rctfvRegion), in which case the unit of measure is points.

  If the **iComment** member is equal to msodocexcommentExternalHyperlink, the add-in should use **rcdvRegion**. In this case, the add-in needs to apply the current EMF transformation matrix to **rcdvRegion** to convert it to the page space.

  If the **iComment** member is equal to msodocexcommentExternalHyperlinkRctfv, the add-in should use **rctfvRegion**. In this case, **rctfvRegion** is already in the page space, so no transformation is needed.

- **wzLink[MAX_PATH]**   Specifies the destination URL for this hyperlink.

DocExComment_InternalHyperlink(Rctfv)

The **DocExComment_InternalHyperlink(Rctfv)** structure describes a hyperlink that links to a location within the document. Note that, although Publisher passes a separate EMF for each page of the document, the destination of the hyperlink specified by **DocExComment_InternalHyperlink(Rctfv)** could be on a different page than the source location.

```C++
struct DocExComment_InternalHyperlink
{
    DWORD ident {};
    DWORD iComment {};
    union
    {
        RECT  rcdvRegion;
        struct
        {
            float xLeft;
            float yTop;
            float dxWidth;
            float dyHeight;
        } rctfvRegion;
    };
    DWORD iTargetPage {};
    float xtfvTarget {};
    float ytfvTarget {};
    float dytfTargetPage {};
};
```

The members of **DocExComment_InternalHyperlink(Rctfv)** structure are as follows:

- **ident** Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.
- **iComment** Specifies the MSODOCEXCOMMENT value, msodocexcommentInternalHyperlink or msodocexcommentInternalHyperlinkRctfv.
- **rcdvRegion** and **rctfvRegion** As with the **DocExComment_ExternalHyperlink** structure, this member is a union that specifies the region of the page that is the source location of the hyperlink. The region can be represented as a RECT type (rcdvRegion) that uses device pixels as the unit of measure, or as a structure that contains floating-point coordinates (rctfvRegion), in which case the unit of measure is points.

  If the **iComment** member is equal to msodocexcommentInternalHyperlink, the add-in should use **rcdvRegion**. In this case, the add-in needs to apply the current EMF transformation matrix to **rcdvRegion** to convert it to the page space.

  If the **iComment** member is equal to msodocexcommentInternalHyperlinkRctfv, the add-in should use **rctfvRegion**. In this case, **rctfvRegion** is already in the page space, so no transformation is needed.

- **iTargetPage** Specifies the page number of the destination page within the document.
- **xtfvTarget** Specifies the x-coordinate of the target location on the destination page. The unit of measure for this value is points.
- **ytfvTarget** Specifies the y-coordinate of the target location on the destination page. The unit of measure for this value is points.
- **dytfTargetPage** The height of the destination page in points. The offset specified by the **ytfvTarget** member is relative to the upper-left corner of the page. However, some fixed-format types use a coordinate system that is relative to the bottom-left corner of the page. For these types of documents, the page height is required to convert the offset.

DocExComment_ColorInfo

The **DocExComment_ColorInfo** structure specifies color-state information for the EMF. For more information about this structure, see the section Extended Color Support.

```C++
struct DocExComment_ColorInfo
{
    DWORD ident {};
    DWORD iComment {};
    COLORREF clr { 0 };
    BOOL fForeColor {};
};
```

The members of the **DocExComment_ColorInfo** structure are as follows:

- **ident** Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.
- **iComment** Specifies the MSODOCEXCOMMENT value, msodocexcommentColorInfo.
- **clr** Specifies a color ID that represents a current color state in the EMF.
- **fForeColor** Specifies whether the color ID in the **clr** member represents a foreground color or a background color. If this member has a value of **true**, the

color ID represents a foreground color. If this member has a value of **false**, the color ID represents a background color.

DocExComment_ColorEnable

The **DocExComment_ColorEnable** structure specifies whether color mapping is enabled for subsequent content in the EMF. For more information about this structure, see the section Extended Color Support.

```
C++

struct DocExComment_ColorEnable
{
    DWORD ident {};
    DWORD iComment {};
    BOOL fEnable {};
};
```

The members of the **DocExComment_ColorEnable** structure are as follows:

- **ident** Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.
- **iComment** Specifies the MSODOCEXCOMMENT value, msodocexcommentColorMapEnable.
- **fEnable** Specifies whether color mapping is enabled for subsequent content. A value of **true** indicates that color mapping is enabled. A value of **false** indicates that color mapping is disabled.

DocExComment_BeginStructNode

The **DocExComment_BeginStructNode** structure marks the start of a document structure node. Structure nodes serve one of two possible purposes:

- Structure nodes can identify the type of content they contain and specify the hierarchical relationship between that content and other content in the document.
- Structure nodes can specify alternate text for elements in the document.

If the **fContentNode** member has a **true** value, the **DocExComment_BeginStructNode** is followed later in the document by a **DocExComment_EndStructNode**. The **DocExComment_EndStructNode** marks the end of the content that is wrapped by the information in the **DocExComment_BeginStructNode**.

The collection of structure nodes within the document forms a tree; each node has a parent node and may also have sibling nodes. The **idNodeParent** and **iSortOrder** members describe the structure of this tree. Note that a child node may or may not appear between the **DocExComment_BeginStructNode** and **DocExComment_EndStructNode** structures of the parent node in the EMF.

```C++
struct DocExComment_BeginStructNode
{
    DWORD ident {};
    DWORD iComment {};
    int idNodeParent {};
    int iSortOrder {};
    MSODOCEXSTRUCTNODE desn;
    BOOL fContentNode {};
    int cwchAltText {};
};
```

The members of the **DocExComment_BeginStructNode** structure are as follows:

- **ident** Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.
- **iComment** Specifies the MSODOCEXCOMMENT value, msodocexcommentBeginStructNode.
- **idNodeParent** Specifies the ID of the parent node. A value of **0** specifies the root node. A value of **-1** specifies the currently open structure node, that is, the *enclosing* structure node.
- **iSortOrder** Specifies the sort order of the structure node among its sibling nodes. The sort order enables the add-in to order the content correctly in the exported document.

  No two nodes can have the same sort order. However, the set of integers that constitute the sort order do not need to be contiguous.

  A value of **-1** indicates that the sibling order is the same order in which the nodes appear in the EMF comments. Note that the order in which the content appears in the EMF is not necessarily the order in which the content is consumed by a user of the document.
- **desn** Specifies a **MSODOCEXSTRUCTTYPE** structure, which is defined earlier in the document.

The **idNode** member specifies the ID of the node. This member may not have a value of **0**. A value of **-1** indicates that child nodes do not use the **idNodeParent** member to specify this node as their parent. Instead, this node can be a parent only by enclosing child nodes in the EMF. Multiple nodes can have a ID of **-1**. If the ID is not **-1**, the value is unique across the document.

The **nodetype** specifies the type of structure node. This member is equal to one of the values from the **MSODOCEXSTRUCTTYPE** enumeration type. The following table lists examples of document structure node types.

Table 7. Document structure node types

Expand table

Type Value	Description
msodocexStructTypePara	A block of text within an article. Its parent node must be an article.
msodocexStructTypeFigure	A graphical element (for example, an image or collection of shapes) that has a textual representation. The textual representation is the alternate text used for reading or searching the document.
msodocexStructTypeArticle	A group of nodes forming a single flow of text that should be read or searched as a contiguous block of content. Some documents have a single article and others have multiple articles.
msodocexStructTypeHeading	A heading in the text.
msodocexStructTypeTable	A block of text forming a table.
msodocexStructTypeTR	A block of text forming a single row of a table.
msodocexStructTypeTD	A block of text forming a single cell in a table row.
msodocexStructTypeTH	A block of text forming a single header cell in a table row.
msodocexStructTypeList	A block of text forming a list.
msodocexStructTypeListItem	A block of text forming a list item.
msodocexStructTypeListBody	A block of text forming the body of a list item.
msodocexStructTypeDocument	A document.
msodocexStructTypePage	A page in the document.

Type Value	Description
msodocexStructTypeTOC	A table of contents.
msodocexStructTypeTOCI	An item in a table of contents.
msodocexStructTypeExtLink	A link to an external resource.
msodocexStructTypeIntLink	A link to an internal resource.
msodocexStructTypeFootnote	A footnote.
msodocexStructTypeEndnote	An endnote.
msodocexStructTypeTextbox	A text box.
msodocexStructTypeHeader	A block of text forming a header.
msodocexStructTypeFooter	A footer.
msodocexStructInlineShape	An inline shape.
msodocexStructAnnotation	An annotation.
msodocexStructTypeSpanBlock	A block of text.
msodocexStructTypeWorkbook	A workbook.
msodocexStructTypeWorksheet	A worksheet.
msodocexStructTypeMacrosheet	A macrosheet.
msodocexStructTypeChartsheet	A chartsheet.
msodocexStructTypeDialogsheet	A dialogsheet.
msodocexStructTypeSlide	A slide.
msodocexStructTypeChart	A chart.
msodocexStructTypeDiagram	A SmartArt diagram.
msodocexStructTypeBulletText	Buller text.
msodocexStructTypeTextLine	A line of text.
msodocexStructTypeDropCap	A drop cap.
msodocexStructTypeSection	A section.
msodocexStructTypeAnnotationBegin	The beginning of an annotation.
msodocexStructTypeAnnotationEnd	The end of an annotation.

Type Value	Description
msodocexStructTypeParaRTLAttr	A block of text within an article with right-to-left layout.
msodocexStructTypeTableRTLAttr	A block of text forming a table with right-to-left layout.
msodocexStructTypeHeadingRTLAttr	A heading in the text with right-to-left layout.
msodocexStructTypeListItemRTLAttr	A block of text forming a list item with right-to-left layout.
msodocexStructTypeParaUnannotatableAttr	A block of text within an article that is not annotatable.
msodocexStructTypeTHead	The header row area in a table.
msodocexStructTypeTBody	The body area in a table, i.e. the portion between the THead and TFoot.
msodocexStructTypeLabel	A label.
msodocexStructTypeEquation	An equation.
msodocexStructTypeIntLinkNoteRef	A footnote or endnote reference mark link.
msodocexStructTypeTFoot	The footer row area in a table.

fContentNode   Specifies whether a **DocExComment_EndStructNode** structure marks the end of this structure node. If **fContentNode** is **true**, a **DocExComment_EndStructNode** structure closes off the content bounded by the node. If this **fContentNode** has a **false** value, then the node does not bound any content.

The **fContentNode** member affects the interpretation of the parent ID value of subsequent nodes. If **fContentNode**is **true**, nodes that are inserted between this **DocExComment_BeginStructNode** and a subsequent **DocExComment_EndStructNode**, and that have a parent ID of **-1**, are children of this node. However, if **fContentNode** is **true**, nodes inserted after this **DocExComment_BeginStructNode**, and that have a parent ID of **-1**, are not children of this node. They are children of the next-most-recently specified node that has **fContentNode** equal to **false**.

You can nest document structure nodes to arbitrary depth.

cwchAltText   Specifies the number of Unicode characters in the block of alternate text that follows the structure. This Unicode string specifies alternate text for the node (for example, alternate text for an image).

DocExComment_EndStructNode

The **DocExComment_EndStructNode** structure marks the end of the content that is decorated by the information in the **DocExComment_BeginStructNode**.

```C++
struct DocExComment_EndStructNode
{
    DWORD ident {};
    DWORD iComment {};
};
```

The members of the **DocExComment_EndStructNode** structure are as follows:

- **ident** Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.
- **iComment** Specifies the MSODOCEXCOMMENT value, msodocexcommentEndStructNode.

DocExComment_BeginTextRun

The **DocExComment_BeginTextRun** structure identifies the language of a sequence of text in the document and provides the Unicode code points for the text.

Although some text-rendering EMF records use Unicode as the text representation, others use the glyphs that are drawn on the screen, rather than the original source text. A glyph is the index of a given shape in the font, which can be different from font to font.

There can be cases where several Unicode code points are combined into a single glyph or where a single Unicode code point is broken into multiple glyphs. Because the mapping from code points to glyphs is context-dependent, a user cannot text search or copy/paste in a document that contains only glyphs. Therefore, Publisher sometimes provides the Unicode text as well as the glyphs.

```C++
struct DocExComment_BeginTextRun
{
    DWORD ident {};
    DWORD iComment {};
    DWORD lcid {};
    int cGlyphIndex {};
    int cwchActualText {};
};
```

The members of the **DocExComment_BeginTextRun** structure are as follows:

- **Ident** Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.
- **iComment** Specifies the MSODOCEXCOMMENT value, msodocexcommentBeginTextRun.
- **lcid** Specifies the LCID for the text sequence.
- **cGlyphIndex** Specifies the size of an array that follows this structure. This array implements a glyph index table that maps Unicode code points in the actual text to the corresponding glyphs in the EMF. Each element of the array corresponds to a code point in the text. The value of that element specifies the first glyph used to render that code point in the EMF. Two or more adjacent code points may have the same value in the array, which means that they both resolve to the same glyph. The value can also be **0**, which means that this code point does not map to any glyph.
- **cwchActualText** Specifies the size of the sequence of Unicode code points that follow the glyph index table. This is the text that a consumer of the document can use for searching, copying/pasting, and accessibility. The value of this member can be **0**, which means that no Unicode text is provided.

DocExComment_EndTextRun

The **DocExComment_EndTextRun** structure marks the end of a text sequence, the beginning of which was marked by a **DocExComment_BeginTextRun** structure.

```C++
struct DocExComment_EndTextRun
{
    DWORD ident {};
    DWORD iComment {};
};
```

The members of the **DocExComment_EndTextRun** structure are as follows:

- **ident** Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.

- **iComment** Specifies the MSODOCEXCOMMENT value, msodocexcommentEndTextRun.

DocExComment_UnicodeForNextTextOut

The **DocExComment_UnicodeForNextTextOut** structure functions similarly to the **DocExComment_BeginTextRun** and **DocExComment_EndTextRun** structures. However, **DocExComment_UnicodeForNextTextOut** specifies Unicode code points for only the following EMF TextOut record, rather than for a block of EMF content bounded by begin and end structures.

```C++
struct DocExComment_UnicodeForNextTextOut
{
    DWORD ident {};
    DWORD iComment {};
    int cGlyphIndex {};
    int cwchActualText {};
};
```

The members of the **DocExComment_UnicodeForNextTextOut** structure are as follows:

- **ident** Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.

- **iComment** Specifies the MSODOCEXCOMMENT value, msodocexcommentUnicodeForNextTextOut.

- **cGlyphIndex** Specifies the size of an array that follows this structure. This array implements a glyph index table that maps Unicode code points in the actual text to the corresponding glyphs in the EMF. Each element of the array corresponds to a code point in the text. The value of that element specifies the first glyph used to render that code point in the EMF. Two or more adjacent code points may have the same value in the array, which means that they both resolve to the same glyph.

- **cwchActualText** Specifies the size of the sequence of Unicode code points that follow the glyph index table. This is the text that a consumer of the document can use for searching, copying/pasting, and accessibility.

DocExComment_EPSColor

The **DocExComment_EPSColor** structure specifies color information for an encapsulated PostScript (EPS) file embedded in the EMF. For more information about this structure, see the section Extended Color Support.

```C++
typedef struct
{
    DWORD ident {};
    DWORD iComment {};
    BYTE colorInfo[];
} DocExComment_EPSColor;
```

The members of the **DocExComment_EPSColor** structure are as follows:

- **ident**   Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.
- **iComment**   Specifies the MSODOCEXCOMMENT value, msodocexcommentEPSColor.
- **colorInfo[]**   Specifies the color information for the EPS file. The add-in should pass this information to Publisher using the **IMsoDocExporterSite::SetEPSInfo** method.

DocExComment_EPSColorCMYKJPEG

The **DocExComment_EPSColorCMYKJPEG** structure specifies the start, in the EMF, of a binary object that is a CMYKJPEG file stream. For more information about this structure, see the section Extended Color Support.

```C++
typedef struct
{
    DWORD ident {};
    DWORD iComment {};
} DocExComment_EPSColorCMYKJPEG;
```

The members of the **DocExComment_EPSColorCMYKJPEG** structure are as follows:

- **ident**   Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.
- **iComment**   Specifies the MSODOCEXCOMMENT value, msodocexcommentEPSCMYKJPEG;

DocExComment_EPSColorSpotImage

The **DocExComment_EPSColorSpotImage** structure provides spot color information for the subsequent RGB image. For more information about this structure, see the section Extended Color Support.

```C++
typedef struct
{
    DWORD ident {};
    DWORD iComment {};
    COLORREF cmykAlt { 0 };
    COLORREF rgbAlt { 0 };
    float flTintMin {};
    float flTintMax {};
    char szSpotName[1];
} DocExComment_EPSColorSpotImage;
```

The members of the **DocExComment_EPSColorSpotImage** structure are as follows:

- **ident** Specifies the constant value, msodocexsignature, which identifies this EMF comment as containing semantic information.
- **iComment** Specifies the MSODOCEXCOMMENT value, msodocexcommentEPSSpotImage.
- **cmykAlt** Specifies a CMYK color ID.
- **rgbAlt** Specifies an RGB color ID.
- **flTintMin** Specifies the minimum tint.
- **flTintMax** Specifies the maximum tint.
- **szSpotName[1]** Specifies a variable length, zero-terminated string that contains the spot name.

Extended Color Support

To support extended color spaces in Publisher, additional EMF semantic records and interfaces are needed because EMF only supports RGB (red-green-black) colors. Extended color spaces include CMYK (cyan-magenta-yellow-black) and spot color space, which are commonly used in commercial printing.

Publisher uses color mapping to represent extended colors in the document EMF. Publisher builds a color table for all colors used in the document and replaces actual colors with color IDs in the EMF. The type for the color ID is **COLORREF**, which is the same type that is used for RGB color. For information about the COLORREF structure, see COLORREF.

To resolve color IDs in the EMF back to the extend color space, the add-in calls back to Publisher through the **HrResolveColor** method of the **IMsoDocExporterSite** interface. The add-in passes Publisher an interface pointer to an **IDOCEXCOLOR** interface as one of the parameters to **HrResolveColor**. Publisher takes the color IDs, also specified in the call to **HrResolveColor**, converts them to extended color (RGB, CMYK, or spot color), and passes them back to the add-in through the methods in the **IDOCEXCOLOR** interface.

Vector Color and Recolored Images

Vector colors are any **COLORREF** values that the add-in receives from Publisher. For example, text color, line stroke color, and color for metafile recolor. When color mapping is enabled, Publisher uses a color ID for **COLORREF** rather than a real RGB color value. If Publisher provides the add-in an **IMsoDocExporterSite** interface pointer by calling the **SetDocExporterSite** method of the **IMsoDocExporter** interface, the add-in should always call the **IMsoDocExporterSite::HrResolveColor** method to convert the **COLORREF** to an extended color, which the add-in receives through the methods in the **IDOCEXCOLOR** interface.

To support vector color mapping, the add-in needs to do the following:

- Implement class support for an **IDOCEXCOLOR** interface. The methods in this interface enable Publisher to pass extended color back to the add-in.
- Cache the following color state values from the semantic records in the EMF.
- Set foreground color for recoloring. This is set through the **DocExComment_ColorInfo** structure.
- Set background color for recoloring. This is set through the **DocExComment_ColorInfo** structure.
- Determine when color mapping is enabled. This is set through the **DocExComment_ColorEnable** structure.
- For a vector color, create an **IDOCEXCOLOR** interface with the color ID, so that **IDOCEXCOLOR::GetUnresolvedRGB** returns the color ID. The add-in should call the **IMsoDocExporterSite::HrResolveColor** method with the **IDOCEXCOLOR** interface and cached color states. Publisher calls the **IDOCEXCOLOR** interface methods with the final color, which can be RGB, CMYK, spot, or registration tint.

- When either foreground color or background color for recoloring is specified from an EMF semantic record, the add-in should recolor images in the add-in (for example, metafiles or raster pictures).

Non-Recolored Images

EMF supports CMYK *images* using GDI+. Therefore, images in the EMF may be either RGB or CMYK. If the image is a CMYK image, the add-in needs to convert the image to the target color space.

Publisher maintains a target color space for the document. The add-in can use this target color space by calling the **IMsoDocExporterSite::HrConvertImageColorSpace** method with the image's color space.

Color from EPS Files

Encapsulated Postscript (EPS) is a metafile type that supports extended color spaces. User who embed EPS images in a Publisher document expect the color information to be used in the fixed-format output. Inside Publisher, the EPS is converted to an EMF with EPS-related semantic records. This EMF is then embedded in the page EMF file that the application passes to the add-in.

To support color in EPS files, the add-in needs to do the following:

- Call the **IMsoDocExporterSite::SetEPSInfo** method for **DocExComment_EPSColor** records encountered in the EMF.
- Extract the CMYK image from the **DocExComment_EPSColorCMYKJPEG** record in the EMF. This record contains a binary object that is the actual CMYK JPEG file stream. Use it to replace the RGB image specified in the subsequent call to the **StretchDIBits** function.
- The **DocExComment_EPSColorSpotImage** record provides spot color information for the subsequent RGB image, which is always an index image. The add-in needs to convert the spot image to the target color space.
- The add-in can optionally call the **IMsoDocExporterSite:: HrGetSpotRecolorInfo** method to obtain the document's target color from Publisher. Then the add-in can recolor the subsequent RGB image by mapping colors from the palette of the RGB image to **flTintMin** and **flTintMax** tints specified in the **DoxExComment_EPSColorSpotImage** record. The luminosity for each color of the palette is used for the mapping.

Note that the **DocExComment_EPSStart** record is only informational. The add-in can ignore this record.

SetDocExporterSite

Publisher calls **SetDocExporterSite** to provide the add-in with a pointer to an **IMsoDocExporterSite** interface. The **IMsoDocExporterSite** interface exposes methods that enable extended color support.

```C++
void SetDocExporterSite(
    IMsoDocExporterSite* pDocExporterSite
);
```

The *pDocExporterSite* parameter specifies the interface pointer to the **IMsoDocExporterSite** interface.

HrSetPageHeightForPagination

An application can call the **HrSetPageHeightForPagination** method to specify the page height in points.

```C++
HRESULT HrSetPageHeightForPagination(
    float dytfPageHeight
);
```

Some applications maintain the user's document in an unpaginated format. In these cases, the add-in paginates the document using the page height specified by the application in the call to **HrSetPageHeightForPagination**. The *dytfPageHeight* parameter specifies the page height in points.

After specifying the page height information, the application passes the add-in the entire document as a single in-memory EMF file in a call to **HrAddPageFromEmf**. The add-in then uses the page-height and EMF file to paginate the document.

The add-in returns the pagination information back to the application in subsequent calls to the **HrGetPageBreaks** method.

HrGetPageBreaks

An application can call the **HrGetPageBreaks** method to obtain the number and location of page breaks for documents that are paginated by the add-in.

```C++
HRESULT HrGetPageBreaks(
    float* rgdytfPageBreaks,
    int* pcchPageBreaks,
    BOOL* pfCanTrustLastBreakIsEndOfDocument
);
```

After the add-in paginates a document using the page height specified by the **HrSetPageHeightForPagination** method, it returns the pagination information in subsequent calls that the application makes to the **HrGetPageBreaks** method.

The *rgdytfPageBreaks* parameter is a pointer to an array of float values that specify the locations of the page breaks in points. The first element in the array (index 0) is the location of the first page break, the second element is the location of the second page break, and so on. Therefore, the values of these elements are successively increasing.

The *pcchPageBreaks* parameter is a pointer to an integer value that specifies the number of page breaks in the document.

The *pfCanTrustLastBreakIsEndOfDocument* parameter specifies whether the location of the last page break is the end of the document or the beginning of the last page of the document. A **true** value indicates that the last page break is the end of the document.

The application calls **HrGetPageBreaks** twice to obtain the pagination information. On the first call, the application calls **HrGetPageBreaks** to obtain the number of page breaks.

```C++
HrGetPageBreaks(NULL, &nPageBreaks, NULL);
```

The application then calls **HrGetPageBreaks** a second time to obtain the actual locations. On the second call, the application passes a buffer of sufficient size to hold the array of page-break locations.

```C++
HrGetPageBreaks(rgPageBreaks, &nPageBreaks, fCanStopAtLastPageBreak);
```

After receiving the page break information from the add-in, the application re-initiates the fixed-format export process, beginning with a call to the **HrCreateDoc** method, followed by a call to **HrAddPageFromEmf** for each of the pages given by the page-break information.

HrAddOutlineNode

Publisher calls the **HrAddOutlineNode** method to pass the add-in a structure that describes a node within a user-navigable outline for the exported document.

```C++
HRESULT HrAddOutlineNode(
    int idNodeParent
    const MSODOCEXOUTLINENODE* pNode
);
```

The fixed-format export code can use the information passed by the **HrAddOutlineNode** method to construct a user-navigable outline of the export document. From the user's perspective, each node in the outline is represented by some title text that maps to a particular location within the document.

Each call to **HrAddOutlineNode** specifies information for a single node in this outline. Each node is identified by a node ID that is unique within the outline. An ID of **0** is reserved for the root node. The outline is hierarchical, that is, it has a tree structure in which each node has a single parent and zero or more child nodes.

The first parameter to **HrAddOutlineNode** provides the ID of the node that is the parent of the node being passed in.

Publisher always calls **HrAddOutlineNode** for a parent node before calling the method for any of the parent node's children. In other words, the export code is assured of already having the node information for the node identified by the *idNodeParent* parameter. The only exception is the initial call to **HrAddOutlineNode** that specifies the root node. For this call, the value of *idNodeParent* is **0**.

Additional information that the export code needs for each node is passed by **HrAddOutlineNode** in an **MSODOCEXOUTLINENODE** structure pointed to by the *pNode* parameter.

```C++
typedef struct _MsoDocexOutlineNode
{
    int idNode {};
    WCHAR rgwchNodeText[cwchMaxNodeText];
    int iDestPage {};
    float dytfvDestPage {};
    float dxtfvDestOffset {};
    float dytfvDestOffset {};
} MSODOCEXOUTLINENODE;
```

The members of the **MSODOCEXOUTLINENODE** are described as follows:

- **idNode** The ID for the node. A value of **-1** indicates that this node cannot have child nodes in the outline. Otherwise, this member has a value that is unique across the document.
- **rgwchNodeText** A Unicode string that represents the title text for each node. This text is not required to be unique across the outline.
- **iDestPage** The page number of the page that contains the destination location within the document.
- **dytfvDestPage** The height of the destination page in points. The offset specified by the **dytfvDestOffset** member is relative to the upper-left corner of the page. However, some fixed-format types use a coordinate system that is relative to the bottom-left corner of the page. For these types of documents, the page height is required to convert the offset.
- **dxtfvDestOffset** The horizontal offset of the destination location on the destination page.
- **dytfvDestOffset** The vertical offset of the destination location on the destination page.

HrAddDocumentMetadataString

Publisher calls the **HrAddDocumentMetadataString** method to specify document metadata in the form of a Unicode string.

```C++
HRESULT HrAddDocumentMetadataString(
    MSODOCEXMETADATA metadataType,
    const WCHAR* pwchValue
);
```

The *metadatatype* parameter specifies the type of metadata represented by the string. The *metadatatype* parameter must be one of the following values from the MSODOCEXMETADATA enumeration type.

Table 8. Enumerated values of MSODOCEXMETADATA

Expand table

Value	Description
msodocexMetadataTitle	The title of the document.
msodocexMetadataAuthor	The author of the document
msodocexMetadataSubject	String that describes the subject matter of the document (for example, business or science).
msodocexMetadataKeywords	Keyword relevant to the document content.
msodocexMetadataCreator	The creator of the document, possibly distinct from the author.
msodocexMetadataProducer	The producer of the document, possibly distinct from the author or creator.
msodocexMetadataCategory	String that describes the type of document (for example, memo, article, or book).
msodocexMetadataStatus	Status of the document. This field can reflect where the document is in the publication process (for example, draft or final).
msodocexMetadataComments	Miscellaneous comments relevant to the document.

For a given document, each metadata type can have only one string associated with it. So, for example, if the document has multiple keywords, they are passed to the add-in as one concatenated string.

The *pwchValue* parameter specifies a Unicode string that contains the metadata itself.

How the add-in incorporates the text-string metadata into the exported document depends on the implementation details of the export code and the type of fixed-format used in the exported document.

HrAddDocumentMetadataDate

Publisher calls the **HrAddDocumentMetadataDate** method to specify document metadata in the form of a FILETIME structure.

C++

```
HRESULT HrAddDocumentMetadataDate(
    MSODOCEXMETADATA metadataType,
    const FILETIME* pftLocalTime
);
```

The *metadatatype* parameter specifies the type of metadata represented by the **FILETIME** structure. The *metadatatype* parameter must be one of the following values from the MSODOCEXMETADATA enumeration type.

Table 9. Enumerated values of MSODOCEXMETADATA

Expand table

Value	Description
msodocexMetadataCreationDate	The creation date for the document.
msodocexMetadataModDate	The last-modified date for the document.

The *pftLocalTime* parameter specifies a pointer to a FILETIME structure that contains the date and time information for the metadata. The following code snippet demonstrates how to extract this information from the structure.

C++

```
SYSTEMTIME st = { 0 };
WCHAR s[100];
FileTimeToSystemTime(pfiletime, &st);
swprintf(s, 99, L" %04d-%02d-%02dT%02d:%02d:%02dZ", st.wYear % 10000,
    st.wMonth % 100, st.wDay % 100, st.wHour % 100, st.wMinute % 100,
    st.wSecond % 100);
```

How the add-in incorporates the date and time metadata into the exported document depends on the implementation details of the export code and the type of fixed-format used in the exported document.

HrFinalize

Publisher calls the **HrFinalize** method at the end of the document-export process.

C++

```
HRESULT HrFinalize();
```

The code that implements fixed-format export should use **HrFinalize** to perform tasks such as flushing data buffers, writing remaining data to disk, and freeing memory and other resources.

Conclusion

You can extend the fixed-format export feature of Office applications by implementing the **IMsoDocExporter** interface. The methods of this interface provide a channel for Office applications to communicate to the add-in the visual content and semantic information in the document to export. The visual content of the document is provided to the add-in as one or more in-memory enhanced metafiles. The semantic information is provided as specially formatted comment records within this EMF. Additional methods in the interface enable Office applications to communicate metadata and structural information about the document.

Additional Resources

For more information, see the following resources:

- Word.Document.ExportAsFixedFormat2
- Excel.Workbook.ExportAsFixedFormat
- PowerPoint.Presentation.ExportAsFixedFormat3
- Project.Project.ExportAsFixedFormat
- Publisher.Documents.ExportAsFixedFormat
- Visio.Document.ExportAsFixedFormat
- OneNote.Application.Publish

Office 2024 PDF Accessibility

Article • 11/26/2024

Summary

Accessibility of PDFs exported by Word, Excel, and PowerPoint is greatly improved in Office 2024, Office LTSC 2024, and Microsoft 365 Version 2408 and later. This page has a summary of the improvements to help add-in developers and template providers adapt to the changes.

Expand table

Number	Improvement	Word	Excel	PowerPoint
1	Document has <Document> tag			Yes
2	Chart has <Figure> tag with Alt Text	Yes	Yes	Yes
3	Add-in has <Figure> tag with Alt Text	Yes	Yes	Yes
4	3D Model has <Figure> tag with Alt Text	Yes	Yes	Yes
5	Ink has <Figure> tag with Alt Text	Yes	Yes	Yes
6	Animated Ink has <Figure> tag with Alt Text			Yes
7	Table with Alt Text has <Figure> tag with Alt Text			Yes
8	SmartArt has single <Figure> tag (no <Figure> tags for child elements)	Yes	Yes	Yes
9	SmartArt without Alt Text has <Figure> tag Alt Text set to text outline of SmartArt	Yes	Yes	Yes
10	Group with Alt Text has single <Figure> tag (no <Figure> tags for child elements)	Yes		Yes
11	Graphical Object other than Shape without Alt Text has <Figure> tag with blank Alt Text	Yes	Yes	Yes
12	Shape with whitespace text or no text has <Figure> tag with blank Alt Text		Yes	Yes
13	Shape with Equation without Alt Text or Decorative has <Formula> tag with Alt Text		Yes	Yes
14	Shape with non-whitespace text without Equation, Alt	Yes	Yes	Yes

Number	Improvement	Word	Excel	PowerPoint
	Text, or Decorative has <Sect> tag with text content			
15	Shape with Alt Text and non-whitespace text without Equation has <Figure> tag with Alt Text		Yes	Yes
16	Table Cell with Equation in a Table without Alt Text has <Formula> tag with Alt Text			Yes
17	Summary Zoom, Section Zoom, and Slide Zoom has <TOC> tag without Alt Text			Yes
18	WordArt preserved as text	Yes	Yes	Yes
19	Alt Text includes the Alt Text Title as well as Alt Text Description	Yes	Yes	Yes
20	Alt Text includes the Object type	Yes	Yes	Yes
21	Alt Text for Shape with Alt Text includes the text content		Yes	Yes
22	Header Row in a Table has <THead> tag			Yes
23	Total Row in a Table has <TFoot> tag			Yes
24	Header Cell in Table has Scope=Row, Column, or Both in <TH> tag			Yes
25	Cell in a Table has Row span and Column span set properly on <TH> or <TD> tag			Yes
26	For lists in PowerPoint, bullet or number is in <Lbl> tag			Yes
27	For lists in Word, bullet or number is in <Lbl> tag	Yes		
28	For picture bullets, no <Lbl> tag is included	Yes		Yes
29	For nested lists, <L> tag is in <LBody> tag	Yes		Yes
30	Document language in PDF set as the document language in PowerPoint			Yes
31	Document language in PDF set as the document language in Word	Yes		
32	Text in different language has <Span> tag with Lang property in PowerPoint			Yes
33	Text in different language has <Span> tag with Lang property in Word	Yes		

Number	Improvement	Word	Excel	PowerPoint
34	Actual Text property removed from <Span> tag for text in different language in PowerPoint			Yes
35	Actual Text property removed from <Span> tag for text in different language in Word	Yes		
36	<H1> tag for Title Placeholder on Title or Section Header layout in PowerPoint			Yes
37	<H2> tag for Title Placeholder on other slides in PowerPoint			Yes
38	<H1> and <H2> tag logic in PowerPoint works for custom templates			Yes
39	Hyperlinks aren’t nested in <Figure> tags	Yes		Yes
40	Hyperlinks are preserved even for Decorative Objects and Objects on the Slide Master or Slide Layout			Yes
41	For a Hyperlink on a Graphical Object, <Link> tag is a sibling of the tag for the Object	Yes		Yes
42	For a Hyperlink on text in a Shape with Alt Text, Equation, or Decorative, <Link> tag is a sibling of the tag for the Object	Yes		Yes
43	For a Hyperlink on text in a Table Cell with Alt Text, Equation, or Decorative, <Link> tag is a sibling of the tag for the Object			Yes
44	Document Title in PDF set as Document Title in Word	Yes		
45	Comment Anchor has <Span> tag in Word	Yes		
46	Quote Style has <Quote> tag in Word	Yes		
47	Caption has <Caption> tag in Word	Yes		
48	Paragraph spanning multiple pages has single <P> tag in Word	Yes		
49	Footnote and Endnote have <Link> tag in Word	Yes		
50	Footnote and Endnote have <Note> tag in Word	Yes		
51	Paragraph Quote and Intense Quote Styles have <BlockQuote> tag in Word	Yes		
52	Title Style has <Title> tag in Word	Yes		

Number	Improvement	Word	Excel	PowerPoint
53	Heading Levels beyond Heading 6 have <H6> tag	Yes		
54	Table Header Cell has unique ID		Yes	
55	Cell with Header Style has <H1> tag		Yes	
56	Marked as artifact and no tags for Shape without Alt Text, text, fill, or outline		Yes	Yes
57	Marked as artifact and no tags for Decorative Shapes	Yes	Yes	Yes
58	Marked as artifact and no tags for Decorative Ink	Yes	Yes	Yes
59	Marked as artifact and no tags for Decorative Add-ins	Yes	Yes	Yes
60	Marked as artifact and no tags for Decorative Group	Yes		Yes
61	Marked as artifact and no tags for Decorative Chart	Yes	Yes	Yes
62	Marked as artifact and no tags for Decorative SmartArt	Yes	Yes	Yes
63	Marked as artifact and no tags for Decorative Table			Yes
64	Marked as artifact and no tags for Objects on the Slide Master and Slide Layout			Yes
65	Marked as artifact and no tags for text formatting marks such as underline, strike-through, and highlight		Yes	Yes
66	Marked as artifact and no tags for picture bullets	Yes		Yes
67	Marked as artifact and no tags for Comment anchor text shading, brackets, bubbles, and lines in Word	Yes		
68	Marked as artifact and no tags for paragraph borders, text borders, and text underlines in Word	Yes		
69	Marked as artifact and no tags for paragraph borders, text borders, and text underlines in Word	Yes		
70	Marked as artifact and no tags for text box container in Word	Yes		
71	Marked as artifact and no tags for Slicer scrollbar in Excel		Yes	
72	Marked as artifact and no tags for grid lines and cell borders in Excel		Yes	
73	Marked as artifact and no tags for cell shading in Excel		Yes	

Number	Improvement	Word	Excel	PowerPoint
74	Removed unnecessary <Span> tags for text with justified or distributed alignment			Yes
75	Removed unnecessary <Span> tags for text runs with different formatting			Yes
76	Removed unnecessary <Span> tags for text formatting marks such as highlight			Yes
77	Removed unnecessary <Span> tags for WordArts that have text on a path	Yes	Yes	Yes
78	Removed unnecessary <Span> tags for WordArts that have warped text	Yes	Yes	Yes
79	Removed unnecessary <Span> tags for graphical effect on WordArts such as shadow and reflection	Yes	Yes	Yes
80	Removed unnecessary <Span> tags for the slide background in PowerPoint for macOS and web			Yes
81	Removed unnecessary <P> tags	Yes		
82	Fixed a problem in Word where Save as Adobe PDF did not preserve headings properly	Yes		
83	Fixed a problem in PowerPoint where <TD> and <TH> were omitted for empty Table Cells			Yes
84	Fixed a problem in PowerPoint where Hyperlinks to slides within the presentation were going to the bottoms of the slides rather than the tops of the slides			Yes
85	Fixed a problem in PowerPoint where Hyperlinks in Notes Pages and Outline View were not preserved			Yes
86	Fixed a problem in PowerPoint in Notes Pages where the wrong tags were written for Decorative shapes			Yes
87	Fixed a problem in PowerPoint in Notes Pages where when the notes for a slide spill onto the multiple slides, the wrong tags were written			Yes
88	Fixed a problem in PowerPoint in Notes Pages where for a list that spills across pages, the partial list item on the next page is outside of the <L> tag. Now the partial list item is included in the <L> tag. There is a separate <L> tag in each page.			Yes

Number	Improvement	Word	Excel	PowerPoint
89	Fixed a problem in PowerPoint for the web where Decorative Charts were not marked as Artifact			Yes
90	Fixed a problem in PowerPoint for the web where Hyperlinks to slides within the presentation were going to the bottoms of the slides rather than the tops of the slides			Yes
91	Fixed a problem in PowerPoint for the web where the Alt Text on the <Figure> tag for a SmartArt doesn't include the diagram type			Yes
92	Fixed a problem in PowerPoint for the web where the <Link> tag was missing for a hyperlink to a slide within the presentation			Yes
93	Fixed a problem in PowerPoint for macOS where sometimes the wrong tags are included			Yes
94	Fixed a problem in PowerPoint for macOS where export as PDF fails for a presentation with custom slide numbering			Yes
95	Fixed a problem in PowerPoint for macOS where sometimes hidden slides are included			Yes
96	Fixed a problem in PowerPoint for macOS where sometimes a slide with a Summary Zoom, Section Zoom, and Slide Zoom is blank			Yes
97	Fixed a problem in Excel and PowerPoint where spaces between words were omitted in some cases		Yes	Yes
98	Fixed a problem in Excel and PowerPoint where trailing spaces in a text element were omitted		Yes	Yes
99	Fixed a problem in Excel and PowerPoint where when there are multiple copies of the same SVG in a presentation, the wrong tags were written		Yes	Yes
100	Fixed a problem in Excel and PowerPoint where the wrong tags are written when there are empty paragraphs		Yes	Yes
101	Fixed a problem in Excel where images in cells were marked as artifact		Yes	
102	Fixed a problem in Excel where export as PDF fails when the Workbook contains an Object with a		Yes	

Number	Improvement	Word	Excel	PowerPoint
	Hyperlink to Sheet within the Workbook			
103	Added bookmarks for sections and slides			Yes
104	Added the Presentation.ExportAsFixedFormat3 method to the Object Model in PowerPoint to export as PDF with all of the accessibility improvements listed here enabled			Yes

Excel PDF Accessibility

Article • 11/26/2024

Summary

Authors can ensure that their Execl workbooks are accessible to people with disabilities even when distributing them in PDF format using the following approach:

1. First, they should follow the practices in Accessibility best practices with Excel spreadsheets.
2. Next, they should follow the steps in Create accessible PDFs to preserve the accessibility of the workbook in PDF format.

This article provides details about the information Excel includes in the PDF to make it accessible.

1. PDF/UA tags are included to provide semantic information about the content in the document.
2. Decorative content does not need to be read, so it is marked as <Artifact> in the Content Tree in the PDF and no PDF/UA tags are included.

PDF/UA Tags

Expand table

Type of content	Tags

Workbook	`<Document>`
	`  Type=Workbook`

Range of Cells	`<Table>`
	`  <TR>`
	`    <TD>`
	`      text content`

Table	`<Table>`
	`  <THead>`
	`    <TR>`

Type of content	Tags
	`<TH>` *`text content`* `<TH>` *`text content`* `<TBody>` `<TR>` `<TH>` *`text content`* `<TD>` *`text content`*
Table Header Cell	`<TH>` `ID=`*`unique id`*
Cell with Header Style	`<H1>`
Header and Footer	*`no tags`*
Comment	*`no content`*
Decorative Graphical Object	*`no tags`*
Graphical Object other Group with Alt Text	`<Figure>` `Alt=`*`alt text (object type)`*
Graphical Object other than Shape without Alt Text	`<Figure>` `Alt=`*`blank`*
Shape without Alt Text, text, fill, outline, or hyperlink	*`no tags`*

Type of content	**Tags**
Shape without Alt Text with whitespace text or no text	`<Figure>` `Alt=`*blank*
Shape without Alt Text with Equation	`<Formula>` `Alt=`*equation spelled out in words*
Shape without Alt Text with non-whitespace text without Equation	`<Sect>` *text content*
Shape with Alt Text with non-whitespace text without Equation	`<Figure>` `Alt=`*alt text + text (shape type)*
WordArt without Alt Text or Decorative	`<Sect>` *text content*
Group without Alt Text	*tags for child objects*
Group with Alt Text	`<Figure>` `Alt=`*alt text (object type)* *tags for child objects*
Decorative Picture in Cell	`<TD>` *no tags*
Picture without Alt Text in Cell	`<TD>` *no tags*

Type of content	Tags
Picture with Alt Text in Cell	`<TD>` `<Figure>` `Alt=`*`alt text`*
Hyperlink on Cell	`<TD>` `<P>` `<Link>` `Link - OBJR` `<Span>` *`text content`*
Hyperlink on Object	*`tag for object`* `<Link>` `Link - OBJR`

Artifacts

The following types of content are marked as <Artifact> in the PDF Content Tree and have no PDF/UA tags:

- Slicer scrollbar
- Grid lines
- Cell borders
- Cell shading
- Decorative graphical objects
- Text in SmartArt objects

Availability

The information in this article is applicable to the following versions of Excel.

- Excel for Windows Version 2408 and later.
- Excel for Mac Version 16.89 and later.
- Excel for iOS Version 2.89 and later.
- Excel for Android Build 16.0.18025.XXXXX or later.

It is available to customers with Office 2024 or Office LTSC 2024 and to customers with a Microsoft 365 subscription on Current Channel or Monthly Enterprise Channel. For customers with a Microsoft 365 subscription on Semi-Annual Enterprise Channel it will be available on January 14, 2025.

Currently, Excel for the web does not support saving as an accessible PDF.

PowerPoint PDF Accessibility

Article • 11/26/2024

Summary

Authors can ensure that their PowerPoint presentations are accessible to people with disabilities even when distributing them in PDF format using the following approach:

1. First, they should follow the practices in Make your PowerPoint presentations accessible to people with disabilities.
2. Next, they should follow the steps in Create accessible PDFs to preserve the accessibility of the presentation in PDF format.

This article provides detailed information about the information PowerPoint includes in the PDF to make it accessible.

1. PDF/UA tags are included to provide semantic information about the content in the presentation.
2. Decorative content does not need to be read, so it is marked as <Artifact> in the Content Tree in the PDF and no PDF/UA tags are included.
3. Bookmarks for each section and slide are included to make it easier to navigate the content.

PDF/UA Tags

Expand table

Type of content	Tags
Document	`<Document>`
Slide	`<Sect>`
Comment	*`no tags`*

Type of content	**Tags**
Decorative Object	*no tags*
Object with Alt Text	<Figure> Alt=*alt text (object type)*
Object other than Shape without Alt Text	<Figure> Alt=*blank*
Shape without Alt Text, text, fill, outline, or hyperlink	*no tags*
Shape without Alt Text with whitespace text or no text	<Figure> Alt=*blank*
Shape without Alt Text with Equation	<Formula> Alt=*equation spelled out in words*
Shape without Alt Text with non-whitespace text without Equation	<Sect> *text content*
Shape with Alt Text with non-whitespace text without Equation	<Figure> Alt=*alt text + text (shape type)*
WordArt without Alt Text or Decorative	<Sect> *text content*

Type of content	Tags
Table without Alt Text	`<Table>` `<THead>` `<TR>` `<TH>` *text content* `<TH>` *text content* `<TBody>` `<TR>` `<TH>` *text content* `<TD>` *text content* `<TFoot>` `<TR>` `<TH>` *text content* `<TD>` *text content*
Table Header Cell	`<TH>` `Scope=Row, Column, or Both`
Table Merged Cell	`<TH> or <TD>` `Row span=`*r* `Column span=`*c*
Group without Alt Text	*tags for child objects*
Summary Zoom, Section Zoom, and Slide Zoom	`<TOC>` `Alt=`*alt text* `<TOCI>` `<Link>` `Link - OBJR` `<Span>`

Type of content	**Tags**
Paragraph	`<P>`
Bullets and Numbering	`<L>` `<LI>` `<Lbl>` *bullet or number* `<LBody>` *text content*
Picture Bullets	`<L>` `<LI>` `<LBody>` *text content*
Nested Bullets and Numbering	`<L>` `<LI>` `<Lbl>` *bullet or number* `<LBody>` *text content* `<L>` `<LI>` `<Lbl>` *bullet or number* `<LBody>` *text content* `...`
Text in different language	`<Span>` `Lang=`*language code*
Header and Footer	*no tags*

Type of content	Tags
Objects that appear in Slide Master View	*`no tags (except hyperlinks)`*
Title Placeholder on first slide in presentation or section or slide with Title or Section Header layout	`<H1>`
Title Placeholder on all other slides	`<H2>`

Hyperlink on Text	`<Link>`
	`  Link - OBJR`
	`  text content`

Hyperlink on Object	*`tag for object`*
	`<Link>`
	`  Link - OBJR`
	`  alt text`
	Note: the <Link> is a sibling of the tag for the object.

Artifacts

The following types of content are marked as <Artifact> in the PDF Content Tree and have no PDF/UA tags:

- Decorative objects
- Header and footer
- Objects that appear in Slide Master View
- Table borders
- Text formatting marks including underline, strikethrough, highlight, and shadow
- Text in SmartArt objects

Bookmarks

Bookmarks are included in the PDF for each section and slide in the presentation. The bookmarks use the section names and slide titles given in PowerPoint. Authors should provide a unique and meaningful title for each slide. See Title a slide.

Object Model

The PowerPoint object model includes the Presentation.ExportAsFixedFormat3 method to export the presentation as PDF with accessibility as described here.

1. Bookmarks for each section and slide are only included when Bookmarks:=True is specified.
2. PDF/UA tags are only included when DocStructureTags:=True is specified.
3. The <Document> tag is only included when DocumentMarkup:=True is specified.
4. A <Link> tag is only written as a sibling of the tag for the object when PromotedHyperlinkShape:=True is specified.

The older Presentation.ExportAsFixedFormat2 method does not include the Bookmarks, DocumentMarkup, and PromotedHyperlinkShape parameters. It works the same as the Presentation.ExportAsFixedFormat3 method with these 3 parameters set to False.

Availability

The information in this article is applicable to the following versions of PowerPoint.

- PowerPoint for Windows Version 2408 and later.
- PowerPoint for Mac Version 16.89 and later.
- PowerPoint for iOS Version 2.89 and later.
- PowerPoint for Android Build 16.0.18025.XXXXX or later.
- PowerPoint for the web Build 16.0.18025.XXXXX or later.

It is available to customers with Office 2024 or Office LTSC 2024 and to customers with a Microsoft 365 subscription on Current Channel or Monthly Enterprise Channel. For customers with a Microsoft 365 subscription on Semi-Annual Enterprise Channel it will be available on January 14, 2025.

Word PDF Accessibility

Article • 04/07/2025

Summary

Authors can ensure that their Word documents are accessible to people with disabilities even when distributing them in PDF format using the following approach:

1. First, they should follow the practices in Make your Word documents accessible to people with disabilities.
2. Next, they should follow the steps in Create accessible PDFs to preserve the accessibility of the document in PDF format.

This article provides details about the information Word includes in the PDF to make it accessible.

1. PDF/UA tags are included to provide semantic information about the content in the document.
2. Decorative content does not need to be read, so it is marked as <Artifact> in the Content Tree in the PDF and no PDF/UA tags are included.
3. Bookmarks for headings or Word bookmarks (depending on the option selected) are available to make it easier to navigate the content.

PDF/UA Tags

Expand table

Type of content	Tags
Document	`<Document>`
Title	`<Title>`
H1, H2, etc.	`<H1>, <H2>, etc.`

Type of content	**Tags**
Paragraph, Subtitle, Content Controls, and Legacy Controls	`<P>`
Paragraph Quote and Intense Quote	`<BlockQuote>`
Inline Quote (*Version ≥ 16.0.17004.20000*)	`<Quote>`
Caption	`<Caption>`
Table of Contents	`<TOC>` `  <TOCI>` `    Table of Contents` `  <TOCI>` `    <Link>` `      Link - OBJR` `      <Span>` *`        Complete line text`* `    <Span>` *`      empty`* `  <TOCI>` `    ...` `  <TOCI>` `    ...`
Header and Footer	*`no tags`*
Comment (*Version ≥ 16.0.16831.20002*)	`<P>` *`  paragraph content`* `  <CommentAnchor>` `    <Span>` *`      anchor text`* `    <Annot>` *`      comment content`*

Type of content	Tags
Footnotes and Endnotes	`<P>` *`paragraph content`* `<Link>` `Link – OBJR` `<Span>` *`footnote number`* `<Note>` *`footnote text`*
Signature Line, ActiveX Controls, and OLE Objects	`<P>` `<Sect>`
Text in different language (*Version ≥ 16.0.16922.20000*)	`<Span>` `Lang=`*`language code`*
Bullets and Numbering	`<L>` `<LI>` `<Lbl>` *`bullet or number`* `<LBody>` *`text content`*
Picture Bullets	`<L>` `<LI>` `<LBody>` *`text content`*
Nested Bullets and Numbering	`<L>` `<LI>` `<Lbl>` *`bullet or number`* `<LBody>` *`text content`* `<L>` `<LI>` `<Lbl>` *`bullet or number`*

Type of content	Tags
	`<LBody>` *`text content`* `...`
Table	`<Table>` `<THead>` `<TR>` `<TH>` *`text content`* `<TH>` *`text content`* `<TBody>` `<TR>` `<TH>` *`text content`* `<TD>` *`text content`*
Table Header Cell	`<TH>`
Layout Table (*Version ≥ 16.0.18526.20168, beta channel*)	If a table does not have borders or shaded cells, it is treated as a layout table. In this case, a *<P>* tag is created for each cell.
Decorative Graphical Object	*`no tags`*
Graphical Object with Alt Text	`<Figure>` `Alt=`*`alt text (object type)`*
Graphical Object other than Shape without Alt Text	`<Figure>` `Alt=`*`blank`*
Shape without Alt Text with text	`<Sect>` *`text content`*

Type of content	Tags

Shape with Alt Text with text	`<Figure>`
	  `Alt=`*`alt text + text (shape type)`*

WordArt without Alt Text or Decorative	`<Sect>`
	  `text content`

Picture with Attribution	`<Figure>`
	  `Alt=`*`alt text`*
	`<Sect>`
	  `text content`

Group without Alt Text	*`tags for child objects`*

Hyperlink on Text	`<Link>`
	  `Link - OBJR`
	  `text content`

Hyperlink on Object	*`tag for object`*
	`<Link>`
	  `Link - OBJR`
	  `alt text`
	Note: the <Link> is a sibling of the tag for the object.

Equation (*Version ≥ 16.0.18526.20168*)	`<Formula>`
	  `Alt=`*`alt text`*
	  `Attribute `*`MSFT_MathML`*`=MathML string`

Artifacts

The following types of content are marked as <Artifact> in the PDF Content Tree and have no PDF/UA tags:

- Header and footer
- Decorative graphical objects
- Gray space on the right side of the page for comments
- Pictures in picture bullets
- Underlines
- Borders around text and quote paragraphs
- Lines above footnotes/endnotes
- Text in SmartArt objects

Availability

The information in this article is applicable to the following versions of Word.

- Word for Windows Version 2408 and later.
- Word for Mac Version 16.89 and later.
- Word for iOS Version 2.89 and later.
- Word for Android Build 16.0.18025.XXXXX or later.
- Word for the web Build 16.0.18025.XXXXX or later.

It is available to customers with Office 2024 or Office LTSC 2024 and to customers with a Microsoft 365 subscription on Current Channel or Monthly Enterprise Channel. For customers with a Microsoft 365 subscription on Semi-Annual Enterprise Channel it will be available on January 14, 2025.