[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 17/30 (2006.01)

[12] 发明专利申请公开说明书

[21] 申请号 200480001329.4

[43] 公开日 2006 年 9 月 13 日

[11] 公开号 CN 1833240A

[22] 申请日 2004.7.23
[21] 申请号 200480001329.4
[30] 优先权
[32] 2004. 4.30 [33] US [31] 10/837,041
[86] 国际申请 PCT/US2004/023958 2004.7.23
[87] 国际公布 WO2005/111778 英 2005.11.24
[85] 进入国家阶段日期 2005.5.20
[71] 申请人 微软公司
地址 美国华盛顿州
[72] 发明人 D·奥恩斯坦 A·舒尔
M·J·希尔波格 B·M·琼斯
D·F·埃默森 J·杜尼兹
O·H·弗尔 B·A·麦克肯齐
J·D·潘利 J·波洛克
S·谢斯

[74] 专利代理机构 上海专利商标事务所有限公司
代理人 钱慰民

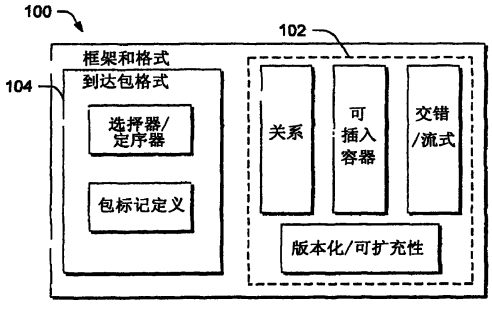
权利要求书 4 页 说明书 103 页 附图 11 页

[54] 发明名称

维护包内部件之间的关系的方法和装置

[57] 摘要

描述了模块化内容框架和文档格式方法和系统。 描述的框架和格式定义了一组构件块，用于组成、包装、分发和呈现以文档为中心的内容。 这些构件块定义了用于文档格式的平台无关框架，使软件和硬件系统能够可靠并一致地生成、交换和显示文档。 该框架和格式用灵活和可扩充的方式来设计。 除该通用框架和格式之外，使用该通用框架定义了一种被称为到达包的特定格式。 到达包格式是用于储存已编页码文档的格式。 到达包的内容可以用各种各样环境内的设备和应用程序之间的完全保真度并跨各种各样情形来显示或打印。



1．一种方法，其特征在于，包括：

创建定义文档的包，其中，所述文档包括构成所述文档的多个部件，并且其中，所述多个部件的每一个具有一相关联的名字；以及

将一关系与所述多个部件的至少一个相关联，其中，所述关系标识了所述相关联的部件和所述包内的至少一个其它部件之间的连接，并且其中，所述相关联的关系储存在一关系部件中，所述关系部件具有从所述相关联部件的名字导出的名字。

2．如权利要求 1 所述的方法，其特征在于，还包括储存关于所述包内至少一个关系的信息。

3．如权利要求 1 所述的方法，其特征在于，还包括：

标识关于与所述包内的部件相关联的关系的信息；以及

储存所标识的信息。

4．如权利要求 1 所述的方法，其特征在于，所述关系标识了有关部件，以及所述相关联的部件和所述有关部件之间的关系的特性。

5．如权利要求 1 所述的方法，其特征在于，所述关系的名字是统一资源标识符。

6．如权利要求 1 所述的方法，其特征在于，还包括通过分析与所述包内的部件相关联的关系发现所述包内部件之间的连接。

7．如权利要求 1 所述的方法，其特征在于，还包括高速缓存关于所述关系的信息。

8．如权利要求 1 所述的方法，其特征在于，所述关系是使用与用于所述关系所连接的部件的模式无关的模式来描述的。

9．如权利要求 1 所述的方法，其特征在于，所述关系被储存，使得它不改变所述相关联部件的任一个的内容。

10．如权利要求 1 所述的方法，其特征在于，所述关系被发送，使得它不改变所述相关联部件的任一个的内容。

11．一个或多个其上具有计算机可读指令的计算机可读介质，当所述指令被执行时，实现权利要求 1 所述的方法。

12．一种包含权利要求 11 所述的计算机可读介质的计算系统。

13．一种方法，其特征在于，包括：

标识包含多个部件的包，其中，所述包定义了文档；

标识与所述多个部件相关联的关系，其中，每一关系定义了第一有关部件、第二有关部件以及所述第一有关部件和所述第二有关部件之间的关系的特性；以及

基于包含在所标识的关系中的信息从所述包检索至少一个部件。

14．如权利要求 13 所述的方法，其特征在于，还包括储存关于与所述多个部件相关联的关系的信息。

15．如权利要求 13 所述的方法，其特征在于，还包括检索与所述包相关联的关系信息。

16．如权利要求 13 所述的方法，其特征在于，还包括通过分析所标识的关系发现所述包内部件之间的连接。

17．一种方法，其特征在于，包括：

创建定义文档的包，其中，所述文档包括构成所述文档的多个部件，并且其中，所述多个部件的每一个具有一相关联的名字；以及

将有关部件的列表与所述多个部件的每一个相关联，其中，所述有关部件的列表标识了所述包内所述相关联的部件和其它部件之间的任何连接，并且其中，所述有关部件的列表具有从所述相关联的部件的名字导出的名字。

18．如权利要求 17 所述的方法，其特征在于，还包括在所述包内储存所述有关部件的列表。

19．如权利要求 17 所述的方法，其特征在于，还包括与所述包分离地储存所述有关部件的列表。

20．如权利要求 17 所述的方法，其特征在于，所述有关部件的列表标识了与所述列表相对应的部件相关联的多个部件。

21．如权利要求 17 所述的方法，其特征在于，还包括通过分析所述有关部件的列表发现所述包内部件之间的连接。

22．如权利要求 17 所述的方法，其特征在于，还包括高速缓存储存在标识所述有关部件列表中的有关部件的部件中的信息。

23．如权利要求 17 所述的方法，其特征在于，所述有关部件列表不改变应用所述有关部件列表应用的源部件，或所述有关部件列表中标识的任何有关部件。

24．一个或多个其上具有计算机可读指令的计算机可读介质，当所述指令被

执行时，实现权利要求 17 所述的方法。

25．一种包含权利要求 24 所述的计算机可读介质的计算系统。

26．一种模式，其特征在于，包括：

标识包内部件之间的关系的至少一个关系元素；

标识每一关系中的目标部件的至少一个属性；以及

标识名字的至少一个属性，其中，所述名字标识了每一关系的目的。

27．如权利要求 26 所述的模式，其特征在于，一附加关系元素将任意数量的关系元素组合在一起。

28．如权利要求 26 所述的模式，其特征在于，所述关系元素不改变所述包内的任何部件。

29．如权利要求 26 所述的模式，其特征在于，多个关系与所述包相关联。

30．如权利要求 26 所述的模式，其特征在于，标识目标部件的所述属性包括统一资源标识符。

31．如权利要求 26 所述的模式，其特征在于，标识名字的所述属性使用统一资源标识符来唯一地定义所述关系的目的。

32．一种在一个或多个计算机可读介质上实施的应用程序接口，其特征在于，包括：

展示创建包的第一方法，其中，所述包将多个部件容纳在一起；

展示创建所述包和多个部件之间的关系的第二方法；以及

展示标识所述多个部件之间的关系的第三方法。

33．如权利要求 32 所述的应用程序接口，其特征在于，还包括删除现有关系的第四方法。

34．如权利要求 32 所述的应用程序接口，其特征在于，还包括检索关于现有关系的信息的第四方法。

35．如权利要求 32 所述的应用程序接口，其特征在于，还包括标识与特定关系相关联的源部件的属性。

36．如权利要求 32 所述的应用程序接口，其特征在于，还包括标识与特定关系相关联的目标统一资源标识符的属性。

37．如权利要求 32 所述的应用程序接口，其特征在于，还包括标识与特定关系相关联的名字的属性。

38．一种在一个或多个计算机可读介质上实施的应用程序接口，其特征在于，

包括：

调用创建包的第一方法，其中，所述包包含多个部件；

调用创建所述包内两个部件之间的关系的第二方法；以及

调用标识所述包内令另一对部件之间的关系的第三方法。

39．如权利要求 38 所述的应用程序接口，其特征在于，还包括对每一关系定义一源部件。

40．如权利要求 38 所述的应用程序接口，其特征在于，还包括对每一关系定义一目标统一资源标识符。

41．如权利要求 38 所述的应用程序接口，其特征在于，还包括定义与每一关系相关联的名字。

维护包内部件之间的关系的方法和装置

技术领域

本发明涉及内容框架、文档格式以及可使用两者的相关方法和系统。

发明背景

当今通常有不同类型的内容框架来表示内容，并且不同类型的文档格式来格式化各种类型的文档。这些框架和格式的每一个常常需要其自己的相关联的软件，以构建、产生、处理或消耗相关联的文档。对于在适当的设备上安装了特定的关联软件的那些人，构建、产生、处理或消耗关联文档并不是一个问题。对于不具有适当软件的那些人，构建、产生、处理或消耗关联的文档通常是不可能的。

针对这一背景，在考虑到文档的产生和消耗的范围内，对这一普遍性有不断的需求。

发明概述

描述了模块化的内容框架和文档格式方法和系统。描述的框架和格式定义了一组构件块，用于组成、包装、分发和呈现以文档为中心的内容。这些构件块定义了一种用于文档格式的平台无关框架，使软件和硬件系统能够可靠并一致地生成、交换和显示文档。该框架和格式是以灵活和可扩充的方式设计的。

除这一通用框架和格式之外，使用该通用框架定义了一种特定的格式，称为到达包（reach package）格式。到达包格式是用于储存已编页码文档的格式。到达包的内容可以用完全的保真度在各种各样的环境中的设备和应用程序之间，并且跨各种各样的情形来显示或打印。

附图的简要描述

图 1 是依照一个实施例的示例性框架和格式的组件的框图。

图 2 是依照一个实施例容纳包括若干部件的文档的示例性包的框图。

图 3 所示是依照一个实施例产生包的示例性书写者以及读取包的示例性阅读

者的框图。

图 4 示出了将三个单独的页面绑定在一起的示例性部件。

图 5 所示是依照一个实施例的示例性选择器，以及被排列以产生包含报表的英语表示和法语表示的财务报表的序列的图示。

图 6 示出了依照一个实施例共同工作以交流包的书写者和阅读者的某些示例。

图 7 示出了文档的多个交错部件的示例。

图 8 和 9 示出了包装图 7 所示的文档的多个部件的不同示例。

图 10 示出了依照一个实施例的示例性到达包以及可构成该包或可在该包中找到的部件的每一个有效类型。

图 11 示出了依照一个实施例从公用语言运行库概念到 XML 的示例性映射。

图 12 示出了依照一个实施例的竖直和横向字形度量。

图 13 示出了依照一个实施例的一对一群集映射。

图 14 示出了依照一个实施例的多对一群集映射。

图 15 示出了依照一个实施例的一对多群集映射。

图 16 示出了依照一个实施例的多对多群集映射。

较佳实施例的详细描述

综述

本文档描述了一种模块化内容框架和文档格式。该框架和格式定义了一组构件块，用于组成、包装、分发和呈现以文档为中心的内容。这些构件块定义了一种用于文档格式的平台无关框架，使软件和硬件系统能够可靠并一致地生成、交换和显示文档。该框架和格式是以灵活和可扩充的方式来设计的。在各种实施例中，对可包括的内容类型、如何呈现内容或构建用于处理内容的客户机的平台没有任何限制。

除这一通用框架之外，使用该通用框架定义了一种特定格式。该格式在本文档中被称为到达包格式，并且是用于储存已编页码或预编页码的文档的格式。到达包的内容可以用完全的保真度在各种各样的环境中的设备和应用程序之间，以及跨各种各样的情形来显示或打印。

下文描述的框架的目标之一是确保独立书写的软件和硬件系统在读取或书写依照下文描述的框架和格式产生的内容时的互操作性。为实现这一互操作性，所描述的格式定义了读取或书写内容的系统必须满足的形式要求。

以下讨论是沿以下线条来组织的，并在两个主要章节中提出一一个名为“框架”，另一个名为“到达包格式”。

名为“框架”的一节提出了一种说明性的包装模型，并描述了构成框架包的各个部件和关系。讨论了关于使用框架包中的描述性元数据的信息，以及映射到物理容器、扩展框架标记的过程，以及对框架版本化机制的使用。

名为“到达包格式”的一节研究了被称为到达包的一个特定类型的框架构建包的结构。该节也描述了对固定的有效负载专用包部件，并定义了一种到达包标记模型和绘制模型。本节以示例性到达标记元素及其属性连同所示的样例一起结束。

作为以下讨论的高级综述，考虑图 1，它一般在 100 示出了本发明的框架和格式的各方面。框架的某些示例性组件在 102 示出，而到达包格式的某些组件在 104 示出。

框架 102 包括示例性组件，包括但不限于，关系组件、可插入容器组件、交错/流组件以及版本化/可扩充性组件，其每一个都在下文更详细地研究。到达包格式 104 包括组件，组件包括选择器/定序器组件以及包标记定义组件。

在以下讨论中，将周期性地回头参考图 1，使得读者可以维持关于所描述的组件适合框架和包格式的那里的观点。

框架

在以下讨论中，提供了对通用框架的描述。各个初级小标题包括“包模型”、“排版部件：选择器和序列”、“描述性元数据”、“物理模型”、“物理映射”、以及“版本化和可扩充性”。每一初级小标题具有一个或多个相关小标题。

包模型

本节描述了包模型，并包括描述包和部件、驱动程序、关系、包关系和起始部件的小标题。

包和部件

在所示和描述的模型中，内容被容纳在包内。包是容纳相关部件的集合的逻辑实体。包的目的是将文档的所有片段（或其它类型的内容）收集到程序员和终端用户易于工作的一个对象。例如，考虑图 2，示出了容纳文档的示例性包 200，文档包括若干部件，部件包括表示文档的 XML 标记部件 202、描述文档中使用的字

体的字体部件 204、描述文档的页面的多个页面部件 206、以及表示文档内的图片的图片部件。表示文档的 XML 标记部件 202 是有利的，因为它可准许容易的可搜索性和参考，而无需对包的整个内容进行语法分析。这将在下文变得显而易见。

贯穿该文档，引入并讨论的阅读者（也称为消费者）和书写者（也称为生产者）的概念。本文档中使用的术语阅读者指的是读取基于模块化内容格式的文件或包的实体。本文档中使用的术语书写者指的是书写基于模块化内容格式的文件或包。作为一个示例，考虑图 3，示出了产生包的书写者和读取包的阅读者。通常，书写者和阅读者被具体化为软件。在至少一个实施例中，与创建和格式化包相关联的大多数处理开销和复杂性被放置在书写者上。这进而从阅读者中消除了大多数处理复杂性和开销，如本领域的技术人员所理解的，这是违背许多现有模型的。这一方面将在下文变得显而易见。

依照至少一个实施例，单个包包含容纳在包内的内容的一个或多个表示。通常，包是单个文件，在本申请中被称为容器。例如，这给予终端用户一种方便的方法来以文档的所有组成片段（图像、字体、数据等）分发其文档。尽管包通常直接对应于单个文件，然而不必要总是如此。包是可以用各种方式来物理地表示的逻辑实体（例如，但不限于，在单个文件中、松散文件的集合、数据库中、通过网络连接的短暂传输等等）。由此，容器容纳包，但是并非所有的包都储存在容器内。

抽象模型与任一物理存储机制无关地描述了包。例如，抽象模型并不涉及“文件”、“流”或与包所位于的物理领域有关的其它物理术语。如下文所讨论的，抽象模型允许用户为各种物理格式、通信协议等创建驱动程序。用类推的方法，当应用程序希望打印图像时，它使用打印机的抽象（由理解特定种类的打印机的驱动程序呈现）。由此，不需要应用程序知道特定的打印设备或如何与打印设备通信。

容器提供了除松散、断开的文件集合之外的许多好处。例如，类似的组成部分可以被聚积，并且内容可以被索引和压缩。另外，组成部分之间的关系可以被识别，并且权限管理、数字签名加密和元数据可以被应用到组成部分。当然，容器可用于并实施上文未具体列出的其它特征。

公用部件属性

在所示并描述的实施例中，部件包括公用属性（如，名字）和字节流。这类似于文件系统中的文件或 HTTP 服务器上的资源。除其内容之外，每一部件具有某些公用部件属性。这包括名字－它是部件的名字，以及内容类型－它是储存在部件

中的内容的类型。部件也可以具有一个或多个相关联的关系，如下文所讨论的。

部件名是在必须在某些方面涉及到部件的任何时刻使用的。在所示和描述的实施例中，名字被组织成分层结构，这类似于文件系统中的路径或 URI 中的路径。以下是部件名的示例：

/document.xml

/tickets/ticket.xml

/images/march/summer.jpeg

/pages/page4.xml

如上文可以见到的，在这一实施例中，部件名具有以下特征：

• 部件名类似于传统文件系统中的文件名。

• 部件名以正斜杠（“/”）开始。

• 与文件系统中的路径或 URI 中的路径一样，部件名可以按照一组类似目录的名字（在以上示例中，为 tickets、images/march）被组织成分层结构。

• 该分层结构包括由正斜杠定界的段。

• 名字的最后一段类似于传统文件系统中的文件名。

重要的是注意，用于命名部件的规则，尤其是可用于部件名的有效字符，对本文档中描述的框架是专用的。这些部件名规则基于互联网标准的 URI 命名规则。依照本实施例，本实施例中用于指定部件名的语法完全与 RFC 2396，同一资源标识符（URI：类属句法）规范的 5（相关 URI 引用）和节 3.3（路径成分）中定义的 abs_path 句法相匹配。

以下附加约束被应用到 abs_path，作为有效的部件名：

• 如节 3（URI 句法分量）和 3.4（查询分量）中定义的查询分量不适用于部件名。

• 如节 4.1（分段标识符）中描述的分段标识符不适应于部件名。

• 任一部件具有通过向现有部件的部件名追加*（“/”分段）来创建的名字是不合法的。

部件名的语法示出如下：

```
part_name     = "/" segment * ( "/" segment )
segment       = *pchar

pchar         = unreserved | escaped |
                ":" | "@" | "&" | "=" | "+" | "$" | ","
unreserved    = alphanum | mark
```

```
escaped      = "%" hex hex
hex          = digit | "A" | "B" | "C" | "D" | "E" | "F" |
               "a" | "b" | "c" | "d" | "e" | "f"
mark         = "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
alpha        = lowalpha | upalpha
lowalpha     = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
               "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
               "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
upalpha      = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
               "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
               "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
digit        = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
               "8" | "9"
alphanum     = alpha | digit
```

可以看到包中所有部件的名字的分段形成了树。这类似于文件系统中所发生的，其中，所有树中的所有非叶节点是文件夹，而叶节点是包含内容的实际文件。名字树中这些类似文件夹的节点（即，非叶节点）服务组织包中的部件的类似功能。然而，重要的是记住，这些“文件夹”仅作为命名分层结构中的概念而存在一它们没有持久格式的其它表现。

部件名不能在“文件夹”级存活。具体地，部件命名分层结构中的非叶节点（“文件夹”）不能包含具有相同名字的部件和子文件夹。

在所述并描述的实施例中，每一部件具有内容类型，它定义了什么类型的内容储存在部件中。内容类型的示例包括：

image/jpeg

text/xml

text/plain; charset="us-ascii"

内容类型在 RFC 2045（多用途互联网邮件扩展；（MIME））中定义的所示的框架中使用。具体地，每一内容类型包括媒体类型（例如，text（文本））、子类型（例如，plain（纯文本））以及关键字＝值形式的可任选参数集（例如，charset="us-ascii"）；多个参数由分号分隔。

部件定址

通常部件包含到其它部件的引用。作为一个简单的示例，想象具有两个部件的容器：标记文件和图像。标记文件希望容纳对图像的引用，使得当处理标记文件时，可识别和定位关联的图像。内容类型和 XML 模式的设计者可使用 URI 来表示这些引用。为使其成为可能，需要定义部件名的领域和 URI 领域之间的映射。

为允许在包内使用 URI，当评估基于包的内容中的 URI 时，必须使用特殊的

URI 解释规则：包本身应当作为 URI 引用的“作者”来处理，而 URI 的路径分量用于导航包内的部件名分层结构。

例如，给定包 URI http://www.example.com/foo/something.package，将对 /abc/bar.xml 的引用解释为意味着名为 /abc/bar.xml 的部件，而非 URI http://www.example.com/abc/bar.xml。

当必须具有从容器中的一个部件到另一个部件的引用时，应当使用相对 URI。使用相对引用允许将容器的内容一起移动到不同的容器（或移动到来自例如文件系统的容器），而不修改部件间的引用。

来自部件的相对引用相对于包含该引用的部件的“基础 URI”来解释。默认地，部件的基础 URI 是部件的名字。

考虑包括具有以下名字的部件的容器：

/markup/page.xml

/image/picture.jpeg

/images/other_picture.jpeg

如果“markup/page.xml”部件包含对“../images/picture.jpeg”的引用，则依照上述规则，该引用必须被解释为涉及部件名“/images/picture.jpeg”。

某些内容类型提供了通过指定内容中的不同基础来覆盖默认基础 URI 的方法。当存在这些覆盖之一时，应当使用明确指定的基础 URI 而非默认的基础 URI。

有时候，“地址”部件中的一部分或特定的点是有用的。在 URI 领域中，使用了分段标识符[见例如 RFC2396]。在容器中，该机制以同样的方式工作。具体地，分段是包含定址的部分的内容类型的背景中理解的额外信息的串。例如，在视频文件中，分段可标识帧，在 XML 文件中，它可通过 xpath 标识 XML 文件的一部分。

分段标识符结合定址部件的 URI 一起使用，来标识定址的部件的分段。分段标识符是可任选的，并且用交叉线（“#”）字符与 URI 分离。由此，它不是 URI 的一部分，但是通常结合 URI 一起使用。

以下描述提供了部件命名的某些指导，因为包和部件命名模型是相当灵活的。该灵活性允许框架包的各种各样应用。然而，重要的是认识到，框架被设计成启用了其中多个不相关软件系统可操作“其自己的”包部件而不会彼此抵触的情形。为允许这一情形，提供了某些方针，如果遵循这些方针，这种情形将成为可能。

此处给出的方针描述了用于最小化或至少减少部件命名冲突的出现，并在它们的确出现时处理它们的机制。创建包内的部件的书写者必须采取措施来检测并处

理与包中现有部件的命名冲突。在发生命名冲突的情况下，书写者可能无法盲目地替换现有部件。

在保证包由单个书写者操纵的情况下，该书写者可与这些方针偏离。然而，如果有多个独立书写者共享一个包的可能性，则所有的书写者必须遵循这些方针。然而，推荐的是所有的书写者在任何情况下都遵循这些方针。

- 要求将部件添加到现有容器的书写者在命名分层结构的新“文件夹”中这样做，而非直接替换根或预先存在的文件夹中的部件。以此方式，名字冲突的可能性被限于部件名的第一段。在该新文件夹内创建的部件可以在没有与现有部件冲突的风险的情况下来命名。
- 在文件夹的“较佳”名字已经被现有部件使用的情况下，书写者必须采用某一策略来选择替换文件夹名字。书写者应当使用向较佳名字追加数字直到找到可用文件夹名的策略（可能在几次不成功迭代之后采用 GUID）。
- 这一政策的一个结果是阅读者不许试图通过“魔法”或“公知”的部件名来定位部件。相反，书写者必须创建与它们所创建的每一文件夹中的至少一个部件的包关系。阅读者必须使用这些包关系来定位部件而非依赖于公知的名字。
- 一旦阅读者找到了文件夹中的至少一个部件（通过上述包关系之一），则它可使用关于该文件夹内的公知部件名的约定来找到其它部件。

驱动程序

此处描述的文件格式可以由不同的应用程序、不同的文档类型等来使用－它们中的许多具有冲突的用途、冲突的格式等等。使用了一个或多个驱动程序来分解各种冲突，例如文件格式中的差异、通信协议中的差异等等。例如，不同的文件格式包括松散文件和复合文件，而不同的通信协议包括 http、网络和无线协议。一组驱动程序将各种文件格式和通信协议抽象成单个模型。可提供多个驱动程序用于不同的情形、不同的消费者要求、不同的物理配置等等。

关系

包中的部件可包含对该包中的其它部件的引用。然而，一般而言，这些引用以对该部件的内容类型专用的方式在引用部件内表示；即，以任意的标记或应用程

序专用编码。这有效地隐藏了来自不理解包含这些引用的部件的内容类型的阅读者的部件之间的内部联接。

即使对于公用内容类型（如到达包一节中描述的固定有效负载标记），阅读者需要对部件中的所有内容进行语法分析，以发现并解析对其它部件的引用。例如，当实现一次打印文档的一页的打印系统时，可能期望识别包含在特定页面中的图片和字体。现有的系统必须对每一页的所有信息进行语法分析，这是耗时的，而且必须理解每一页的语言，这不是某些设备或阅读者的情况（例如，当在去设备的途中通过处理器管线在文档上执行中间处理的设备或阅读者）。相反，此处描述的系统和方法使用了关系来识别部件之间的关系，并描述那些关系的特性。关系语言是简单的，并被定义一次，使得阅读者可以理解关系，而无需多个不同语言的知识。在一个实施例中，关系以 XML 表示为个别的部件。每一部件具有相关联的关系部件，它包含部件是源的关系。

例如，电子表格应用程序使用这一格式并将不同的电子表格储存为部件。不知道关于电子表格语言的应用程序仍能够发现与电子表格相关联的各种关系。例如，应用程序可发现电子表格中的图像以及与电子表格相关联的元数据。提供一个示例关系模式如下：

```
<?xml version="1.0"?>
<xsd:schema xmlns:mmcfrels="http://mmcfrels-PLACEHOLDER"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:attribute name="Target" type="xsd:string"/>
   <xsd:attribute name="Name" type="xsd:string"/>
   <xsd:element name="Relationships">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element ref="Relationship" minOccurs="0" maxOccurs="unbounded"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="Relationship">
      <xsd:complexType>
         <xsd:simpleContent>
            <xsd:extension base="xsd:string">
               <xsd:attribute ref="Target"/>
               <xsd:attribute ref="Name"/>
            </xsd:extension>
         </xsd:simpleContent>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

该模式定义了两个 XML 元素，一个被称为“relationships”，另一个被称为“relationship”。“relationship”元素用于描述此处所描述的单个关系，并且具有

以下属性：（1）“target”，它指示了源部件要关系到的部件，（2）“name”，它指示了关系的类型或特性。定义了“relationships”元素以允许它容纳零个或多个“relationship”元素，并仅用于将这些“relationship”元素搜集在一起成一个单元。

此处所描述的系统和方法引入了更高级机制，以解决称为“关系”的问题。关系提供了表示包中的源部件和目标部件之间的连接种类的额外方法。关系令部件之间的连接可以被直接“发现”，而不需要查看部件中的内容，因此，它们独立于内容专用模式，并且能够更快地被解析。另外，这些关系是协议无关的。各种不同的关系可以与特定的部件相关联。

关系提供了第二个重要的功能：允许部件相关而不需要修改它们。有时候，这一信息担当“注释”的形式，其中，“注释的”部件的内容类型不定义附加给定信息的方法。潜在的示例包括附加的描述性元数据、打印票据和真实注释。最后，某些情况要求信息被特别地附加到现有的部件而不修改该部件一例如，当部件被加密并且不能被解密，或者当部件被数字地签署并且改变它将使签名无效时。在另一示例中，用户可能希望将注释附加到 JPEG 图像文件。JPEG 图像格式当前不提供对识别注释的支持。改变 JPEG 格式以容纳这一用户需求是不实际的。然而，此处讨论的系统和方法允许用户向 JPEG 文件提供注释而不修改 JPEG 图像格式。

在一个实施例中，使用关系部件中的 XML 来表示关系。容器中作为一个或多个关系的源的每一部件具有相关联的关系部件。该关系部件容纳（使用内容类型 application/PLACEHOLDER（应用程序/占位符）以 XML 表达）用于该源部件的关系列表。

下文图 4 示出了环境 400，其中“骨架（spine）”部件 402（类似于固定面板）将三个页面 406、408 和 410 绑定在一起。由骨架绑定在一起的该组页面具有相关联的“打印票据”404。另外，页面 2 具有其自己的打印票据 412。从骨架部件 402 到其打印票据 404 的连接以及从页面 2 到其打印票据 412 的连接使用关系来表示。在图 4 的排列中，骨架部件 402 具有相关联的关系部件，它包含将骨架连接到票据 1 的关系，在以下示例中示出。

```
<Relationships xmlns=”http://mmcfrels-PLACEHOLDER”>
    <Relationship
            Target=”../tickets/ticket1.xml”
            Name=”http://mmcf-printing-ticket/PLACEHOLDER”/>
```

</Relationships>

关系使用单个<Relationships>元素中嵌套的<Relationship>元素来表示。这些元素在 http://mmcfrels(PLACEHOLDER)名字空间中定义。见上文的示例模式，以及下文对示例关系的相关讨论。

关系元素具有以下附加属性：

属性	需要	意义
Target（目标）	是	指向关系的另一端处的部件的 URI。相对 URI 必须相对于源部件来解释。
Name（名称）	是	唯一地定义关系的角色或目的的绝对 URI。

名字属性不必要是实际的地址。不同类型的关系按其名字来标识。这些名字以对 XML 名字空间定义名字空间的同一方式来定义。具体地，通过使用模仿因特网域名空间的名字，非协调方可安全地创建不冲突的关系名字－正如它们可以对 XML 名字空间所做的一样。

不允许关系部件参与其它关系。然而，在所有其它意义上它是第一类部件（例如，它是可 URI 定址的、它可以被打开、读取、删除等等）。关系通常不指向包外部的内容。用于标识关系目标的 URI 一般不包括 URI 模式。

部件及其相关联的关系部件通过命名约定来连接。在这一示例中，骨架的关系部件将储存在/content/_rels/spine.xml.rels 中，而页面 2 的关系将储存在/content/_rels/p2.xml.rels 中。注意，此处使用了两个特殊的命名约定。首先，某些名字分层结构中给定“文件夹”中的（其它）部件的关系部件储存在名为_rels 的“子文件夹”中（以标识关系）。其次，这一关系容纳部件的名字通过向原始部件的名字追加.rels 扩展名来形成。在特定的实施例中，关系部件是内容类型 application/xml+relationshipsPLACEHOLDER（应用程序/xml+关系占位符）。

关系表示了两个部件之间的有向连接。由于表示关系的方式，从其源部件开始遍历关系是有效的（因为找出任何给定部件的关系部件是平凡的）。然而，从关系的目标开始反向遍历关系不是有效的（因为找出对部件的所有关系的方法是浏览容器中的所有关系）。

为使关系的反向遍历成为可能，使用了新关系来表示其它（可遍历）方向。这是关系类型的设计者可以使用的一种建模技术。继续以上示例，如果重要的是能够找出附加了票据 1 的骨架，则将使用从票据连接到骨架的第二关系，如：

在 content/_rels/p1.xml.rels 中：

```
<Relationships xmlns="http://mmcfrels-PLACEHOLDER">
<Relationship
    Target="/content/spine.xml"
    Name="http://mmcf-printing-spine/PLACEHOLDER"/>
</Relationships>
```

包关系

“包关系”用于找出包中公知的部件。该方法避免了依赖于用于找出包中的部件的命名约定，并确保了在不同有效负载中的相同部件名之间没有抵触。

包关系是一种特殊的关系，其目标是部件，而源不是部件：源是整个包。具有“公知”的部件实际上是具有有助于找出该部件的“公知”关系名字。这能够起作用，因为有定义良好的机制以允许由非协调方来命名关系，而某些实施例不包含这样的部件名机制－那些实施例被限于一组方针。包关系在包关系部件中找到，并且使用关系部件的标准命名约定来命名。由此，它被命名为“/_rels/.rels”。

该包关系中的关系部件在找出公知部件时是有用的。

起始部件

包级的公知部件的一个示例是包“起始”部件。这是通常在打开包时被处理的部件。它表示了储存在包内的文档内容的逻辑根。包的起始部件通过以下公知包关系来定位。在一个示例中，该关系具有以下名字：http://mccf-start-part-PLACEHOLDER。

排版部件：选择器和序列

所描述的框架定义了用于从部件构建更高阶结构的两种机制：选择器和序列。

选择器是在多个其它部件之间“选择”的部件。例如，选择器部件可以在表示文档的英语版本的部件和表示文档的法语版本的部件之间“选择”。序列是对多个其它部件“定序”的部件。例如，序列部件可以组合两个部件（成一个线性序列），其中一个表示五页文档，而另一个表示十页文档。

这两种类型的排版部件（序列和选择器）以及用于汇编它们的规则构成了组成模型。排版部件可组成其它排版部件，因此例如，可具有在两个组成部分之间进

行选择的选择器。作为一个示例，考虑图 5，它示出了包含英语表示和法语表示的财务报表的一个示例。这些表示的每一个进一步包括介绍（封面）以及其后的财务（电子表格）。在这一示例中，选择器 500 在报表的英语和法语表示之间进行选择。如果选择了英语表示，则序列 502 将英语介绍部件 506 与英语财务部件 508 定序。或者，如果选择了法语部分，则序列 504 将法语介绍部件 510 与法语财务部件 512 定序。

排版部件 XML

在所述并描述的实施例中，排版部件是使用少数 XML 元素来描述的，它们都取自公用的组成部分名字空间。作为一个示例，考虑下列：

元素：<selection>

属性：无

允许的子元素：<item>

元素：<sequence>

属性：无

允许的子元素：<item>

元素：<item>

属性：Target一组成部分中的部件的部件名

作为一个示例，以下是上文图 5 的示例的 XML：

MainDocument.XML

```
<selection>
  <item target="EnglishRollup.xml" />
  <item target="FrenchRollup.xml" />
</selection>
```

EnglishRollup.XML

```
<sequence>
  <item target="EnglishIntroduction.xml" />
  <item target="EnglishFinancials.xml" />
</sequence>
```

FrenchRollup.XML

```
<sequence>
<item target="FrenchIntroduction.xml">
<item target="FrenchFinancials.xml">
</sequence>
```

在这一XML中，MainDocument.xml表示包中的整个部件，并且根据"selection"标签，指示了要在由"item"标签封装的不同项，即"EnglishRollup.xml"和"FrenchRollup.xml"之间作出选择。

根据"sequence"标签，EnglishRollup.xml和FrenchRollup.xml是将由其各自的"item"标签封装的各自的项定序在一起的序列。

由此，提供了简单的XML语法来描述选择器和序列。这一组成块中的每一部件被构建，并执行一个操作－选择或定序。通过使用部件的分层结构，可构建选择和序列的不同健壮集合。

组成块

包的组成块包括可从包的起始部件到达的所有排版部件（选择器或序列）的集合。如果包的起始部件既非选择器又非序列，则认为该组成块为空。如果起始部件是排版部件，则递归地遍历那些排版部件中的子<item>，以产生排版部件的有向非循环图（当遇到非排版部件时停止遍历）。该图是组成块（依照本发明，它必须是非循环的，以使包有效）。

确定组成部分语义

建立了上述相对直接的XML语法之后，以下讨论描述了表示信息使得可基于内容类型作出选择的方法。即，上述XML提供了足够的信息，以允许阅读者定位被汇编成一个组成部分的部件，但是未提供足够的信息来帮助阅读者知道关于组成部分的特性的更多信息。例如，给定组成两个部件的选择，阅读者如何知道基于什么基准（例如，语言、纸张大小等）来作出选择？答复是这些规则与排版部件的内容类型相关联。由此，用于基于语言在表示之间进行选取的选择器部件具有与基于纸张大小在表示之间进行选取的选择器部件不同的关联内容类型。

通用框架为这些内容类型定义了通用形式：

Application/XML+Selector-SOMETHING（应用程序/XML+选择器-某一内容）

Application/XML+Sequence-SOMETHING（应用程序/XML+序列-某一内容）

这些内容类型中的 SOMETHING 由指示选择或序列的特性的词来替换，例如，页面大小、颜色、语言、阅读者设备上的常驻软件等等。因此，在此框架中，可发明所有种类的选择器和序列，并且每一个可具有非常不同的语义。

描述的框架也定义了以下所有阅读者或阅读设备必须理解的选择器和序列的公知内容类型。

内容类型	规则
Application/XML+Selector+SupportedContentType（应用程序/XML+选择器+支持的内容类型）	基于其内容类型在项之间选取。选择理解给定内容类型的软件对其可用的第一项。

作为一个示例，考虑如下。假定包包含具有页面的文件，并且在页面的中间有要出现视频的区域。在此示例中，页面的视频部件可包括 Quicktime 视频形式的视频。这一情形的一个问题是 Quicktime 视频不是被普遍理解的。然而，假定依照本框架，尤其是下文描述的到达包格式，存在被普遍理解的图像格式－JPEG。当产生包含上述文档的包时，除将视频定义为包的部件之外，生产者还为页面定义了 JPEG 图像，并提出了一种 SupportedContentType（支持的内容类型）选择器，使得如果用户的计算机具有理解 Quicktime 视频的软件，则选择 Quicktime 视频，否则选择 JPEG 图像。

由此，如上所述，框架级选择器和序列组件允许构建健壮的分层结构，在此示例中，它是以 XML 定义的。另外，存在定义明确的方法，确定使用内容类型时选择器和序列的行为。另外，依照一个实施例，通用框架包括一个特定的内容类型，它是预定义的，并允许基于消费者（例如，阅读者或阅读设备）理解什么和不理解什么来处理和使用包。

可使用类似的规则定义其它排版部件内容类型，其示例在下文描述。

描述性元数据

依照一个实施例，描述性元数据部件向包的书写者或生产者提供了储存属性值的方法，它使包的阅读者能够可靠地发现该值。这些属性通常用于记录关于整个包以及容器内的个别部件的附加信息。例如，包内的描述性元数据部件可容纳诸如包作者、关键字、概要等的信息。

在所示并描述的实施例中，描述性元数据以 XML 来表达、被储存在公知内容类型的部件中、并可使用公知关系类型来找到。

描述性元数据容纳元数据属性。元数据属性由属性名和一个或多个属性值来表示。属性值具有简单的数据类型，因此每一数据类型由单个 XML qname 来描述。描述性元数据属性具有简单类型的事实并不意味着不能在包内用复杂的 XML 类型来储存数据。在这一情况下，必须将信息作为完整的 XML 部件来储存。当完成时，删除关于仅使用简单类型的所有约束，但是“平面”描述性元数据属性模型的简单性丢失。

除用于定义属性集的通用机制以外，有一种特定的、定义良好的、使用该机制储存的文档核心属性集。这些文档核心属性通常用于描述文档，并包括如标题、关键字、作者等的属性。

最后，容纳这些文档核心属性的元数据部件也可容纳除文档核心属性之外的附加、用户定义的属性。

元数据格式

依照本发明，描述性元数据具有内容类型，并且依照以下规则的关系达到目标：

描述性元数据发现规则	使用用户定义的属性	使用文档核心属性
描述性元数据部件的内容类型必须是：	application/xml-SimpleTypeProperties-PLACEHOLDER（应用程序/xml-简单类型属性-占位符）	
可具有以描述性元数据部件关系为目标的源部件的内容类型可以是：	任何	任何
以描述性元数据部件为目标的关系的名字可以是：	*用户定义的 Uri 名字空间*	http://mccf-DocumentCore-PLACEHOLDER（http://mccf-文档核心-占位符）
可以附加到源部件的描述性元数据部件的数量可以是：	无限	0 或 1

可附加相同的描述性元数据部件的源部件的数量可以是：	无限	无限

以下 XML 模式用于依照一个实施例表现描述性元数据。关于标记的每一分量的细节在样例后的表格中给出。

```
<mcs:properties xmlns:mcs="http://mmcf-core-services/PLACEHOLDER"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <mcs:property prns:name = "property name" xmlns:prns="property namespace"
            mcs:type="datatype"
            mcs:multivalued="true |false">
        <mcs:value> … value … </mcs:value>
    </mcs:property>
</mcs:properties>
```

标记分量	描述
xmlns:mcs="http://mccf-common-services/PLACEHOLDER"	定义了 MMCF 公用服务名字空间
xmlns:xsd="http://www.w3.org/2001/XMLSchema"	定义了 XML 模式名字空间。许多用户定义的属性和文档核心属性的主要部分内建了使用 XSD 定义的数据类型。尽管每一属性可具有其自己的名字空间，然而 XSD 名字空间被放置在描述性元数据 XML 的根上。
mcs:properties	描述性元数据 XML 的根元素
mcs:property	属性元素。属性元素容纳属性 qname 和值。可以有无限数量的属性元素。属性元素被认为是根元素的直接子元素。
xmlns:prns	属性名字空间：对于文档核心属性，它是 http://mmcf-DocumentCore-PLACEHOLDER 。对于用户定义的属性，它是自定义名字空间。
prns:name	属性名：容纳属性名的串属性
mcs:type="datatype"	类型是容纳属性数据类型定义的串属性，例如，xsd:string
mcs:value	该分量指定了属性的值。值元素是属性元素的直接子元素。如果 mcd:multivalued="true"，则

	可以有无限数量的值元素。

文档核心属性

以下是包括属性名、属性类型和描述的文档核心属性的表格。

名字	类型		描述
Commnets	string、可任选、单值		对作者包括的整个文档的评论。这可以是文档的概要。
Copyright	string、可任选、单值		该文档的版权串
EditingTime	int64、可任选、单值		编辑该文档所花费的时间，用秒表示。由应用程序逻辑设置。该值必须具有适当的类型。
IsCurrentVersion	boolean、可任选、单值		指示该实例是文档的当前版本还是旧版本。该字段可以从VersionHistory导出，但是导出过程可能是昂贵的。
Language	keyword(=string256)、可任选、多值		文档的语言（英语、法语等）。该字段由应用程序逻辑设置。
RevisionNumber	string、可任选、单值		文档的修订。
Subtitle	string、可任选、单值		文档的次级或解释性标题
TextDataProperties	TextDataProperties、可任选、单值		如果该文档具有文本，则该属性定义了文档的文本属性的集合，如段落计数、行计数等。
	CharacterCount	int64	
	LineCount	int64	
	PageCount	int64	
	ParagraphCount	int64	
	WordCount	int64	
TimeLastPrinted	datetime、可任选、单值		最后一次打印该文档的日期和时间
Title	string、可任选、单值		处理文档的应用程序理解的文档标题。这不同于包含包的文件的名字。

TitleSortOrder	string、可任选、单值	标题的分类次序（例如，“The Beatles”将具有分类次序“Beatles”，没有前导的“The”）。
ContentType	Keyword(=string256)、可任选、多值	由应用程序逻辑设置的文档类型。此处储存的类型应当是识别的“MIME类型（mime-type）”。该属性对于某些类型的文档的归类或搜索可以是有用的。

物理模型

物理模型定义了书写者和阅读者使用包的各种方法。该模型基于三个组件：书写者、阅读器以及它们之间的管道。图6示出了共同工作以交流包的书写者和阅读者的某一示例。

管道将数据从书写者传送到阅读者。在许多情况下，管道可以仅包括阅读者作出的从本地文件系统读取包的API调用。这被称为直接访问。

然而，通常阅读者和书写者必须通过某一类型的协议彼此通信。例如，这一通信可跨进程边界或在服务器和台式计算机之间发生。这被称为联网访问，并且由于管道的通信特征（尤其是速度和请求等待时间），它是重要的。

为允许最大的性能。物理包设计必须考虑三个重要领域内的支持：访问样式、布局样式和通信样式。

访问样式

流式消耗

由于书写者和阅读者之间使用联网访问的通信不是瞬时的，因此重要的是允许包的渐进创建和消耗。具体地，依照本实施例，推荐任何物理包格式被设计成允许阅读者在包的所有比特通过管道传送之前，当它接收到的数据（例如，部件）时开始解释并处理该数据。这一能力被成为流消耗。

流式创建

当书写者开始创建包时，它并不总是知道它将会在包中放入什么。作为一个示例，当应用程序开始构建打印池文件包时，它可能不知道有多少页需要被放入包

中。作为另一示例，服务器上动态生成报表的程序可能无法认识到该报表有多长，或者报表具有多少图片一直到它完全生成了该报表。为允许这类书写者，物理包应当允许书写者在已经添加了其它部件之后动态地添加部件（例如，书写者必须不被要求在开始书写时预先规定它将创建多少部件）。另外，物理包应当允许书写者在不知道该部件的最终长度的情况下开始书写部件的内容。这些要求共同启用了流式创建。

同时创建和消耗

在高管线化的体系结构中，流式创建和流式消耗可对特定的包同时发生。当设计物理包时，支持流式创建和支持流式消耗可在相反的方向推动设计。然而，找出支持两者的设计通常是可能的。由于管线化体系结构中的好处，推荐物理包支持同时创建和消耗。

布局样式

物理包容纳部件的集合。这些部件可以用两种样式之一来布局：简单排序和交错。采用简单排序，包内的部件用定义的排序来布局。当以纯线性方式传送这样的包时，从包内的第一字节开始直到最后一个字节，第一部件的所有字节首先到达，然后第二部件的所有字节到达，依此类推。

采用交错布局，多个部件的字节是交错的，这在某些情况下允许改进的性能。从交错中显著获益的两个情况是多媒体回放（例如，同时传送视频和音频）和内嵌资源引用（例如，标记文件的中间对图像的引用）。

交错通过用于组织交错部件的内容的特殊约定来处理。通过将部件分割成片段并交错这些片段，可能达到期望的交错结果，同时仍可能容易地重建原始的较大部件。为理解交错是如何工作的，图 7 示出了涉及两个部件的简单示例：content.xml 702 和 image.jpeg 704。第一部件 content.xml 描述了页的内容，并且在该页的中间是对应当出现在该页上的图像（image.jpeg）的引用。

为理解交错为何是有价值的，考虑如何使用简单排序在包内排列这些部件，如图 8 所示。处理该包（并且顺序地接收字节）的阅读者在接收到所有的 content.xml 部件以及 image.jpeg 之前将无法显示图片。在某些情况下（例如，小或简单的包，或快速通信链路）这可能不是问题。在其它情况下（例如，如果 content.xml 非常大或者通信链路非常慢），则需要读完所有的 content.xml 部件来获得图像将导致

不可接受的性能或对阅读者系统施加不合理的存储器需求。

为更接近理想的性能，能够分割 content.xml 部件并将 image.jpeg 部件插入到中间，紧接着引用图片之处将是较佳的。这允许阅读者更早地开始处理图像：一旦它遇到引用，图像数据就在后面。例如，这将产生图 9 所示的包布局。由于性能利益，通常期望物理包支持交错。根据使用的物理包的类型，可以支持或不支持交错。不同的物理包可不同地处理交错的内部表示。不管物理包如何处理交错，重要的是记住，交错是在物理层上出现的优化，并且被分割成物理文件中的多个片段的部件仍是一个逻辑部件；片段本身不是部件。

通信样式

书写者和阅读者之间的通信可以基于部件的顺序传送或按对部件的随机访问来进行，从而允许它们不按顺序来访问。使用这些通信样式的哪一个取决于管道和物理包格式的能力。一般而言，所有的管道支持顺序传送。物理包必须支持顺序传送。为支持随机访问情形，使用的管道和物理包都必须支持随机访问。某些管道基于允许随机访问的协议（例如，具有字节范围支持的 HTTP 1.1）。为在使用这些管道时允许最大性能，推荐物理包支持随机访问。在缺少这一支持的情况下，阅读者将简单地等待，直到它们需要的部件被顺序地传送。

物理映射

逻辑包装模型定义了包抽象；包的实际示例基于包的某一特定物理表示。包装模型可以被映射到物理持久格式以及各种传输（例如，基于网络的协议）。物理包格式可以被描述为从抽象包装模型的分量到特定的物理格式特征的映射。包装模型未指定应当使用哪些物理包格式来归档、分发或假脱机操作包。在一个实施例中，仅指定了逻辑结构。包可以由松散文件的集合、.ZIP 文件档案、复合文件或某一其它格式 “物理地”实施。所选择的格式由目标消耗设备支持，或由设备的驱动程序支持。

被映射的分量

每一物理包格式为以下分量定义了映射。某些分量是可任选的，并且特定的物理包格式可能不支持这些可任选的分量。

	分量	描述	需要或可任选

部件	Name	命名部件	需要
	ContentType	标识储存在部件中的内容的种类	需要
	PartContents	储存部件的实际内容	需要

公用映射模式

访问样式	StreamingConsumption	允许阅读者在整个包到达之前开始处理部件	可任选
	StreamingCreation	允许书写者在事先不知道将书写的所有部件的情况下将部件写到包	可任选
	SimultaneousCreationand Consumption	允许流式创建和流式消耗在同一包上同一时刻发生	可任选
布局样式	SimpleOrdering	包中部件N的所有字节在部件N+1的字节之前出现	可任选
	Interleaved	多个部件的字节交错	可任选
通信样式	SequentialDelivery	部件 N 的所有内容在部件N+1之前被传送到阅读者	可任选
	Random-Access	阅读者可不以顺序的次序请求部件的传送	可任选

存在许多物理存储格式，其特征部分地与包装模型分量相匹配。在定义从包装模型到这类存储格式的映射时，可能期望利用包装模型和物理存储介质之间的任何能力相似性，同时使用映射的层来提供物理存储介质中不固有存在的附加能力。例如，某些物理包格式可将个别的部件储存为文件系统中的个别文件。以这一物理格式，将许多部件名直接映射到相同的物理文件名字是自然的。使用不是有效文件系统文件名的字符的部件名可能需要某种类型的转义机制。

在许多情况下，不同物理包格式的设计者可能面对单个公共的映射问题。公共映射问题的两个示例是在将任意的内容类型与部件相关联时，以及当支持交错布局样式时产生的。本说明书建议了对这类公共映射问题的公用解决方案。特定物理包格式的设计者可以被鼓励，但并不要求来使用此处定义的公用映射解决方案。

标识部件的内容类型

物理包格式映射定义了用于储存每一部件的内容类型的机制。某些物理包格式具有用于表现内容类型的本机机制（例如，MIME 中的“Content-Type”标题）。对于这些物理包，推荐映射使用本机机制来表示部件的内容类型。对于其它物理格式，使用某些其它机制来表示内容类型。推荐的用于表示这些包中的内容类型的机制是通过在包内包括特别命名的 XML 流，称为类型流。该流不是部件，并且因此其本身不是 URI 可定址的。然而，它可以在物理包内使用用于交错部件的相同机制来交错。

类型流包含具有顶级“Types”元素以及一个或多个“Default”和“Overide”子元素的 XML。“Default”元素定义了从部件名扩展名到内容类型的默认映射。这利用了文件扩展名通常对应于内容类型的事实。“Override”元素用于指定不被默认映射覆盖或不与默认映射一致的部件上的内容类型。包书写者可使用“Default”元素来减少每一部件“Override”元素的数量，但是不必要如此。

“Default”元素具有以下属性：

名字	描述	需要
Extension	部件名扩展名。“Default”元素与其名字以句点以及其后的这一属性值结束的任何部件相匹配。	是
ContentType	RFC2045 中定义的内容类型。指示任何匹配部件的内容类型（除非被“Override”元素覆盖；见下文）。	是

“Override”元素具有以下属性：

名字	描述	需要
PartName	部件名 URI。“Override”元素与其名字等于该属性值的部件相匹配。	是
ContentType	RFC2045 中定义的内容类型。指示匹配部件的内容类型。	是

以下是包含在类型流中的 XML 的一个示例：

```
<Types xmlns="http://mmcfcontent-PLACEHOLDER">
  <Default Extension="txt" ContentType="plain/text" />
  <Default Extension="jpeg" ContentType="image/jpeg" />
  <Default Extension="picture" ContentType="image/gif" />
  <Override PartName="/a/b/sample4.picture"
ContentType="image/jpeg" />
</Types>
```

以下表格示出了部件及其由以上类型流定义的对应内容类型的示例列表：

部件名	内容类型
/a/b/sample1.txt	纯文本/文本
/a/b/sample2.jpeg	图像/jpeg
/a/b/sample3.picture	图像/gif
/a/b/sample4.picture	图像/jpeg

对于包内的每一部件，类型流包含以下的任一个：（a）一个匹配的“Default”元素，（b）一个匹配的“Override”元素，或者（c）匹配的“Default”元素和匹配的“Override”元素两者（在这一情况下，“Override”元素有优先级）。一般而言，对任一给定的扩展名，最多有一个“Default”元素，并且对任一给定的部件名，最多有一个“Override”元素。

类型流中“Default”和“Override”元素的顺序不是重要的。然而，在交错包中，“Default”和“Override”元素在物理包中出现在它们对应的部件之前。

交错

并非所有的物理包都本机地支持部件的数据流的交错。在一个实施例中，到任一这样的物理包的映射使用了本节中描述的通用机制以允许部件的交错。通用机制通过将部件的数据流分割成可与其它部件的片段或整个部件交错的多个片段来工作。部件的个别片段在物理映射中存在，并且在逻辑包装模型中不是可定址的。分段可具有零长度。

定义了以下从部件名到部件的个别片段的名字的唯一映射，使得阅读者可以按其原始的顺序将片段缝合在一起以形成部件的数据流。

用于导出给定部件名的片段名的语法：

```
piece_name = part_name "/" "[" 1*digit "]" [ ".last" ] ".piece"
```

以下有效性约束对由该语法生成的 piece_names 存在：

• 片段号以 0 开始，并且是正的、连续的整数。片段号可以被左零填充（left-zero-padded）。

• 部件的片段集的最后一个片段包含片段名中“.piece”之前的“.last”。

• 片段名在映射到物理包中的名字之前从逻辑部件的名字生成。

尽管不必要以其自然顺序储存片段，但是这样的存储可提供优化的效率。包含交错（分段）的部件的物理包也可包含非交错（单片段）部件，因此以下示例是有效的：

```
spine.xaml/[0].piece
pages/page0.xaml
spine.xaml/[1].piece
pages/page1.xaml
spine.xaml/[2].last.piece
pages/page2.xaml
```

特定映射

下文定义了对以下物理格式的特定映射：Windows 文件系统中的松散文件。

到 Windows 文件系统中的松散文件的映射

为更好地理解如何将逻辑模型的元素映射到物理格式，考虑将 Metro 包表示为 Windows 文件系统中的松散文件的集合的基本情况。逻辑包中的每一部件将包含在单独的文件（流）内。逻辑模型中的每一部件名对应于文件名。

逻辑分量	物理表示
部件	文件
部件名	具有路径的文件名（应当看似 URI、将斜杠改为反斜杠等等）。
部件内容类型	包含表达文件名及其相关联的类型的简单列表的 XML

部件名被转换成有效的 Windows 文件名，如以下表格中示出的。

下文给出的是两个字符集，它们对于逻辑部件名段（URI 段）和 Windows 文件名是有效的。该表格展示了两个重要的内容：

存在两个有效的 URI 符号，冒号（:）和星号（*），在将 URI 转化成文件名时需要对这两个符号转义。

存在有效的文件名符号^ {} [] #，它们不能以 URI 呈现（它们可用于特殊的映射目的，如交错）。

“转义”用作在部件名包含不能在文件名中使用的字符时产生有效文件名字符的技术。为对字符转义，使用了脱字符号（^），其后跟随字符的十六进制表示。

为从 abs_path（部件名）映射到文件名：

首先删除/

将所有的/转化成\

对冒号和星号字符转义

例如，部件名/a:b/c/d*.xaml 变为以下文件名 a^25b\c\d^2a.xaml。

为执行反向映射：

将所有的\转化成/

向串的起始添加/

通过用对应的字符替换^[十六进制代码]对字符解转义（unescape）

来自 URI 语法规则（RFC2396）	对命名文件、文件夹或快捷方式有效的字符
path_segments=segment*("/" segment) segment=*pchar*(";" param) param=*pchar pchar=unreserved | escaped |":" | "@" | "&" | "=" | "+" | "$" | "," unreserved=alphanum|mark alphanum=alpha|digit mark="-" | "_" | "." | "!" | "~" | "*" | "" | "(" | ")" escaped="%" hex hex hex=digit|"A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f"	字母数字|^重音抑扬符号（脱字符号） & “和”符号 ' 撇号（单引号） @ At 符号 { 左花括号 } 右花括号 [开括号] 闭括号 , 逗号 $ 美元符号 = 等号 ! 感叹号 - 连字号 # 数字符号 (开圆括号) 闭圆括号

	% 百分号 . 句号 + 加号 ~ 代字号 _ 下划线

版本化和可扩充性

与其它技术规范一样，此处包含的规范可用未来的增强来进展。这一规范的第一版的设计包括用于基于第一版书写的软件系统以及为未来版本书写的软件系统之间的文档的未来互换的计划。类似地，该规范允许第三方创建对该规范的扩展。这一扩展可以例如允许构造充分利用某一特定打印机的特征的文档，同时仍维持与不知道该打印机的存在的其它阅读者的兼容性。

使用固定有效负载标记的新版本或对标记的第三方扩展的文档要求阅读者进行关于行为（例如，如何可视地呈现某些内容）的适当判断。为指导阅读者，文档的作者（或生成该文档的工具）应当为遇到其它未识别元素或属性的阅读者标识适当的行为。对于到达文档，这一类型的指导是重要的。

新的打印机、浏览器和其它客户机可实现对未来特征的各种支持。利用新版本或扩展的文档作者必须仔细地考虑不知道那些扩展版本的阅读者的行为。

版本化名字空间

XML 标记识别基于名字空间 URL。对于任一 XML 名字空间，期望阅读者识别所有的 XML 元素以及该名字空间中定义的 XML 属性，或不识别任何东西。如果阅读者不识别新的名字空间，则阅读者需要执行文档中指定的后退呈现操作。

XML 名字空间 URI “http://PLACEHOLDER/version-control” 包括用于构造固定有效负载标记的 XML 元素和属性，该标记是版本自适应和扩展自适应的。固定有效负载不需要在其中具有版本化元素。然而，为构建自适应的内容，必须使用<ver:Compatibility.Rules>和<ver:AlternativeContent>XML 元素的至少一个。

这一固定有效负载标记规范具有与其相关联的 xmlns URI：“http://PLACEHOLDER/pdl”。在固定有效负载中使用该名字空间将指示阅读者应用程序，仅使用该规范中定义的元素。该规范的未来版本将具有其自己的名字空间。熟悉新名字空间的阅读者应用程序将知道如何支持先前的版本中定义的元素或

属性的超集。不熟悉新版本的阅读者应用程序将新版本的 URI 考虑成它是对 PDL 的某一未知扩展的 URL 一样。这些应用程序可能不知道在名字空间之间存在关系，即一个是另一个的超集。

后向和“前向”兼容性

在支持此处所讨论的系统和方法的应用程序或设备的环境中，兼容性由客户机对使用规范的先前版本或规范的未知扩展或版本创作的文档进行语法分析和显示的能力来指示。各种版本化机制解决了“后向兼容性”，允许客户机的未来实现能够支持基于规范的下级版本的文档，如下文所示出的。

当诸如打印机等已实现的客户机接受到使用标记语言的未来版本构建的文档时，客户机能够对可用呈现选项进行语法分析并理解它们。依照规范的较旧版本书写的客户机软件使用较新版本特征处理某些文档的能力通常被称为“前向兼容性”。被书写成允许前向兼容性的文档被描述为“版本自适应的”。

此外，由于已实现的客户机也需要能够支持具有表示新元素或属性的未知扩展的文档，因此各种语义支持“扩展自适应的”文档的更一般情况。

如果打印机或察看器遇到未知的扩展，它将查找连同扩展的使用一起嵌入的信息，以找出关于自适应地呈现环绕的内容的指导。这一自适应涉及用理解的内容替换未知的元素或属性。然而，自适应可采用其它形式，包括纯忽略未知内容。在缺乏明确指导的情况下，阅读者应当将标记中未识别的扩展的存在视为错误条件。如果未提供指导，则假定扩展对理解内容是基本的。呈现失败将被捕捉并报告给用户。

为支持这一模型，标记语言的新的和扩展的版本应当逻辑上组合名字空间中的相关扩展。以此方式，文档作者能够使用最少数量的名字空间来利用扩展的特征。

版本化标记

用于支持扩展自适应行为的 XML 词汇包括以下元素：

版本化元素和分层结构	描述
<Compatibility.Rules>	控制语法分析器如何对未知元素或属性反应。
<Ignorable>	声明相关联的 URI 是可忽略的。
<ProcessContent>	声明如果元素被忽略，则元素的内容将如同它被忽略

	的元素的容器包含那样被处理。
<CarryAlong>	向文档编辑工具指示当修改文档时是否应当保存可忽略的内容。
<MustUnderstand>	倒转被声明为可忽略的元素的效果。
<AlternateContent>	在利用版本化/扩展特征的标记中，<AlternateContent>元素关联由不能处理被指定为Prefer 的标记的阅读者应用程序使用的替代“fallback”标记。
<Prefer>	指定较佳的内容。该内容是客户机知道的版本/扩展特征。
<Fallback>	对于下级客户机，指定了要被较佳内容替代的“下级”内容。

<Compatibility.Rules>元素

Compatibility.Rules 可以被附加到可容纳附加的属性的任一元素上，如附加到 Xaml 根元素上。<Compatibility.Rules>元素控制语法分析器如何对未知元素或属性反应。通常这些项被报告为错误。向 Compatibility.Rules 属性添加 Ignorable 元素通知了编译器来自某些名字空间的项可被忽略。

Compatibility.Rules 可包含元素 Ignorable 和 MustUnderstand。默认地，所有的元素和属性被假定为 MustUnderstand。元素和属性可以通过将 Ignorable 元素添加到其容器的 Compatibility.Rules 属性中来变成 Ignorable。元素或属性可以通过将 MustUnderstand 元素添加到嵌套容器之一再一次变成 MustUnderstand。一个 Ignorable 或 MustUnderstand 指同一 Compatibility.Rules 元素中的特定名字空间 URI。

<Compatibility.Rules>元素影响容器的内容，而非容器自己的标签或属性。为影响容器的标签或属性，其容器必须包含兼容性规则。Xaml 根元素可用于为将另外成为根元素的元素，如 Canvas 指定兼容性规则。Compatibility.Rules 复合属性是容器中的第一元素。

<Ignorable>元素

<Ignorable>元素声明了所包含的名字空间 URI 是可忽略的。如果在当前块或容器块中在项的前方声明了<Ignorable>标签，并且名字空间 URI 对语法分析器是未知的，则该项可认为是可忽略的。如果 URI 已知，则可不予处理 Ignorable 标签，并且所有的项都是理解的。在一个实施例中，未被明确声明为 Ignorable 的所有项必须被理解。Ignorable 元素可包含<ProcessContent>和<CarryAlong>元素，它们用于修改如何忽略元素以及向文档编辑工具给予应当如何在编辑的文档中保存这些内容的指导。

<ProcessContent>元素

<ProcessContent>元素声明了如果元素被忽略，则元素的内容将如同它由被忽略的元素的容器包含那样被处理。

<ProcessContent>属性

属性	描述
Elements	对其要处理内容的元素名的用空格定界的列表，或者“*”指示应当被处理的所有元素的内容的。如果未指定，则元素属性默认为“*”。

<CarryAlong>元素

可任选的<CarryAlong>元素向文档编辑工具指示当修改文档时是否应当保存可忽略的内容。编辑工具保存或丢弃可忽略内容的方法是在编辑工具的域中。如果多个<CarryAlong>元素指名字空间中的同一元素或属性，则指定的最后一个<CarryAlong>具有优先级。

<CarryAlong>属性

属性	描述
Elements	当编辑文档时请求连同一起传送的元素名的用空格定界的列表，或“*”指示名字空间中的所有元素的内容应当被一起传送。如果未指定，则元素属性默认为“*”。
Attributes	元素内要一起传送的属性名的用空格定界的列表，或“*”指示

	元素的所有属性应当被一起传送。当元素被忽略并一起传送时，所有的属性被一起传送，而不管该属性的内容如何。如果指定的属性在未被忽略的元素中使用，如下文的示例中那样，则只有该属性有效果。属性默认为“*”。

<MustUnderstand>元素

<MustUnderstand>是倒转 Ingorable 元素的效果的元素。例如，当与替换内容结合时，这一技术是有用的。在<MustUnderstand>元素定义的范围外，元素保持 Ignorable。

<MustUnderstand>属性

属性	描述
NamespaceUri	其项必须被理解的名字空间的 URI。

<AlternateContent>元素

<AlternateContent>元素允许如果指定内容的任一部分不被理解则提供替换内容。AlternateContent 块使用了<Prefer>和<Fallback>块。如果<Prefer>块中的任何内容不被理解，则使用<Fallback>块中的内容。名字空间被声明为<MustUnderstand>，以指示要使用后退。如果名字空间被声明为可忽略并且在<Prefer>块中使用了该名字空间，则不使用<Fallback>块中的内容。

版本化标记示例

使用<Ignorable>

本示例使用了假想的标记名字空间 http://PLACEHOLDER/Circle，它定义了其初始版本中的元素 Circle，并使用了引入到标记的未来版本（版本 2）中的 Circle 的 Opacity 属性，以及引入到标记的更后版本（版本 3）中的 Luminance 属性。这一标记在版本 1 和 2 中，以及 3 和随后的版本中保持可加载。另外，<CarryAlong>元素指定了当编辑时，甚至在编辑器不理解 v3:Luminance 时，也必须保存 v3:Luminance。

对于版本 1 阅读者，忽略 Opacity 和 Luminance

对于版本 2 阅读者，仅忽略 Luminance

对于版本 3 及以后的版本的阅读者，使用所有的属性。

```
<FixedPanel
    xmlns="http://PLACEHOLDER/fixed-content"
    xmlns:v="http://PLACEHODER/versioned-content"
    xmlns:v1="http://PLACEHODER/Circle/v1"
    xmlns:v2="http://PLACEHODER/Circle/v2"
    xmlns:v3="http://PLACEHODER/Circle/v3" >
  <v:Compatibility.Rules>
    <v:Ignorable NamespaceUri=" http://PLACEHODER/Circle/v2" />
    <v:Ignorable NamespaceUri=" http://PLACEHODER/Circle/v3" >
      <v:CarryAlong Attributes="Luminance" />
    </v:Ignorable>
  </v:Compatibility.Rules>
  <Canvas>
    <Circle Center="0,0" Radius="20" Color="Blue"
      v2:Opacity="0.5" v3:Luminance="13" />
    <Circle Center="25,0" Radius="20" Color="Black"
      v2:Opacity="0.5" v3:Luminance="13" />
    <Circle Center="50,0" Radius="20" Color="Red"
      v2:Opacity="0.5" v3:Luminance="13" />
    <Circle Center="13,20" Radius="20" Color="Yellow"
      v2:Opacity="0.5" v3:Luminance="13" />
    <Circle Center="38,20" Radius="20" Color="Green"
      v2:Opacity="0.5" v3:Luminance="13" />
  </Canvas>
</FixedPanel>
```

使用<MustUnderstand>

以下示例阐明了对<MustUnderstand>元素的使用。

```
<FixedPanel
   xmlns="http://PLACEHOLDER/fixed-content"
   xmlns:v="http://PLACEHODER/versioned-content"
   xmlns:v1="http://PLACEHODER/Circle/v1"
   xmlns:v2="http://PLACEHODER/Circle/v2"
   xmlns:v3="http://PLACEHODER/Circle/v3" >
  <v:Compatibility.Rules>
    <v:Ignorable NamespaceUri="http://PLACEHODER/Circle/v2" />
    <v:Ignorable NamespaceUri="http://PLACEHODER/Circle/v3" >
      <v:CarryAlong Attributes="Luminance" />
    </v:Ignorable>
  </v:Compatibility.Rules>
  <Canvas>
    <v:Compatibility.Rules>
      <v:MustUnderstand NamespaceUri="http://PLACEHODER/Circle/v3" />
    </v:Compatbility.Rules>
    <Circle Center="0,0" Radius="20" Color="Blue"
      v2:Opacity="0.5" v3:Luminance="13" />
    <Circle Center="25,0" Radius="20" Color="Black"
      v2:Opacity="0.5" v3:Luminance="13" />
    <Circle Center="50,0" Radius="20" Color="Red"
      v2:Opacity="0.5" v3:Luminance="13" />
    <Circle Center="13,20" Radius="20" Color="Yellow"
      v2:Opacity="0.5" v3:Luminance="13" />
    <Circle Center="38,20" Radius="20" Color="Green"
      v2:Opacity="0.5" v3:Luminance="13" />
  </Canvas>
</FixedPanel>
```

对<MustUnderstand>元素的使用导致对 v3:Luminance 的引用错误，即使它在根元素中被声明为 Ignorable。如果与使用例如添加到版本 2 中的 Canvas 的 Luminance 属性（见下文）的替换内容结合，则这一技术是有用的。在 Canvas 元素的范围外，Circle 的 Luminance 属性再次是可忽略的。

```
<FixedPanel
   xmlns="http://PLACEHOLDER/fixed-content"
   xmlns:v="http://PLACEHODER/versioned-content"
   xmlns:v1="http://PLACEHODER/Circle/v1"
   xmlns:v2="http://PLACEHODER/Circle/v2"
   xmlns:v3="http://PLACEHODER/Circle/v3" >
  <v:Compatibility.Rules>
   <v:Ignorable NamespaceUri="http://PLACEHODER/Circle/v2" />
   <v:Ignorable NamespaceUri="http://PLACEHODER/Circle/v3" >
    <v:CarryAlong Attributes="Luminance" />
   </v:Ignorable>
  </v:Compatibility.Rules>
  <Canvas>
   <v:Compatibility.Rules>
    <v:MustUnderstand NamespaceUri="http://PLACEHODER/Circle/v3" />
   </v:Compatbility.Rules>
   <v:AlternateContent>
    <v:Prefer>
     <Circle Center="0,0" Radius="20" Color="Blue"
      v2:Opacity="0.5" v3:Luminance="13" />
     <Circle Center="25,0" Radius="20" Color="Black"
      v2:Opacity="0.5" v3:Luminance="13" />
     <Circle Center="50,0" Radius="20" Color="Red"
      v2:Opacity="0.5" v3:Luminance="13" />
     <Circle Center="13,20" Radius="20" Color="Yellow"
      v2:Opacity="0.5" v3:Luminance="13" />
     <Circle Center="38,20" Radius="20" Color="Green"
      v2:Opacity="0.5" v3:Luminance="13" />
    </v:Prefer>
    <v:Fallback>
     <Canvas Luminance="13">
      <Circle Center="0,0" Radius="20" Color="Blue"
       v2:Opacity="0.5" />
      <Circle Center="25,0" Radius="20" Color="Black"
       v2:Opacity="0.5" />
      <Circle Center="50,0" Radius="20" Color="Red"
       v2:Opacity="0.5" />
      <Circle Center="13,20" Radius="20" Color="Yellow"
       v2:Opacity="0.5" />
      <Circle Center="38,20" Radius="20" Color="Green"
       v2:Opacity="0.5" />
     </Canvas>
    </v:Fallback>
   </v:AlternateContent>
  </Canvas>
</FixedPanel>
```

使用<AlternateContent>

如果任一元素或属性被声明为<MustUnderstand>，但是在<AlternateContent>块的<Prefer>块中不被理解，则整体跳过<Prefer>块，并如常地处理<Fallback>块

（即，遇到的任何 MustUnderstand 项被报告为错误）。

```
<v:AlternateContent>
  <v:Prefer>
    <Path xmlns:m="http://schemas.example.com/2008/metallic-finishes"
      m:Finish="GoldLeaf" ..... />
  </v:Prefer>
  <v:Fallback>
    <Path Fill="Gold" ..... />
  </v:Fallback>
</v:AlternateContent>
```

到达包格式

在以下的讨论中，提供了对特定文件格式的描述。本节中的各个初级小标题包括“对到达包格式的介绍”、“到达包结构”、“固定有效负载部件”、“固定页标记基础”、“固定有效负载元素和属性”以及“固定页标记”。每一初级小标题具有一个或多个相关的小标题。

对到达包格式的介绍

在上文描述了示例性框架之后，以下的描述是使用上述工具提供的特定格式之一。要意识到和理解，以下描述仅构成了一个示例性格式，并不想要限制要求保护的本发明的应用。

依照本实施例，单个包可包含多个有效负载，其每一个担当文档的不同表示。有效负载是部件的集合，包括可标识的“根”部件和该根部件的有效处理所需的所有部件。例如，有效负载可以是文档的固定表示、可重新流动的表示或任何任意的表示。

以下描述定义了被称为“固定有效负载”的特定表示。固定有效负载具有包含固定面板标记的根部件，它进而引用固定页部件。这些共同描述了多页文档的精确呈现。

容纳至少一个固定有效负载，并遵循以下描述的其它规则的包被称为到达包。到达包的阅读者和书写者可基于到达包格式的规范实现其自己的语法分析器和呈现引擎。

到达包的特征

依照所描述的实施例，到达包解决了信息工作者对分发、归档和呈现文档的需求。使用已知的呈现规则，到达包可以从保存它们的格式中明白且明确地再生或

打印，而无需将客户机设备或应用程序绑定到特定的操作系统或服务库。另外，由于到达有效负载是以中立的、应用程序无关的方式来表达的，因此文档通常可以在没有用于创建该包的应用程序的情况下来察看和打印。为提供这一能力，在到达包中引入并包含了固定有效负载的概念。

依照所描述的实施例，固定有效负载具有固定数量的页，并且分页符总是相同的。固定有效负载中页面上所有元素的布局是预定的。每一页具有固定的大小和方向。由此，不必要在消费端上执行布局计算，并且可容易地呈现内容。这不仅应用到图形，也应用到文本，它在固定有效负载中用精确的排字布局来表示。页面的内容（文本、图形、图像）使用更强大但是更简单的可视原语集来描述。

到达包支持用于组织页面的各种机制。一组页面逐个被“粘合”在一起成“固定面板”。该组页面粗略地等效于传统的多页文档。固定面板然后可进一步参与组成一构建序列和选择以汇编“复合”文档的过程。

在所示并描述的实施例中，到达包支持一种特定种类的序列，称为固定面板序列，它可用于例如将一组固定面板粘合在一起成单个、较大的“文档”。例如，想像将来自不同来源的两个文档粘合在一起：一个两页的封面备忘录（固定面板）以及二十页的报表（固定面板）。

到达包支持多个特定选择器，它们可在构建包含“相同”内容的替换表示的文档包时使用。具体地，到达包允许基于语言、颜色能力和页面大小的选择。由此，例如，可具有双语言文档，它使用选择器在文档的英语表示和法语表示之间选取。

除选择器和序列对于到达包的组成的这些简单使用之外，重要的是注意，选择器和序列还可指进一步的选择器和序列，由此允许构建强大的聚积分层结构。依照本实施例，关于什么可以完成以及什么不能完成的确切规则在下文名为“到达包结构”一节中指定。

另外，到达包可包含附加的有效负载，它们不是固定有效负载，而是文档的更丰富并且可能可编辑的表示。这允许包包含在编辑器应用程序中运作良好的丰富的、可编辑的文档，以及视觉上准确并可以在不编辑应用程序的情况下察看的表示。

最后，依照该实施例，到达包支持所谓的打印票据。打印票据提供了应当在打印包时使用的设置。这些打印票据可以用各种方式附加，以实现实质的灵活性。例如，打印票据可以被附加到整个包，并且其设置将影响整个包。打印票据还可以在结构中的较低级别上附加（例如，附加到个别的页），并且这些打印票据将提供在打印它们所附加的部件时使用的覆盖设置。

到达包结构

如上所述，到达包支持一组特征，包括“固定”页面、固定面板、组成部分、打印票据等等。这些特征在包内使用包模型的核心组件来表示：部件和关系。在本节及其相关的小节中，提供了对“到达包”的完整描述，包括所有这些部件和关系必须如何被汇编、相关等的描述。

到达包结构综述

图 10 示出了一个示例性到达包，并且在本实施例中，示出了可构成包或在包中找到的部件的每一有效类型。下文提供的表格列出了每一有效部件类型，并提供了每一个的描述：

固定页 application/xml+FixedPage-PLACEHOLDER	每一固定页部件表示页的内容
固定面板 application/xml+FixedPanel-PLACEHOLDER	每一固定面板按顺序将一组固定页粘合在一起
字体	字体可以嵌入在包中，以确保文档字形的可靠再现
图像 image/jpeg image/png	图像部件可包括在内
排版部件 application/xml+Selector+[XXX] application/xml+Sequence+[XXX]	选择器和序列可用于构建“组成”块，将较高级组织引入到包中。
描述性元数据 application/xml+SimpleTypeProperties-PLACEHOLDER	描述性元数据（如，标题，关键字）可对文档包括在内。
打印票据 application/xml+PRINTTICKET-PLACEHOLDER	打印票据可被包括在内以提供当打印包时使用设置

由于到达包被设计成“在任何地方察看和打印”的文档，因此到达包的阅读者和书写者必须共享对什么构成了“有效”到达包的公用、明白定义的期望。为提

供“有效”到达包的定义，下文首先定义若干概念。

到达排版部件

到达包必须包含可通过从包的起始部件开始遍历组成块来“发现”的至少一个固定面板。依照所描述的实施例，发现过程遵循以下算法：

- 从包的起始部件开始递归地遍历排版部件图。
- 当执行该遍历时，仅遍历到作为到达排版部件（以下描述）的排版部件中。
- 在图的边上定位所有的终端节点（没有向外的弧的节点）。

这些终端节点指的是（通过其<item>元素）一组被称为到达有效负载根的部件。

固定有效负载

固定有效负载是其根部件是固定面板部件的有效负载。例如，图 10 中的每一固定有效负载具有相关联的固定面板部件作为其根部件。有效负载包括固定面板的有效处理所需的所有部件的完整闭包。这包括：

- 固定面板本身；
- 从固定面板内引用的所有固定页；
- 由有效负载中的任一固定页（直接或通过选择器间接）引用的所有图像部件；
- 从有效负载内的任一固定页内使用的图像刷直接或间接引用的所有到达选择器（下文描述）；
- 由有效负载中的任一固定页引用的所有字体部件；
- 附加到固定有效负载的任一部件的所有描述性元数据部件；以及
- 附加到固定有效负载中的任一部件的任一打印票据。

用于到达包的有效性规则

在适当的位置有了上述定义，现在描述依照所描述的实施例描述“有效”到达包的一致性规则：

- 到达包必须具有使用上述包关系的标准机制定义的起始部件；
- 到达包的起始部件必须是选择器或序列；
- 到达包必须具有作为固定面板的至少一个到达有效负载根；

- 打印票据部件可以被附加到任一排版部件、固定面板部件或固定面板内标识的任一固定页部件。在本实例中，这是通过http://PLACEHOLDER/HasPrintTicketRel关系来完成的；
  - o 打印票据可以被附加到这些部件的任一个或所有；
  - o 任一给定部件必须附加不多于一个的打印票据；
- 描述性元数据部件可以被附加到包内的任一部件；
- 固定有效负载中的每一字体对象必须满足“字体部件”一节中定义的字体格式规则；
- 从固定有效负载的任一固定页对图像的引用必须指向（可能通过其它选择器递归地）作出选择来找出要呈现的实际图像的选择器；
- 固定有效负载中使用的每一图像对象必须满足“图像部件”一节中定义的字体格式规则；
- 对于从固定页（直接或通过选择器间接）引用的任一字体、图像或选择器部件，必须有从引用固定页到被引用的部件的“需要的部件”关系（关系名=http://mmcf-fixed-RequiredResource-PLACEHOLDER）。

到达排版部件

尽管到达包可包含许多类型的排版部件，然而仅明确定义的排版部件的类型集具有依照该文档明确定义的行为。这些具有明确定义的行为的排版部件被称为到达排版部件。不同于这些的部件在确定到达包的有效性时不是相关的。

以下排版部件类型被定义为到达排版部件：

语言选择器 application/xml+selector+language	基于其自然语言在表示之间选择
颜色选择器 application/xml+selector+color	基于它们是单色还是彩色在表示之间选择
页面大小选择器 application/xml+selector+contenttype	基于其页面大小在表示之间选择
内容类型选择器 application/xml+selector+contenttype	基于其内容类型是否可被系统理解在表示之间选择
固定序列	将固定内容的子元素组合成序列

application/xml+sequence+fixed	

到达选择器

被定义为到达排版部件的那些选择器排版部件被称为到达选择器。如上所述，语言选择器基于其自然语言，如英语或法语在表示之间选择。为发现该语言，选择器调查其每一项。仅作为 XML 的那些项被考虑在内。对于那些项，调查每一项的根元素以确定其语言。如果 xml:lang 属性不存在，则忽略该部件。选择器然后依次考虑这些部件的每一个，选择其语言与系统的默认语言相匹配的第一个部件。

颜色选择器基于它们是单色还是彩色在表示之间选择。页面大小选择器基于其页面大小在表示之间选择。内容类型选择器基于其内容类型是否能被系统理解在表示之间选择。

到达序列

被定义为到达排版部件的那些序列排版部件被称为到达序列。固定的序列将固定内容的子部件组合成序列。

固定有效负载部件

固定有效负载能包括以下种类的部件：固定面板部件、固定页面部件、图像部件、字体部件、打印票据部件以及描述性元数据部件，其每一个在下文其自己的小标题下讨论。

固定面板部件

固定有效负载的文档结构将固定页面标识为骨架的一部分，如下文所描述的。骨架部件和页面部件之间的关系在骨架的关系流内定义。固定面板部件是内容类型 application/xml+PLACEHOLDER。

固定有效负载内容的骨架通过在<Document>元素内包括<FixedPanel>元素在标记中指定。在下文的示例中，<FixedPanel>元素指定了骨架中容纳的页的来源。

```
<!-- SPINE -->
<Document $XMLNSFIXED$ >
    <FixedPanel>
        <PageContent Source="p1.xml" />
        <PageContent Source="p2.xml" />
    </FixedPanel>
</Document>
```

<Document>元素

<Document>元素没有属性，并且必须只有一个子元素：<FixedPanel>。

<FixedPanel>元素

<FixedPanel>元素是文档的骨架，它逻辑上将已排序的页面序列绑定在一起成单个多页文档。页面总是指定了其自己的宽度和高度，但是<FixedPanel>元素也可任选地指定高度和宽度。该信息可用于各种各样的目的，包括，例如基于页面大小在替换表示之间选择。如果<FixedPanel>元素指定了高度和宽度，则它通常可与<FixedPane>内的页面的宽度和高度对齐，但是这些尺寸不指定个别页的高度和宽度。

以下表格依照所描述的实施例总结了 FixedPanel 属性。

<FixedPanel>属性	描述
PageHeight	包含在<FixedPanel>内的页面的典型高度。可任选。
PageWidth	包含在<FixedPanel>内的页面的典型宽度。可任选。

<PageContent>元素是<FixedPanel>元素唯一允许的子元素。<PageContent>元素是与文档的页顺序相匹配的顺序标记次序。

<PageContent>元素

每一<PageContent>元素指的是单个页面的内容的来源。为确定文档中页的数量，可对包含在<FixedPanel>中的<PageContent>子元素进行计数。

<PageContent>元素没有允许的子元素，并且有单个需要的属性－Source，它指的是页的内容的固定页面部件。

如<FixedPanel>元素一样，<PageContent>元素可任选地包括 PageHeight 和 PageWidth 属性，此处反映了单个页的大小。需要的页面大小在固定页面部件内指定；<PageContent>上的可任选大小仅是建议性的。<PageContent>大小属性允许诸如文档察看器等应用程序对于文档快速地做出可视布局估算，而不加载和语法分析所有的个别固定页面部件。

下文提供的表格总结了<PageContent>属性并提供了属性的描述。

<PageContent>属性	描述
Source	参考页内容的 URI 串，容纳在包内的相异部件内。内容被标识为包内的部件。需要。
PageHeight	可任选
PageWidth	可任选

页内容的 URI 串必须引用内容相对于包的部件位置。

固定页面部件

<FixedPanel>中的每一<PageContent>元素按名字（URI）引用了固定页面部件。每一固定页面部件包含描述内容的单个页面的呈现的固定页面标记。固定页面部件是内容类型 application/xml+PLACEHOLDER-FixedPage。

在标记中描述固定页面

以下是源内容的标记如何查找上述实例骨架标记（<PageContentSource="p1.xml" />）中引用的页面的示例。

```
//   /content/p1.xml
<FixedPage PageHeight="1056" PageWidth="816">
       <Glyphs
              OriginX = "96"
              OriginY = "96"
              UnicodeString = "This is Page 1!"
              FontUri = "../Fonts/Times.TTF"
              FontRenderingEmSize = "16"
       />
</FixedPage>
```

以下表格总结了 FixedPage 的属性并提供了属性的描述。

FixedPage 的属性	描述
PageHeight	需要
PageWidth	需要

固定页面标记中的读取顺序

在一个实施例中，包含在固定页面内的 Glyphs 子元素的标记顺序必须与页面

的文本内容的期望读取顺序相同。该读取顺序可用于来自察看器中的固定页面的顺序文本的交互式选择/赋值，以及允许可访问性技术对顺序文本的访问。确保标记顺序和读取顺序的之间的这一对应性是生成固定页面标记的应用程序的责任。

图像部件

支持的格式

依照所描述的实施例，到达包中的固定页面使用的图像部件可以是固定数量的格式，例如 PNG 或 JPEG，尽管可使用其它格式。

字体部件

依照所描述的实施例，到达包支持有限数量的字体格式。在所示并描述的实施例中，支持的字体格式包括 TrueType 格式和 OpenType 格式。

如本领域的技术人员所理解的，OpenType 字体格式是 TrueType 字体格式的扩展，添加了对 PostScript 字体数据和复杂的排字布局的支持。OpenType 字体文件包含表格格式的数据，它包括 TrueType 轮廓字体或 PostScript 轮廓字体。

依照所描述的实施例，到达包内不支持以下字体格式：Adobe 类型 1、位图字体、具有隐藏属性的字体（使用系统标志来判断是否枚举它）、矢量字体以及 EUDC 字体（其字体家族名是 EUDC）。

分设置字体

固定有效负载标识使用下文详细描述的 Glyphs 元素的所有文本。由于在本实施例中格式是固定的，因此可能对字体进行分设置以仅包含固定有效负载所需的字形。因此，到达包内的字体可以基于字形使用率来分设置。尽管分设置的字体将不包含原始字体中的所有字形，然而分设置的字体必须是有效的 OpenType 字体文件。

打印票据部件

打印票据部件提供了可在打印包时使用的设置。这些打印票据可以用各种方式附加，以实现实质上的灵活性。例如，打印票据可被“附加”到整个包，并且其设置将影响整个包。打印票据可以在结构中的较低级进一步附加（例如，附加到个别的页），并且这些打印票据将提供在打印它们所附加的部件时使用的覆盖设置。

描述性元数据

如上所述，描述性元数据部件向包的书写者或生产者提供了储存属性值的方法，使包的阅读者能够可靠地发现这些值。这些属性通常用于记录关于整个包以及容器内个别部件的附加信息。

固定页面标记基础

本节描述了与固定页面标记相关联的某些基础信息，并包括以下小节：“固定有效负载和其它标记标准”、“固定页面标记模型”、“资源和资源引用”、以及“固定页面绘制模型”。

固定有效负载和其它标记标准

到达包内的固定有效负载的固定面板和固定页面标记是来自 Windows® Longhorn 的 Avalon XAML 标记的子集。即，尽管固定有效负载标记作为独立的 XML 标记格式（如在本文档中编制的）是独立的，但是它以与 Longhorn 系统相同的方式加载，并呈现了原始多页文档的 WYSIWYG 再现。

作为 XAML 标记的某些背景，考虑以下。XAML 标记是允许用户将对象的层次以及对象后的编程逻辑指定为基于 XML 的标记语言的机制。这为对象模型提供了以 XML 描述的能力。这允许诸如微软公司的.NET 框架的公用语言运行库（CLR）等类中的可扩充类以 XML 来访问。XAML 机制提供了 XML 标签到 CLR 对象的直接映射，以及以标记表示相关代码的能力。可以明白并理解，各个实现不需要特别地使用 XAML 的基于 CLR 的实现。相反，基于 CLR 的实现仅构成了可在本文档中描述的实施例环境中采用 XAML 的一种方法。

更具体地，考虑结合图 11 的以下内容，它示出了 CLR 概念（左侧分量）到 XML（右侧分量）的示例性映射。名字空间使用称为反射的 CLR 概念在 xmlns 声明中找到。类直接映射到 XML 标签。属性和事件直接映射到属性。使用这一分层结构，用户可指定 XML 标记文件中的任一 CLR 对象的分层树。xaml 文件是具有 xaml 扩展名以及 application/xaml+xml 的媒体类型的 xml 文件。xaml 文件具有一个根标签，它通常使用 xmlns 属性指定了名字空间。名字空间可以在标签的其它类型中指定。

继续，xaml 文件中的标签一般映射到 CLR 对象。标签可以是元素、复合属性、

定义或资源。元素是一般在运行时例示的 CLR 对象，并形成了对象的分层结构。复合属性标签用于设置父标签中的属性。定义标签用于将代码添加到页并定义资源。资源标签提供了仅通过将树指定为资源来重新使用对象树的能力。定义标签也可在另一标签中被指定为 xmlns 属性。

一旦以标记（通常由书写者）合适地描述了文档，可以（通常由阅读者）对标记进行语法分析和处理。合适配置的语法分析器从根标签中确定应当搜索哪些 CLR 汇编和名字空间来找出标签。在许多情况下，语法分析器进行查找，并找出由 xmlns 属性指定的 URL 中的名字空间定义文件。名字空间定义文件提供了汇编的名字及其安装路径和 CLR 名字空间的列表。当语法分析器遇到标签时，语法分析器使用标签的 xmlns 以及该 xmlns 的 xmlns 定义文件来确定标签引用哪些 CLR 类。语法分析器按定义文件中指定的汇编和名字空间的顺序进行搜索。当它找到匹配时，语法分析器例示该类的对象。

由此，上文描述，并且在上文通过引用更完全结合在本申请中的机制允许使用标记标签以基于 XML 文件来表示对象模型。这一将对象模型表示为标记标签的能力可以用于异步或同步地创建矢量图形绘图、固定格式的文档、自适应流文档和应用程序 UI。

在所示并描述的实施例中，固定有效负载标记是最小的，几乎完全是呈现原语的 Avalon XAML 的节俭的子集。它用完全的保真度可视地表示了可以用 Avalon 表示的任何内容。固定有效负载标记是 Avalon XAML 元素和属性的子集－加上附加的约定、正则形式或与 Avalon XAML 相比在使用率上的限制。

定义的完全最小固定有效负载标记集减少了与到达包阅读者，如打印机 RIP 或交互式察看器应用程序的实现和测试相关联的成本－并减少了相关联的语法分析器的复杂度和存储器覆盖区。节俭的标记集也最小化了分设置、错误或到达包书写者和阅读者之间的不一致性的机会，从而令格式及其生态系统内在地更健壮。

除最小的固定有效负载标记之外，到达包将为附加的语义信息指定标记，以支持具有诸如超链接、版面/大纲结构和导航、文本选择和文档可访问性等特征的察看器到达包或呈现。

最后，使用上述版本化和可扩充性机制，用用于特定的目标消耗应用程序、察看器或设备的更丰富的元素集来补充最小的固定有效负载标记是可能的。

固定页面标记模型

在所示并描述的实施例中，固定页面部件是基于 XML 元素、XML 元属性和 XML 名字空间以基于 XML 的标记语言来表达的。本文档中定义了三个 XML 名字空间以包括在固定页面标记中。一个这样的名字空间参考了本说明书中其它地方定义的版本控制元素和属性。用于固定页面标记中的元素和属性的原则名字空间是“http://schemas.microsoft.com/MMCF-PLACEHOLDER-FixedPage”。最后，固定页面标记引入了“资源”的概念，它需要下文描述的第三个名字空间。

尽管固定页面标记是使用 XML 元素和 XML 元属性来表达的，然而其规范基于“内容”和“属性”的更高级抽象模型。固定页面元素都被表达为 XML 元素。仅少数固定页面元素可容纳“内容”，表达为 XML 子元素。但是属性值可使用 XML 元属性或使用 XML 子元素来表达。

固定页面标记也依赖于资源字典和资源引用的双概念。资源字典和多个资源引用的组合允许单个属性值由多个固定页面标记元素的多个属性共享。

固定页面标记中的属性

在所示并描述的实施例中，有三种形式的标记，它们可用于指定固定页面标记属性的值。

如果属性是使用资源引用来指定的，则属性名用作 XML 元属性名，并且属性值的特殊句法指示了资源引用的存在。用于表达资源引用的句法在名为“资源和资源引用”一节中描述。

未被指定为资源引用的任何属性值可以使用标识其值被设置的属性的嵌套 XML 子元素以 XML 来表达。该“复合属性句法”在下文描述。

最后，某些非资源引用属性值可被表达为简单文本串。尽管所有这样的属性值可以使用复合属性句法来表达，然而它们也可使用简单的 XML 元属性句法来表达。

对于任一给定的元素，任一属性可以被设置不多于一次，无论用于指定该值的句法是什么。

简单属性句法

对于可被表达为简单串的属性值，可使用 XML 元属性句法来指定属性值。例如，给定称为“SolidColorBrush（纯色刷）”的固定页面标记元素，它具有名为“Color（颜色）”的属性，可使用以下句法来指定属性值：

```
<!-- Simple Attribute Syntax -->
<SolidColorBrush Color="#FF0000" />
```

复合属性句法

某些属性值不能被表达为简单的串，例如 XML 元素用于描述属性值。这一属性值不能使用简单属性句法来表达。但是它们可使用复合属性句法来表达。

在复合属性句法中，使用了 XML 子元素，但是从父元素名和属性名的组合导出 XML 元素名，以点分隔。给定固定页面标记元素<Path>（路径），它具有属性“Fill”，它可被设置到<SolidColorBrush>，可使用以下标记来设置<Path>元素的“Fill”属性：

```
<!-- Compound-Property Syntax -->
<Path>
    <Path.Fill>
        <SolidColorBrush Color="#FF0000" />
    </Path.Fill>
    …
</Path>
```

复合属性句法即使在简单属性句法足以表达属性值的情况下也可使用。因此，前一节的示例：

```
<!-- Simple Attribute Syntax -->
<SolidColorBrush Color="#FF0000" />
```

可替代地以复合属性句法来表达：

```
<!-- Compound-Property Syntax -->
<SolidColorBrush>
      <SolidColorBrush.Color>#FF0000</SolidColorBrush.Color>
</SolidColorBrush>
```

当使用复合属性句法来指定属性值时，表示“属性”的 XML 子元素必须出现在表示“内容”的 XML 子元素之前。个别复合属性 XML 子元素的顺序不是重要的，只需它们一起出现在父元素的任何“内容”之前。

例如，当使用<Canvas>（画布）元素（下文描述）的 Clip（裁剪）和 RenderTransform（渲染变换）属性时，这两个属性都必须出现在<Canvas>的任何<Path>和<Glyphs>内容之前：

```
<Canvas>
    <!-- First, the property-related child elements -->
    <Canvas.RenderTransform>
        <MatrixTransform Matrix="1,0,0,1,0,0">
    </Canvas.RenderTransform>
    <Canvas.Clip>
        <PathGeometry>
            …
        </PathGeometry>
    </Canvas.Clip>

    <!-- Then, the "Contents" -->
    <Path …>
        …
    </Path>
    <Glyphs …>
        …
    </Glyphs>
</Canvas>
```

资源和资源引用

资源字典可用于容纳可共享的属性值，其每一个被称为资源。其本身是固定页面标记元素的任一属性值可以容纳在资源字典中。资源字典中的每一资源携带一名字。资源名可用于参考来自属性的 XML 元属性的资源。

在所示并描述的实施例中，<Canvas>和<FixedPage>元素可携带资源字典。资源字典是以标记表达为名为“Resource（资源）”的属性内的<Canvas>和<FixedPage>元素的属性。然而，个别的资源值直接嵌入在<FixedPage.Resources>或<Canvas.Resources>XML 元素内。句法上，<Canvas.Resources>和<FixedPage.Resource>的标记类似于具有“内容”的标记元素。

依照本实施例，<Canvas.Resources>或<FixedPage.Resources>必须在<Canvas>或<FixedPage>的任何复合属性句法属性值之前。类似地，它们必须在<Canvas>或<FixedPage>的任何“内容”之前。

定义固定有效负载资源字典

任何<FixedPage>或<Canvas>可携带资源字典，用<Canvas.Resource>XML 元素来表达。单个资源字典内的每一元素被给予一唯一的名字，通过使用与该元素相关联的 XML 元属性来标识。为将该“Name”元属性与对应于属性的那些元属性进行区分，Name 元属性取自与固定格式元素的名字空间不同的名字空间。该 XML 名字空间的 URI 是“http://schemas.microsoft.com/PLACEHOLDER-for-resources”。在以下示例中，定义了两种几何结构：一个是矩形，另一个是圆形。

```
<Canvas xmlns:def="http://schemas.microsoft.com/PLACEHOLDER-for-resources">
       <Canvas.Resources>
        <PathGeometry def:Name="Rectangle">
            <PathFigure>
                ...
            </PathFigure>
        </PathGeometry>
        <PathGeometry def:Name="Circle">
            <PathFigure>
                ...
            </PathFigure>
        </PathGeometry>
       </Canvas.Resources>
</Canvas>
```

引用资源

为将属性值设为上文定义的资源之一，使用在{}内包含资源名的 XML 属性值。例如，“{Rectangle}”将表示要使用该几何结构。在以下标记样例中，由字典中的几何结构对象定义的矩形区域将用 SolidColorBrush 来填充。

```
<Canvas>
       <Canvas.Resources>
        <PathGeometry def:Name="Rectangle">
                    ...
        </PathGeometry>
       </Canvas.Resources>
       <Path>
        <Path.Data>
            <PathGeometry PathGeometry="{Rectangle}" />
        </Path.Data>
        <Path.Fill>
            <SolidColorBrush Color="#FF0000" />
        </Path.Fill>
    </Path>
</Canvas>
```

依照本实施例，资源引用必须不出现在资源字典的资源定义内。

用于解析资源引用的检查规则

尽管在同一资源字典内可能无法使用单个名字两次，然而同一名字可以用在单个固定页面部件的两个不同的资源字典内。此外，内部<Canvas>的资源字典可重复使用某些外部<Canvas>或<FixedPage>的资源字典内定义的名字。

当使用资源引用来设置元素的属性时，搜索各个资源字典来找出给定名字的资源。如果具有属性的元素是<Canvas>，则搜索该<Canvas>的资源字典（如果有的话）以找出期望名字的资源。如果元素不是<Canvas>，则搜索以最接近的包含

<Canvas>或<FixedPage>开始。如果在最初搜索的资源字典内未定义期望的名字，则咨询下一最接近的包含<Canvas>或<FixedPage>。如果搜索继续到根<FixedPage>元素，并且在与该<FixedPage>相关联的资源字典内未找到期望名字的资源，则出现错误。

以下示例表示了这些规则。

```
<FixedPage xmlns:def="http://schemas.microsoft.com/PLACEHOLDER-for-resources"
       PageHeight="1056" PageWidth="816">
    <FixedPage.Resources>
        <Fill def:Name="FavoriteColorFill">
            <SolidColorBrush Color="#808080" />
        </Fill>
    </FixedPage.Resources>

       <Canvas>
        <Canvas.Resources>
            <Fill def:Name="FavoriteColorFill">
                <SolidColorBrush Color="#000000" />
            </Fill>
        </Canvas.Resources>
        <!-- The following Path will be filed with color #000000 -->
        <Path Fill="{FavoriteColorFill}">
            <Path.Data>
                …
            </Path.Data>
        </Path>

        <Canvas>
            <!-- The following Path will be filed with color #000000 -->
            <Path Fill="{FavoriteColorFill}">
                <Path.Data>
                    …
                </Path.Data>
            </Path>
        </Canvas>
       </Canvas>

    <-- The following path will be filled with color #808080 -->
    <Path Fill="{FavoriteColorFill}">
        <Path.Data>
            …
        </Path.Data>
       </Path>

</FixedPage>
```

固定页面绘制模型

固定页面元素（或嵌套的画布子元素）是在其上呈现其它元素的元素。内容的排列由对固定页面（或画布）指定的属性、对固定页面（或画布）上的元素指定的属性、以及对固定页面名字空间定义的组成规则来控制。

使用画布来定位元素

在固定标记中，所有的元素都相对于坐标系统的当前原点(0, 0)来定位。当前原点可通过向包含元素的固定页面或画布的每一元素应用 RenderTransform 元属性来移动。

以下示例示出了通过 RenderTransform 对元素的定位。

```
<Canvas>
    <Canvas.Resources>
        <PathGeometry def:Name="StarFish">
            <!-- Various PathFigures in here -->
            ...
        </PathGeometry>
        <PathGeometry def:Name="LogoShape">
            <!-- Various PathFigures in here -->
            ...
        </PathGeometry>
    </Canvas.Resources>

    <!-- Draw a green StarFish and a red LogoShape shifted by 100 to the right and 50 down
-->
    <Canvas>
        <Canvas.RenderTransform>
            <MatrixTransform Matrix="1,0,0,1,100,50"/>
        </Canvas.RenderTransform>
        <Path Fill="#00FF00" Data="{StarFish}"/>
        <Path Fill="#FF0000" Data="{LogoShape}"/>
    </Canvas>

    <!-- Draw a green StarFish and a red LogoShape shifted by 200 to the right and 250 down
-->
    <Canvas>
        <Canvas.RenderTransform>
            <MatrixTransform Matrix="1,0,0,1,200,250"/>
        </Canvas.RenderTransform>
        <Path Fill="#00FF00" Data="{StarFish}"/>
        <Path Fill="#FF0000" Data="{LogoShape}"/>
    </Canvas>
</Canvas>
```

坐标系统和单位

依照所示并描述的实施例，最初设置坐标系统，使得该坐标系统中的一个单位等于英寸的 1/96，被表达为浮点值，坐标系统的原点(0, 0)是固定页面元素的左上角。

RenderTransform 元属性可以在任一子元素上指定，以向当前坐标系统应用仿射变换。

页面尺寸

页面尺寸由固定页面元素上的“PageWidth（页宽度）”和“PageHeight（页高度）”参数来指定。

组成规则

固定页面使用了具有 α 通道的绘图程序模型。依照所描述的实施例，组成必须依照这些规则，并且以以下顺序发生：

- 固定页面（或任何嵌套的画布）被认为是无界的表面，向该表面上按它们出现在标记中的次序绘制子元素。该表面的 α 通道被初始化为“0.0”（全部透明）。实际上，理想的无界表面可以被认为是一种位图缓冲区，它足够大以容纳通过呈现所有的子元素产生的所有标记。
- 表面的内容使用由固定页面（或画布）的渲染变换属性指定的仿射变换来变换。
- 所有的子元素被呈现到表面上、由固定页面（或画布）的裁剪属性来裁剪（也使用渲染变换属性来变换）。固定页面另外裁剪到由(0, 0, PageWitch, PageHeight)指定的矩形。如果子元素具有不透明（Opacity）属性或不透明掩模（OpacityMask）属性，则它在呈现到表面上之前被应用到该子元素。
- 最后，固定页面（或画布）的内容被呈现到其包含元素上。在固定页面的情况下，包含元素是物理映像表面。

呈现依照这些规则发生：

- 在表面上产生标记的唯一元素是“字形（Glyphs）”和“路径（Path）”。
- 所有其它的呈现效果可通过将“字形”和“路径”元素定位到“画布”上，并应用其各种有效元属性来实现。

固定有效负载元素和属性

依照所示并描述的实施例，固定有效负载包括在标记中用于表示页面及其内容的一小组 XML 元素。固定面板部件中的标记使用<Document>、<FixedPanel>和<PageContent>元素将文档的页面一起带到一公用的、容易索引的根。每一固定页面部件表示仅具有<Path>和<Glyphs>元素（它们共同完成所有的绘制）的<FixedPage>元素，以及组合它们的<Canvas>元素中的页面内容。

固定有效负载标记的元素分层结构在以下名为“顶级元素”、“用于路径、裁剪的几何结构”、“用于填充路径、字形或不透明掩模的刷子”、“用于固定页面或画布的资源字典”、“用于 α 透明性的不透明掩模”、“裁剪路径”和“变换”

的章节中总结。

顶级元素

- <Document>[对每一固定面板部件只有一个]
  - 元属性：
    - [无]
  - 子元素：
    - <FixedPanel>[只有一个]
- <FixedPanel>
  - 元属性：
    - PageHeight[可任选]
    - PageWidth[可任选]
  - 子元素：
    - <PageContent>[这些子元素的 1-N 个]
- <PageContent>
  - 元属性：
    - Source[需要]
    - PageHeight[可任选]
    - PageWidth[可任选]
  - 子元素：
    - [无]
- <Fixedpage>
  - 通过简单 XML 元属性直接表达的属性：
    - PageHeight[需要（此处或作为子元素）]
    - PageWidth[需要（此处或作为子元素）]
  - 表达为 XML 子元素的资源字典本身：
    - <FixedPage.Resouces>
  - 通过 XML 子元素表达的属性
    - <FixedPage.PageHeight>[需要（此处或作为元属性）]
    - <FixedPage.PageWidth>[需要（此处或作为元属性）]
  - 通过 XML 子元素表达的内容：

        - <Canvas>
        - <Path>
        - <Glyphs>
- <Canvas>
    - 通过简单 XML 元属性直接表达的属性：
        - Opacity
    - 通过资源字典引用表达的属性：
        - Clip
        - RenderTransform
        - OpacityMask
    - 表达为 XML 子元素的资源字典本身：
        - <Canvas.Resources>
    - 通过 XML 子元素表达的属性：
        - <Canvas.Opacity>
        - <Canvas.Clip>
        - <Canvas.RenderTransform>
        - <Canvas.OpacityMask>
    - 通过 XML 子元素表达的内容：
        - <Canvas>
        - <Path>
        - <Glyphs>
- <Path>
    - 通过简单 XML 元属性直接表达的属性：
        - Opacity
    - 通过资源字典引用表达的属性：
        - Clip
        - RenderTransfrom
        - OpacityMask
        - Fill
    - 通过 XML 子元素表达的属性
        - <Path.Opacity>

        - <Path.Clip>
        - <Path.RenderTransform>
        - <Path.OpacityMask>
        - <Path.Fill>
        - <Path.Data>
- <Glyphs>
    - 通过简单 XML 元属性直接表达的属性：
        - Opacity
        - BidiLevel
        - FontFaceIndex
        - FontHintingEmSize
        - FontRenderingEmSize
        - FontUri
        - Indices
        - OriginX
        - OriginY
        - Sideways
        - StyleSimulations
        - UnicodeString
    - 通过资源字典引用表达的属性：
        - Clip
        - RenderTransform
        - OpacityMask
        - Fill
    - 通过 XML 子元素表达的属性
        - <Glyphs.Clip>
        - <Glyphs.RenderTransfrom>
        - <Glyphs.OpacityMask>
        - <Glyphs.Fill>
        - <Glyphs.Opacity>
        - <Glyphs.BidiLevel>

- <Glyphs.FontFaceIndex>
- <Glyphs.FontHintingEmSize>
- <Glyphs.FontRenderingEmSize>
- <Glyphs.FontUri>
- <Glyphs.Indices>
- <Glyphs.OriginX>
- <Glyphs.OriginY>
- <Glyphs.Sideways>
- <Glyphs.StyleSimulations>
- <Glyphs.UnicodeString>

用于路径、裁剪的几何结构

- <Path.Data>
  - 元属性：
    - [无]
  - 表达为单个 XML 子元素的属性值：

    [Path.Data 总共只有这些子元素的一个]
    - <GeometryCollection>
    - <PathGeometry>
- <GeometryCollection>
  - 元属性：
    - CombineMode
  - 子元素：

    [1-N 个子元素]
    - <GeometryCollection>
    - <PathGeometry>
- <PathGeometry>
  - 元属性：
    - FillRule
  - 子元素：

    [1-N 个子元素]

- <PathFigure>

- <PathFigure>
  - 元属性：
    - [无]
  - 子元素：

    [首先出现 StartSegment，最后出现 CloseSegment，中间有 1-N 个 Ploy*]
    - <StartSegment>
    - <PolyLineSegment>
    - <PolyBezierSegment>
    - <CloseSegment>
- <StartSegment>
  - 通过简单 XML 元属性直接表达的属性：
    - Point
  - 通过 XML 子元素表达的属性
    - <StartSement.Point>
- <PolyLineSegment>
  - 通过简单 XML 元属性直接表达的属性：
    - Points
  - 通过 XML 子元素表达的属性
    - <PolyLineSegment.Points>
- <PolyBezierSegment>
  - 通过简单 XML 元属性直接表达的属性：
    - Points
  - 通过 XML 子元素表达的属性
    - <PolyBezierSegment.Points>

用于填充路径、字形或不透明掩模的刷子

- <Path.Fill>
  - 元属性：
    - [无]
  - 表达为简单 XML 子元素的属性值：

[Path.Fill 只有这些子元素的一个]

- <SolidColorBrush>
- <ImageBrush>
- <DrawingBrush>
- <LinearGradientBrush>
- <RadialGradientBrush>

- <Glyphs.Fill>
  - 元属性：
    - [无]
  - 表达为单个 XML 子元素的属性值：

    [Glyphs.Fill 只有这些子元素的一个]
    - <SolidColorBrush>
    - <ImageBrush>
    - <DrawingBrush>
    - <LinearGradientBrush>
    - <RadialGradientBrush>
- <SolidColorBrush>
  - 通过简单 XML 元属性直接表达的属性：
    - Opacity
    - Color
  - 通过 XML 子元素表达的属性
    - <SolidColorBrush.Opacity>
    - <SolidColorBrush.Color>
- <ImageBrush>
  - 通过简单 XML 元属性直接表达的属性：
    - Opacity
    - HorizontalAlignment
    - VerticalAlignment
    - Viewbox
    - ViewPort
    - Stretch

        - TileMode
        - ContentUnits
        - ViewportUnits
        - ImageSource
    - 通过资源字典引用表达的属性：
        - Transform
    - 通过 XML 子元素表达的属性：
        - <ImageBrush.Opacity>
        - <ImageBrush.Transform>
        - <ImageBrush.HorizontalAlignment>
        - <ImageBrush.VerticalAlignment>
        - <ImageBrush.ViewBox>
        - <ImageBrush.ViewPort>
        - <ImageBrush.Stretch>
        - <ImageBrush.TileMode>
        - <ImageBrush.ContentUnits>
        - <ImageBrush.ViewportUnits>
        - <ImageBrush.ImageSource>
- <DrawingBrush>
    - 通过简单 XML 元属性直接表达的属性：
        - Opacity
        - HorizontalAlignment
        - VerticalAlignment
        - ViewBox
        - ViewPort
        - Stretch
        - TileMode
        - ContentUnits
        - VeiwportUnits
    - 通过资源字典引用表达的属性：
        - Transform

        - Drawing
    - 通过 XML 子元素表达的属性：
        - <DrawingBrush.Opacity>
        - <DrawingBrush.Transform>
        - <DrawingBrush.HorizontalAlignment>
        - <DrawingBrush.Verticalalignment>
        - <DrawingBrush.ViewBox>
        - <DrawingBrush.ViewPort>
        - <DrawingBrush.Stretch>
        - <DrawingBrush.TileMode>
        - <DrawingBrush.ContentUnits>
        - <DrawingBrush.ViewportUnits>
        - <DrawingBrush.Drawing>
- <DrawingBrush.Drawing>
    - 通过 XML 子元素表达的内容：
        - <Canvas>
        - <Path>
        - <Glyphs>
- <LinearGradientBrush>
    - 通过简单 XML 元属性直接表达的属性：
        - Opacity
        - MappingMode
        - SpreadMethod
        - StartPoint
        - EndPoint
    - 通过资源字典引用表达的属性：
        - Transform
        - GradientStops
    - 通过 XML 子元素表达的属性
        - <LinearGradientBrush.Opacity>
        - <LinearGradientBrush.Transform>

        - <LinearGradientBrush.MappingMode>
        - <LinearGradient.Brush.SpreadMethod>
        - <LinearGradientBrush.StartPoint>
        - <LinearGradientBrush.EndPoint>
        - <LinearGradientBrush.GradientStops>
- <RadialGradientBrush>
    - 通过简单 XML 元属性直接表达的属性：
        - Opacity
        - Center
        - Focus
        - RadiusX
        - RadiusY
    - 通过资源字典引用表达的属性：
        - Transform
        - GradientStops
    - 通过 XML 子元素表达的属性：
        - <RadialGradientBrush.Opacity>
        - <RadialGradientBrush.Transform>
        - <RadialGradientBrush.Center>
        - <RadialGradientBrush.Focus>
        - <RadialGradientBrush.RadiusX>
        - <RadialGradientBrush.RadiusY>
        - <RadialGradientBrush.GradientStops>
- <GradientStops>
    - 通过 XML 子元素表达的内容：
        - <GradientStop> [这些子元素的 1-N 个]
- <GradientStop>
    - 通过简单 XML 元属性直接表达的属性：
        - Color
        - Offset
    - 通过 XML 子元素表达的属性

- <GradientStop.Color>
- <GradientStop.Offset>

用于固定页面或画布的资源字典

- <FixedPage.Resources>
- <Canvas.Resources>

这些元素在上文讨论资源字典的一节中已作了讨论。

用于α透明性的不透明性掩模

- <Canvas.OpacityMask>
  - 元属性：
    - [无]
  - 表达为单个 XML 子元素的属性值：

    [Canvas.OpacityMask 只有这些子元素的一个]
    - <SolidColorBrush>
    - <ImageBrush>
    - <DrawingBrush>
    - <LinearGradientBrush>
    - <RadialGradientBrush>
- <Path.OpacityMask>
  - 元属性：
    - [无]
  - 表达为单个 XML 子元素的属性值：

    [Path.OpacityMask 只有这些子元素的一个]
    - <SolidColorBrush>
    - <ImageBrush>
    - <DrawingBrush>
    - <LinearGradientBrush>
    - <RadialGradientBrush>
- <Glyphs.OpacityMask>
  - 元属性：

- [无]

- 表达为单个 XML 子元素的属性值：

  [Glyphs.OpacityMask 只有这些子元素的一个]

  - <SolidColorBrush>
  - <ImageBrush>
  - <DrawingBrush>
  - <LinearGradientBrush>
  - <RadialGradientBrush>

裁剪路径

- <Canvas.Clip>
  - 元属性：
    - [无]
  - 表达为单个 XML 子元素的属性值：

    [Canvas.Clip 只有这些子元素的一个]

    - <GeometryCollection>
    - <PathGeometry>
- <Path.Clip>
  - 元属性：
    - [无]
  - 表达为单个 XML 子元素的属性值：

    [Path.Clip 只有这些子元素的一个]

    - <GeometryCollection>
    - <PathGeometry>
- <Glyphs.Clip>
  - 元属性：
    - [无]
  - 表达为单个 XML 子元素的属性值：

    [Glyphs.Clip 只有这些子元素的一个]

    - <GeometryCollection>
    - <PathGeometry>

变换

- <Canvas.RenderTransform>
  - 表达为单个 XML 子元素的属性值：
    - <MatrixTransform>[需要]
- <Path.RenderTransfrom>
  - 表达为单个 XML 子元素的属性值：
    - <MatrixTransform[需要]
- <Glyphs.RenderTransform>
  - 表达为单个 XML 子元素的属性值：
    - <MatrixTransform>[需要]
- <MatrixTransform>
  - 通过简单 XML 元属性直接表达的属性：
    - Matrix
  - 通过 XML 子元素表达的属性：
    - <MatrixTransform.Matrix>
- <ImageBrush.Transform>
  - 通过简单 XML 元属性直接表达的属性：
    - MatrixTransform
  - 通过 XML 子元素表达的属性
    - <ImageBrush.Transform.MatrixTransform>
- <DrawingBrush.Transform>
  - 通过简单 XML 元属性直接表达的属性：
    - MatrixTransform
  - 通过 XML 子元素表达的属性
    - <DrawingBrush.Transform.MatrixTransform>
- <LinearGradientBrush.Transform>
  - 通过简单 XML 元属性直接表达的属性：
    - MatrixTransform
  - 通过 XML 子元素表达的属性：
    - <LinearGradientBrush.Transform.MatrixTransform>

- <RadialGradientBrush.Transform>
  - 通过简单 XML 元属性直接表达的属性：
    - MatrixTransform
  - 通过 XML 子元素表达的属性：
    - <RadialGradientBrush.Transform.MatrixTransform>

固定页面标记

每一固定页面部件表示<FixedPage>元素中作为根的XML标记中的页面内容。该固定页面标记提供了书写者和阅读者之间的文档的 WYSIWYG 保真度，只有一小组元素和属性：<Path>和<Glyphs>元素（共同完成了所有的绘制），以及组合它们的<Canvas>元素。

公用元素属性

在讨论对固定页面标记中的每一元素专用的属性之前，考虑对绘制和组合元素公用的属性：Opacity、Clip、RenderTransform 和 OpacityMask。这些不仅是对顶级元素公用的属性，它们也是从父元素到子元素“聚积”其结果的唯一属性，如上文“组成规则”一节中所描述的。聚积是应用组成规则的结果。以下表格提供了这些通用属性的概括描述，以及对每一属性的更完整讨论。

属性	元素	描述
Opacity	Canvas 、 Path 、 Glyphs 以 及 SolidColorBrush 、 ImageBrush 、 DrawingBrush 、 LinearGradientBrush 、 RadialGradientBrush	定义元素的均匀透明性

子元素	元素	描述
Clip	Canvas、Path、Glyphs	Clip 限制了可向其在画布上应用刷子的区域。
RenderTransfrom	Canvas、Path、Glyphs	RanderTransform 为该元素的子元素建立了新的坐标结构。仅 MatrixTransfrom 支持。

OpacityMask	Canvas、Path、Glyphs	指定了以与 Opacity 属性相同的方式应用的 α 值的矩形掩模，但是允许逐像素的基础上不同的 α 值。

Opacity 属性

Opacity（不透明性）属性用于在渲染时（α 渲染）透明地混合两个元素。Opacity 属性的范围从 0（完全透明）到 1（完全不透明）。该闭区间外的值在标记语法分析过程中被限制到该范围内。因此，有效地，[-∞...0]是透明的，而[1... ∞]是不透明的。

Opacity 属性通过以下计算来应用（假定非自左乘源和目标颜色，两者都被指定为 scRGB）：

O_E：元素的 Opacity 属性或 OpacityMask 中对应位置处的 α 值

A_S：源表面中存在的 α 值

R_S：源表面中存在的红值

G_S：源表面中存在的绿值

B_S：源表面中存在的蓝值

A_D：目标表面中已经存在的 α 值

R_D：目标表面中已经存在的红值

G_D：目标表面中已经存在的绿值

B_D：目标表面中已经存在的蓝值

A^*：对目标表面所得的 α 值

R^*：对目标表面所得的红值

G^*：对目标表面所得的绿值

B^*：对目标表面所得的蓝值

用 T 下标指定的所有值是临时值（例如，R_{T1}）。

步骤 1：将源 α 值与不透明性值相乘

$A_S=A_S*O_E$

步骤 2：自左乘源 α

$A_{T1}=A_S$

$R_{T1}=R_S*A_S$

$G_{T1}=G_S*A_S$

$B_{T1}=B_S*A_S$

步骤 3a：自左乘目标 α

$A_{T2}=A_D$

$R_{T2}=R_D*A_D$

$G_{T2}=G_D*A_D$

$B_{T2}=B_D*A_D$

步骤 3b：混合

$A_{T2}=(1-A_{T1})*A_{T2}+A_{T1}$

$R_{T2}=(1-A_{T1})*R_{T2}+R_{T1}$

$G_{T2}=(1-A_{T1})*G_{T2}+G_{T1}$

$B_{T2}=(1-A_{T1})*B_{T2}+B_{T1}$

步骤 4：反转自左乘

如果 $A_{T2}=0$，则将所有的 A^*、R^*、G^*、B^*设为 0

否则：

$A^*=A_{T2}$

$R^*=R_{T2}/A_{T2}$

$G^*=G_{T2}/A_{T2}$

$B^*=B_{T2}/A_{T2}$

Clip 属性

Clip 属性被指定为几何结构元素<GeometryCollection>或<PathGeomerty>之一（见 Path.Data 以知道细节）

Clip 属性用以下方式来应用：

- 所有落入由 Clip 子元素描述的几何结构元素内的呈现内容是可见的。
- 所有落入由 Clip 子元素描述的几何结构元素外的呈现内容是不可见的。

RenderTransform 子元素

MatrixTransform 是对元素可用的唯一变换属性。它表达了仿射变换。句法如下：

```
<X.RenderTransform>
    <MatrixTransform Matrix="1,0,0,1,0,0"/>
</X.RenderTransform>
```

X 表示向其应用变换的元素。

Matrix 属性中指定的六个数字是 m00、m01、m10、m11、dx、dy。

完整的矩阵看似为：

m00 m01 0

m10 m11 0

dx dy 1

给定坐标 X、Y 通过应用以下计算用 RenderTransfrom 变换来产生所得的坐标 X'、Y'：

X' = X * m00 + Y * m10 + dx

Y' = X * m01 + Y * m11 + dy

OpacityMask 子元素

OpacityMask 指定了刷子（Brush），但是与填充刷子（Fill Brush）相反，仅使用刷子的 α 通道（见上述 Opacity 属性）作为用于呈现元素的附加参数。元素的每一像素的每一 α 值然后被额外地与 OpacityMack 刷子中的对应位置处的 α 值相乘。

<Canvas>元素

<Canvas>元素用于将元素组合在一起。通常，当固定页面元素共享组合的公用属性（即，Opacity、Clip、RenderTransfrom 或 OpacityMask）时，它们在<Canvas>中被组合在一起。通过在画布上将这些元素组合在一起，通常可将公用属性应用到画布而非个别的元素。

<Canvas>的属性和子元素

<Canvas>元素只有先前描述的公用属性：Opacity、Clip、RenderTransform 和 OpacityMask。它们如下表中描述的用于<Canvas>元素：

属性	在画布上的效果

Opacity	定义画布的均匀透明性

子元素	在画布上的效果
Clip	Clip 描述了可由 Canvas 的子元素向其应用刷子的区域。
RenderTransform	RenderTransform 为画布的子元素建立了新的坐标结构，如另一画布。仅支持 MatrixTransform。
OpacityMask	指定了以与 Opacity 属性相同的方式应用的 α 值的矩形掩模，但是允许逐像素基础上不同的 α 值。

以下标记示例示出了对<Canvas>的使用。

```
<Canvas>
    <Path Fill="#0000FF">
        <Path.Data>
            <PathGeometry>
                <PathFigure>
                    <StartSegment Point="0,0"/>
                    <PolylineSegment Points="100,0 100,100 0,100 0,0"/>
                    <CloseSegment/>
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
</Canvas>
```

关于画布标记中的读取顺序，考虑以下内容。如固定页面一样，包含在 Canvas 内的 Glyphs 子元素的标记顺序必须与文本内容的期望读取顺序相同。该读取顺序必须用于来自察看器中的固定页面的顺序文本的交互式选择/复制，以及允许可访问性技术对顺序文本的访问。确保标记顺序和读取顺序之间的这一对应性是生成固定页面标记的应用程序的责任。

包含在嵌套的 Canvas 元素内的 Glyphs 子元素在出现在 Canvas 之前和之后的兄弟 Glyphs 元素之间内嵌。

示例：

```
<FixedPage>
    <Glyphs . . . UnicodeStrıng="Now ıs the time for " />
    <Canvas>
        <Glyphs . . . UnicodeString="all good men and women " />
        <Glyphs . . . UnicodeStrıng="to come to the aıd " />
    </Canvas>
    <Glyphs . . . UnicodeString="of the party." />
</FixedPage>
```

<Path>元素

Path 元素是描述几何区域的基于 XML 的元素。几何区域是可被填充，或者可用作裁剪路径的形状。诸如矩形和椭圆形等常见的几何结构类型可以使用路径几何结构来表示。路径通过指定需要的 Geometry.Data 子元素和诸如 Fill 或 Opacity 等渲染属性来描述。

<Path>的属性和子元素

以下属性适用于下文描述的<Path>元素：

属性	在 Path 上的效果
Opacity	定义了填充的路径的均匀透明性。

子元素	在 Path 上的效果
Clip	Clip 描述了可由路径的几何结构向其应用刷子的区域。
RenderTransform	RenderTransform 为路径的子元素建立了新坐标结构，如由 Path.Data 定义的几何结构。仅支持 MatrixTransform。
OpacityMask	指定了以与 Opacity 属性相同的方式应用的 α 值的矩形掩模，但是允许对表

	面的不同区域的不同α值。
Data	描述了路径的几何结构。
Fill	描述了用于对路径的几何结构着色的刷子。

为描述如何对由<Path.Data>子元素的几何结构描述的区域着色，使用了 Fill 属性。为限制可在其上绘制<Path.Data>形状的区域，使用了 Clip 属性。

使用<Path>来描述几何结构

路径的几何结构被指定为<Path.Data>的一系列嵌套子元素，如下文所示。几何结构可以用包含一组<PathGeometry>子元素的<GeometryCollection>或包含<PathFigures>的单个<PathGeometry>子元素来表示。

```
<Path>
    <Path.Data>
        <GeometryCollection>
            <PathGeometry>
                <PathFigure>

                </PathFigure>
            </PathGeometry>
        </GeometryCollection>
    </Path.Data>
<Path>
```

相同的<GeometryCollection>或<PathGeometry>元素定义了用于裁剪 Canvas、Path 或 Glyphs 的 Clip 属性中使用的路径的几何结构。

以下表格介绍了定义路径几何结构的子元素的分层结构。

几何结构元素	描述
GeometryCollection	使用 Boolean CombineMode 操作呈现的一组 PathGeometry 元素。
PathGeometry	其每一个使用相同的 FillRule 选项来填充的一组 PathFigure 元素。
PathFigure	一个或多个片段元素的集合。
StartSegment PolyLineSegment PolyBezierSegment	

CloseSegment	

GeometryCollection

GeometryCollection 是一组被组合在一起来依照布尔 CombineMode 选项呈现的几何结构对象。GeometryCollection 元素是固定页面标记中用于构建几何形状的可视组合的机制。

属性	在 GeometryCollection 上的效果
CombineMode	指定了用于组合几何结构的不同模式。

CombineMode 属性指定了用于组合 GeometryCollection 中的一组几何形状的布尔操作。根据模式，可包括或排除不同的区域。

CombineMode 选项	描述
Complement	指定了现有区域由从新区域中删除现有区域的结果来替代。换言之，从新区域中排除现有区域。
Exclude	指定了现有区域用从现有区域中删除新区域的结果来替代。换言之，从现有区域中排除新区域。
Intersect	通过取交集将两个区域组合。
Union	通过取并集将两个区域组合。
Xor	通过仅采用由一个或另一个区域包含，但非两个区域都包含的区域来组合两个区域。

CombineMode 如下处理：

不可交换　Complement 和 Exclude 不是可交换的，因此在 GeometryCollection 中的第一个几何结构和每一个别的剩余几何结构之间定义。例如，对于集合{g1, g2, g3}，Exclude 的 CombineMode 将被应用为((g1 排除 g2)) 并且(g1 排除 g3))。

可交换　布尔操作 Union、Xor、Intersect 是可交换的，因此与顺序无关地应用于几何结构。

PathGeometry

PathGeometry 元素包含一组 PathFigure 元素。PathFigure 的并集定义了 PathGeometry 的内部。

属性	在 GeometryCollection 上的效果
FillRule	指定了用于填充描述闭合区域的路径的替换算法。

关于 FillRule 属性，考虑以下内容。PathGeomerty 的填充区域是通过采用将其 Filled 属性设为真的所有包含的 PathFigure，并应用 FillRule 来确定闭合区域来定义的。FillRule 选项指定了如何将 Geometry 内包含的 Figure 元素的相交区域组合在一起形成 Geometry 的所得区域。

依照所描述的实施例，提供了奇偶填充（EvenOdd Fill）和非零填充（NonZero Fill）算法。

奇偶填充算法通过绘制从画布上的一点到任一方向上的无穷大的射线，然后检查穿过该射线的形状的片段之处，来确定该点的“内部性”。从计数零开始，从左到右每次添加一个穿过该射线的片段，并且从右到左每次减去一个穿过该射线的路径片段。如果这一数字是奇数，则该点在内部；否则，该点在外部。

非零填充算法通过绘制从画布上的一点到任一方向上的无穷大的射线，并对该射线穿过的给定形状的路径片段数进行计数，来确定该点的“内部性”。在对穿过进行计数之后，如果结果是 0，则该点在路径外部。否则，该点在内部。

PathFigure

PathFigure 元素包括一个或多个线段或曲线段的集合。线段元素定义了 PathFigure 的形状。PathFigure 必须总是定义闭合的形状。

属性	在 PathFigure 上的效果
FillRule	指定了用于填充描述闭合区域的路径的替换算法。

图形需要起始点，之后每一线段或曲线段从添加的最后一个点继续。PathFigure 集合中的第一段必须是 StartSegment，而 CloseSegment 必须是最后一段。StartSegment 具有 Point 属性。CloseSegment 没有属性。

StartSegment 属性	描述
Point	线段（起始点）的位置。

用于 Path.Data 几何结构的固定有效负载标记

以下提供了用于绘制并填充画布上的路径的标记。在以下具体示例中，在画布上绘制矩形路径，并用纯绿色刷子来填充。

```
<Canvas>
    <Path Fill="#0000FF">
        <Path.Data>
            <PathGeometry>
                <PathFigure>
                    <StartSegment Point="0,0"/>
                    <PolylineSegment Points="100,0 100,100 0,100 0,0"/>
                    <CloseSegment/>
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
</Canvas>
```

以下标记描述了绘制立方 Bézier 曲线。即，除 PolyLineSegment（折线段）之外，固定有效负载标记包括用于绘制立方 Bézier 曲线的 PolyuBezierSegment。

```
<Canvas>
    <Path Fill="#0000FF">
        <Path.Data>
            <PathGeometry>
                <PathFigure>
                    <StartSegment Point="0,0"/>
                    <PolybezierSegment Points="100,0 100,100 0,100 0,0"/>
                    <CloseSegment/>
                </PathFigure>
            </PathGeometry>
        </Path.Data>
    </Path>
</Canvas>
```

刷子

刷子用于对由<Path>元素定义的几何形状的内部进行着色，并填充用<Glyphs>元素呈现的字符位图。刷子也用于定义<Canvas.OpacityMask>、<Path.OpacityMask>和<Glyphs.OpacityMask>中的 α 透明性掩模。固定页面标记包括以下刷子：

刷子类型	描述
SolidColorBrush	用纯色填充定义的几何区域。
ImageBrush	用图像填充区域。
DrawingBrush	用矢量图填充区域。
LinearGradientBrush	用线性梯度填充区域。
RadialGradientBrush	用径向梯度填充区域。

属性在各刷子之间变化，尽管所有的刷子都具有 Opacity 属性。ImageBrush

和 DrawingBrush 共享平铺显示能力。两个用梯度填充的刷子也具有公用的属性。

对标记中刷子子元素的使用示出如下：

```
<Path>
    <Path.Fill>
        <SolidColorBrush Color="#00FFFF"/>
    </Path.Fill>
    ...
</Path>
```

刷子的公用属性

依照所描述的实施例，以下属性适用于所有的刷子，除简单刷子 SolidColorBrush 之外，它具有更少的可任选子元素。

属性	刷子类型	描述
Opacity	所有刷子	

子元素	刷子类型	描述
Transform	所有刷子，除 SolidColorBrush 之外	描述了应用于刷子的坐标空间的矩阵变换。

用于 DrawingBrush 和 ImageBrush 的公用属性

HorizontalAlignment	DrawingBrush、ImageBrush	Center、Left 或 Right
VerticalAlignment	DrawingBrush、ImageBrush	Center、Bottom 或 Top
ViewBox	DrawingBrush、ImageBrush	
ViewPort	DrawingBrush、ImageBrush	
Stetch	DrawingBrush、ImageBrush	None、Fill、Uniform 或 UniformToFill
TileMode	DrawingBrush、ImageBrush	None、Tile、FilpY、FilpX 或 FilpXY
ContentUnits	DrawingBrush、ImageBrush	Absolute 或 RelativeToBoundingBox
ViewportUnits	DrawingBrush、ImageBrush	Absolute 或 RelativeToBoundingBox

HorizontalAlignment（水平对齐）属性指定了刷子如何在它所填充的区域内水平对齐。VerticalAlignment（垂直对齐）属性指定了刷子如何在它所填充的区域内垂直对齐。ViewBox 属性具有默认值(0, 0, 0, 0)，被解释为未设置。当未设置时，不作出任何调整，并且忽略 Stretch 属性。察看框（ViewBox）为内容指定了新坐标系统，即，重新定义了察看口（ViewPort）的范围和原点。Stretch（拉伸）属性帮助指定那些内容如何映射到察看口。Viewbox 属性的值是四个“无单位”数字<min-x>、<min-y>、<width>和<height>的列表，由空格和/或逗号分隔，并且是 Rect 类型。ViewBox rect 指定了映射到边框的用户空间中的矩形。它与插入 scaleX 和 scaleY 一样起作用。Stretch 属性（在选项不同于无的情况下）提供了用于保持图形的长宽比的附加控制。附加变换被应用到给定元素的所有子孙元素，以达到指定的效果。如果在 Brush 上有变换，则它在 ViewBox 映射“之上”应用。

Stretch 属性具有以下模式：None、Fill、Uniform、UniformToFil。

Stretch 属性选项	描述
None	默认。保持原始大小。
Fill	不保持长宽比，并且缩放内容以填充建立的边界。
Uniform	均匀地缩放大小直到图像适合建立的边界。
UniformToFill	均匀地缩放大小以填充建立的边界，并在需要时剪裁。

简单刷子及其属性

Path.Brush 和 Canvas.Brush 子元素包括以下：SolidColorBrush、ImageBrush 和 DrawingBrush。

SolidColorBrush 用纯色填充定义的几何区域。如果有颜色的 α 分量，则将它以相乘的方式与 Brush 中对应的不透明性属性组合。

属性	效果
Color	为填充的元素指定颜色

以下示例示出了如何对 SolidColorBrush 表达颜色属性。

```
<Path>
    <Path.Fill>
        <SolidColorBrush Color="#00FFFF"/>
    </Path.Fill>
    ...
</Path>
```

ImageBrush 可用于用图像填充空间。ImageBrush 的标记允许指定 URI。如果所有其它属性被设为其默认值，则将拉伸图像以填充区域的边框。

属性	效果
ImageSource	指定了图像资源的 URI

ImageSource 属性必须引用支持的到达图像格式之一或通往这些类型之一的图像的其中一个选择器。

DrawingBrush 可用于用矢量图来填充空间。DrawingBrush 具有 Drawing 子元素，其使用在以下示出的标记中。

```
<Path>
    <Path.Fill>
        <DrawingBrush>
            <DrawingBrush.Drawing>
                <Drawing>
                    <Path … />
                    <Glyphs … />
                </Drawing>
            </DrawingBrush.Drawing>
        </DrawingBrush>
    </Path.Fill>
</Path>
```

梯度刷子及其属性

梯度通过将一组梯度停点（stop）指定为梯度刷子的 XML 子元素来绘制。这些梯度停点沿某一类行进指定了颜色。本框架中支持两种类型的梯度刷子：线性和径向。

梯度是通过在指定的颜色空间内的梯度停点之间完成内插来绘制的。LinearGradientBrush 和 RadialGradientBrush 共享以下公用属性：

属性	描述
SpreadMethod	该属性描述了刷子应当如何填充主要、初始梯度区域外的内容区域。默认值是 Pad（衬填）。
MappingMode	该属性确定是否相对于对象的边框来解释描述梯度的参数。默认值是 relative-to-bounding-box（相对于边框）。

子元素	描述
GradientStops	容纳 GradientStop 元素的已排序序列。

对于 SpreadMethod 属性，考虑以下内容。SpreadMethod 选项指定了如何填充

空间。默认值是 Pad。

SpreadMethod 属性选项	在 Gradient 上的效果
Pad	使用第一个颜色和最后一个颜色来分别填充起始处和结尾处的剩余区域。
Reflect	该梯度停点以相反的顺序重复地重放来填充空间。
Repeat	按顺序重复梯度停点直到填充了空间。

MappingMode 属性

对于 LinearGradientBrush，考虑以下内容。LinearGradientBrush 指定了沿一矢量的线性梯度刷子。

属性	描述
EndPoint	线性梯度的终点。LinearGradientBrush 从 StartPoint（起点）到 EndPoint（终点）内插颜色，其中 StartPoint 表示偏移 0，而 EndPoint 表示偏移 1。默认是 1, 1。
StartPoint	线性梯度的起点。

以下标记示例示出了对 LinearGradientBrush 的使用。用线性梯度填充了具有矩形路径的页面：

```
<FixedPanel>
 <FixedPage>
     <Path>
         <Path.Fill>
             <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
                 <LinearGradientBrush.GradientStops>
                     <GradientStopCollection>
                         <GradientStop Color="#FF0000" Offset="0"/>
                         <GradientStop Color="#0000FF" Offset="1"/>
                     </GradientStopCollection>
                 </LinearGradientBrush.GradientStops>
             </LinearGradientBrush>
         </Path.Fill>
         <Path.Data>
             <PathGeometry>
                 <PathFigure>
                     <StartSegment Point="0,0"/>
                     <PolyLineSegment Points="100,0 100,100 0,100"/>
                     <CloseSegment/>
                 </PathFigure>
             </PathGeometry>
         </Path.Data>
     </Path>
 </FixedPage>
</FixedPanel>
```

该示例示出了用线性梯度填充的具有矩形路径的页面。路径在裁剪它的八边形的形状内也有裁剪属性。

```
<FixedPanel>
    <FixedPage>
        <Path>
            <Path.Clip>
                <PathGeometry>
                    <PathFigure>
                        <StartSegment Point="25,0"/>
                        <PolyLineSegment Points="75,0 100,25 100,75 75,100 25,100 0,75 0,25"/>
                        <CloseSegment/>
                    </PathFigure>
                </PathGeometry>
            </Path.Clip>
            <Path.Fill>
                <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
                    <LinearGradientBrush.GradientStops>
                        <GradientStopCollection>
                            <GradientStop Color="#FF0000" Offset="0"/>
                            <GradientStop Color="#0000FF" Offset="1"/>
                        </GradientStopCollection>
                    </LinearGradientBrush.GradientStops>
                </LinearGradientBrush>
            </Path.Fill>
            <Path.Data>
                <PathGeometry>
                    <PathFigure>
                        <StartSegment Point="0,0"/>
                        <PolyLineSegment Points="100,0 100,100 0,100"/>
                        <CloseSegment/>
                    </PathFigure>
                </PathGeometry>
            </Path.Data>
        </Path>

    </FixedPage>
</FixedPanel>
```

RadialGradient（径向梯度）在编程模型中与线性梯度相似。然而，线性梯度具有定义梯度矢量的起点和终点，而径向梯度具有圆周以及焦点来定义梯度行为。圆周定义了梯度的终点－换言之，1.0 的梯度停点定义了圆周周界的颜色。焦点定义了梯度的中心。0.0 的梯度停点定义了焦点的颜色。

属性	描述
Center	该径向梯度的中心点。RadialGradientBrush 从焦点到椭圆的周界内插颜色。周界由中心和半径来确定。默认是 0.5, 0.5。

Focus	径向梯度的焦点
RadiusX	定义径向梯度的椭圆的 X 维半径。默认是 0.5
RadiusY	定义径向梯度的椭圆的 Y 维半径。默认是 0.5
FillGradient	Pad、Reflect、Repeat

α 和透明性

依照所示并描述的实施例，每一元素的每一像素携带范围从 0.0（完全透明）到 1.0（完全不透明）的 α 值。当混合元素以达到透明性的视觉效果时，使用 α 值。

每一元素可具有 Opacity（不透明性）属性，元素的每一像素的 α 值将与该属性均匀地相乘。

另外，OpacityMask 允许指定每一像素的不透明性，它将控制如何将呈现的内容混合成其目标。由 OpacityMask 指定的不透明性与已经发生在内容的 α 通道中存在的任何不透明性用乘法组合。由 OpacityMask 指定的每一像素的 Opacity 通过查找掩模中每一像素的 α 通道来确定－忽略颜色数据。

OpacityMask 的类型是 Brush。这允许指定如何以各种不同的方式将 Brush 的内容映射到内容的范围。如用于填充几何结构那样，Brush 默认为填充整个内容空间，在适当时拉伸或复制其内容。这意味着图像刷将拉伸其图像源以完全覆盖内容，而梯度刷将从边延伸到边。

α 混合需要的计算在先前“Opcity 属性”一节中描述。

以下示例示出了如何使用 OpacityMask 在 Glyphs 元素上创建“渐弱效果”。本示例中的 OpacityMask 是从不透明的黑渐弱到透明的黑的线性梯度。

```
//   /content/p1.xml
<FixedPage PageHeight="1056" PageWidth="816">
       <Glyphs
              OriginX = "96"
              OriginY = "96"
              UnicodeString = "This is Page 1!"
              FontUri = "../Fonts/Times.TTF"
              FontRenderingEmSize = "16"
       >
        <Glyphs.OpacityMask>
            <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
            <LinearGradientBrush.GradientStops>
                    <GradientStopCollection>
                    <GradientStop Color="#FF000000" Offset="0"/>
                    <GradientStop Color="#00000000" Offset="1"/>
                    </GradientStopCollection>
                </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
        </Glyphs.OpacityMask>
       </Glyphs>
```

```
</FixedPage>
```

到达文档中的图像

在固定页面上，图像填充了闭合区域。为将图像放置在固定页面上，首先必须在页面上指定区域。区域由 Path 元素的几何结构来定义。

Path 元素的 Fill 属性指定了所描述的区域的填充内容。图像是填充的一种类型，由图像耍画到区域中。所有的刷子具有默认的行为，该行为将通过在适当时拉伸或重复（平铺）刷子内容来填充整个区域。在图像刷的情况下，由 ImageSource 属性指定的内容将被拉伸以完全覆盖该区域。

以下标记示出了如何将图像放到画布上。

```
<Canvas>
    <Path>
        <Path.Data>
            <GeometryCollection>
                ...
            </GeometryCollection>
        </Path.Data>
        <Path.Fill>
            <ImageBrush ImageSource="/images/dog.jpg" />
        </Path.Fill>
    </Path>
</Canvas>
```

由于许多图像是矩形的，包括在资源字典中的矩形路径元素在简化标记时是有用的。路径然后可使用 RenderTransform 属性来定位（见上文）。

```
<Canvas>
    <Canvas.Resources>
        <PathGeometry def:Name="Rectangle">
            <PathFigure>
                <StartSegment Point="0,0"/>
                <PolylineSegment Points="100,0 100,100 0,100"/>
                <CloseSegment/>
            </PathFigure>
        </PathGeometry>
    </Canvas.Resources>
       <Canvas>
        <Canvas.RenderTransform>
            <MatrixTransform Matrix="1,0,0,1,100,100"/>
        </Canvas.RenderTransform>
        <Path Data="{Rectangle}">
            <Path.Fill>
                <ImageBrush ImageSource="/images/dog.jpg" />
            </Path.Fill>
        </Path>
       </Canvas>
</Canvas>
```

颜色

颜色可以在所示并描述的标记中使用 scRGB 或 sRGB 表示法来指定。scRGB 规范被称为“IEC 61966-2-2 scRGB”，并且可从 www.iec.ch 获得。

ARGB 参数在下表中描述。

名字	描述
R	当前颜色的红 scRGB 分量
G	当前颜色的绿 scRGB 分量
B	当前颜色的蓝 scRGB 分量
A	当前颜色的 α scRGB 分量

颜色映射

当前，考虑了用指定颜色背景的元数据来对彩色元素加标签。该元数据可包含 ICC 颜色概览或其它颜色定义数据。

<Glyphs>元素

文本在固定有效负载中使用 Glyphs 元素来表示。该元素被设计成满足用于打印和到达文档的需求。

Glyphs 元素可具有以下属性的组合。

属性	目的	标记表示（Glyphs 元素）
Origin	行程中的第一字形的原点。放置字形，使得其前进矢量的前沿及其基线与该点相交。	由 OriginX 和 OriginY 属性指定
FontRenderingEmSize	以绘画表面单位表示的字体大小（默认为 1 英寸的 1/96）	按长度单位来测量
FontHintingEmSize	提示向内的点的大小。字体可包括以不同的大小产生精细差异的提示，如更厚的字干以及尺寸更小的更开放的字碗（bowl），以产生看上去更向纯缩放字罐（can）的相同样式。这不与用于设备像素分辨率的提示相	按表示字体的磅值的双精度型来测量

	同，后者被自动处理。至今为止（2003年3月）没有已知的字体包括大小提示。默认值为12磅。	
GlyphIndices	表示该行程的16比特字形号数组	Indices 属性的一部分。见下文的表示。
AdvanceWidths	前进宽度（advance width）数组，对GlyphIndices 中的每一字形有一个。行程中第 n（n>0）个字形的名义原点是第n-1个字形的名义原点加上沿行程前进矢量添加的第n-1个前进宽度。 基本字形一般具有非零的前进宽度，组合的字形一般具有零的前进宽度。	Indices 属性的一部分。见下文的表示。
GlhphOffsets	字形偏移数组。添加到上文计算的名义字形原点以生成字形的最终原点。基础字形一般具有(0, 0)的字形偏移，组合字形一般具有将它们正确地放置在最接近的前导基础字形上的偏移。	Indices 属性的一部分。见下文的表示。
GlyphTypeface	从其中能够取出该行程中的所有字形的物理字体。	FontUri 、FontFaceIndex 和StyleSimulations 属性
UnicodeString	可任选* 由该字形行程表示的字符数组。 *注意，对于从 Win32 打印机驱动程序生成的 GlyphRun，最初由 Win32 ExtTextOut(ETO_GLYPHINDEX) 调用打印的文本被传递到具有字形索引而没有 Unicode 码点的驱动程序。	是

	在这一情况下，生成的字形标记，以及构造的 GlyphRun 对象将省略码点。由于没有码点，固定格式察看器中诸如剪切和黏贴或搜索等功能不可用，然而文本显示保持可能。	
ClusterMap	对 Unicode 串中的每一字符有一个条目。 每一值给出了 GlyphIndices 中表示 Unicode 串中对应字符的第一个字形的偏移。 当多个字符映射到单个字形，或者单个字符映射到多个字形，或者多个字符不可见地映射到多个字形时，一个或多个字符以及一个或多个字形被称为群集。 ClusterMap 中用于多字符群集的所有条目映射到群集的第一个字形的 GlyphIndices 数组中的偏移。	Indices 属性的一部分。见下文的表示。
Sideways	字形在其一侧布局。 默认地，字形被呈现为它们是水平文本，其原点对应于西文的基线原点。如果设置了旁路标志，则字形在其一侧上转向，原点为未转向的字形的上中心。	是
BidiLevel	Unicode 算法的 bidi 嵌套级。在数字上偶数值暗示了从左到右布局，而在数字上奇数值暗示了从右到左布局。从右到左布局放置行程对远端在第一字形的右边，用前进矢量的正值，放置后来的字形到先前字形的左边。	是

Brush	用于绘制字形的前景刷。	从 ShapeFill 属性中选取。
Language	行程的语言，通常来自标记的 xml:lang 属性。	由 xml:lang 属性指定。

文本标记综述

字形度量

每一字形定义了指定它如何与其它字形对齐的度量。依照一个实施例的示例性度量在图 12 中示出。

前进宽度和组合记号

一般而言，字体内的字形是基础字形或可附加到基础字形的组合记号。基础字形通常具有非零的前进宽度，以及 0, 0 的偏移矢量。而组合记号通常具有零前进宽度。偏移矢量可用于调整组合记号的位置，并且因此对组合记号可具有非 0, 0 值。

字形行程中的每一字形具有控制其位置的三个值。这些值指示了原点、前进宽度和字形偏移，其每一个描述如下：

- **原点**：每一字形被假定给予一名义原点，对于行程中的第一个字形，这是行程的原点。
- **前进宽度**：每一字形的前进宽度提供了下一字形相对于该字形原点的原点。前进矢量总是在行程渐进的方向上绘制。
- **字形偏移（基础或记号）**：字形偏移矢量相对于其名义原点调整该字形位置。

字符、字形和群集映射

群集映射对每一 Unicode 码点包含一个条目。条目中的值是 GlyphIndices 数组中表示该码点的第一个字形的偏移。或者，当码点是一组表示不可见字符群集的码点的一部分时，GlyphIndices 数组中的第一个字形表示代表该群集的第一个字形的偏移。

群集映射

群集映射可表示码点－字形映射，它可以是一对一、多对一、一对多或多对多。一对一映射是每一码点仅由一个字形表示，图 13 中的群集映射条目是 0、1、2……。

多对一映射是两个或多个码点映射到单个字形。那些码点的条目指定了字形索引缓冲区中该字形的偏移。在图 14 的示例中，“f”和“i”字符用连字（ligature）来替换，这在许多衬线字体中是常见的排字惯例。

对于一对多映射，结合图 5 考虑以下内容。“Sara Am”包含位于先前的基础字符（环）上方的部分，以及位于基础字符（勾）右侧的部分。当对泰文文本进行微调时，将勾与基础字符分开，而环保留在基础字符的上方，因此许多字体将环和勾编码为单独的字形。当一个码点映射到两个或多个字形时，该码点的 ClusterMap（群集映射）中的值引用 GlyphIndeces 数组中表示该码点的第一个字形。

对于多对多映射，结合图 16 考虑以下内容。在某些字体中，字符群集的一组不可见码点映射到一个以上字形。例如，这在支持印度语脚本的字体中是常见的。当该组不可见码点映射到一个或多个字形时，每一码点的 ClusterMap 中的值引用 GlyphIndeces 数组中表示该码点的第一个字形。

以下示例示出了泰米尔单词கபாத的 Unicode 和字形表示。前两个码点组合在一起以生成三个字形。

指定群集

群集规范优于非 1:1 群集的第一个字形的字形规范（映射比一字符对一字形映射更复杂）。

每一群集规范具有以下形式：

（ClusterCodepointCount[:ClusterGlyphCount]）

群集规范部分	类型	目的	默认值
ClusterCodepointCount	正整数	组合在一起以形成该群集的 16 比特 Unicode 码点的数量	1
ClusterGlyphCount	正整数	组合在一起以形成该群集的 16 比特字形索引的数量	1

<Glyphs>标记

Glyphs 元素将字体指定为 URI、版面索引和一组上述的其它属性。例如：

```
<Glyphs
    FontUri              = "file://c:/windows/fonts/times.ttf"
    FontFaceIndex        = "0"              <!-- Default 0 ==>
    FontRenderingEmSize = "20"              <!-- No default -->
    FontHintingEmSize    = "12"             <!-- Default 12 -->
    StyleSimulations     = "BoldSimulation" <!-- Default None -->
    Sideways             = "false"          <!-- Default false -->
    BidiLevel            = "0"              <!-- Default 0 -->
    Unicode              = " ... "          <!-- Unicode rep -->
    Indices              = " ... "          <!-- See below -->
    remaining attributes ...
/>
```

每一字形规范具有以下形式：

[GlyphIndex][,[Advance][,[uOffset][,[vOffset][,[Flags]]]]]

字形规范的每一部分是可任选的：

字形规范部分	目的	默认值
GlyphIndex	呈现物理字体中的字形索引	如由内部文本中的对应 Unicode 码点的字体字符映射表所定义的。
Advance	下一字形相对于本字形的原点的放置。 在由 Sideway 和 BidiLevel 属性定义的前进方向上测量。 按字体 em 大小的 1/100 测量。 必须计算前进，使得舍入误差不会聚积。 见以下关于如何达到这一要求的注释。	如由字体 HMTX 或 VMTX 字体度量表定义的。
uOffset、vOffset	相对于字形原点的偏移以移动该字形。 通常用于将记号附加到基础字符。 按字体 em 大小的 1/100 测量。	0, 0
Flags	区别基础字形和组合记号	0（基础字形）

对于计算前进而没有舍入误差聚积，考虑以下内容。每一前进值必须被计算为后来的字形的未舍入原点减去先前字形的已计算（即，已舍入）前进宽度的总和。以此方式，每一字形被定位到其确切位置的 0.5% em 之内。

字形标记示例

```
<Canvas xmlns="http://schemas.microsoft.com/2005/xaml/">

<Glyphs
   FontUri             = "file://c:/windows/fonts/times.ttf"
   FontFaceIndex       = "0"
   FontRenderingEmSize = "20"
   FontHintingEmSize   = "12"
   StyleSimulations    = "ItalicSimulation"
   Sideways            = "false"
   BidiLevel           = "0"
   OriginX             = "75"
   OriginY             = "75"
   Fill                = "#00FF00"
   UnicodeString       = "inner text ..."
/>

<!-- 'Hello Windows' without kerning -->

<Glyphs
   OriginX             = "200"
   OriginY             = "50"
   UnicodeString       = "Hello, Windows!"
   FontUri             = "file://C:/Windows/Fonts/Times.TTF"
   Fill                = "#00FF00"
   FontRenderingEmSize = "20"
/>


<!-- 'Hello Windows' with kerning -->

<Glyphs
   OriginX             = "200"
   OriginY             = "150"
   UnicodeString       = "Hello, Windows!"
   Indices             = ";;;;;;;;,89"
   FontUri             = "file://C:/Windows/Fonts/Times.TTF"
   Fill                = "#00FF00"
   FontRenderingEmSize = "20"
/>


<!-- 'Open file' without 'fi' ligature -->

<Glyphs
   OriginX             = "200"
   OriginY             = "250"
   UnicodeString       = "Open file"
   FontUri             = "file://C:/Windows/Fonts/Times.TTF"
   Fill                = "#00FF00"
   FontRenderingEmSize = "20"
/>


<!-- 'Open file' with 'fi' ligature -->

<Glyphs
   OriginX             = "200"
   OriginY             = "350"
   UnicodeString       = "Open file"
   Indices             = ";;;;;(2:1)191"
   FontUri             = "file://C:/Windows/Fonts/Times.TTF"
   Fill                = "#00FF00"
   FontRenderingEmSize = "20"
/>

<!-- 'ежик в тумане' using pre-composed 'e' -->
```

```
<Glyphs
   OrigınX              = "200"
   OrigınY              = "450"
   xml:lang             = "ru-RU"
   UnıcodeString        = "ежик в тумане"
   FontUri              = "fıle://C:/Wındows/Fonts/Times.TTF"
   Fıll                 = "#00FF00"
   FontRenderingEmSize = "20"
/>

<!-- 'ежик в тумане' usıng compositıon of 'e' and diaeresıs -->

<Glyphs
   OriginX              = "200"
   OriginY              = "500"
   xml:lang             = "ru-RU"
   UnıcodeString        = "ежик в тумане"
   Indices              = "(1:2)72;142,0,-45"
   FontUrı              = "C:\/Wındows\/Fonts\/Times.TTF"
   Fıll                 = "#00FF00"
   FontRenderıngEmSize = "20"
/>


<!-- 'ежик в тумане' Forced renderıng rıght-to-left showıng
combınıng mark ın logıcal order -->

<Glyphs
   OrıginX              = "200"
   OrıginY              = "550"
   BıdıLevel            = "1"
   xml:lang             = "ru-RU"
   UnicodeString        = "ежик в тумане"
   Indıces              = "(1:2)72;142,0,-45"
   FontUri              = "fıle://C:/Wındows/Fonts/Times.TTF"
   Fıll                 = "#00FF00"
   FontRenderingEmSize = "20"
/>

</Canvas>
```

优化字形标记的大小

如果目标客户机能可靠地重新生成标记细节，如字形索引和前进宽度，则可从标记中省略它们。以下选项允许常用简单脚本的显著优化。

优化字形索引的标记

当在 Unicode 串中的字符位置和字形串中的字形位置之间有一对一映射，并且字形索引是字体的 CMAP（字符映射）表中的值，并且 Unicode 字符有明确的语义时，字形索引可以从标记中省略。

当字符到字形的映射时，应当在标记中提供字形索引：

- 不是一对一，如两个或多个码点形成单个字形（连字），或者

- 一个码点生成多个字形，或者
- 发生了任一其它形式的字形替换，如通过 OpenType 特征的应用。

当呈现引擎可能替换不同于字体的 CMAP（字符映射）表中字形的字形时，应当在标记中提供字形索引。当期望的字形表示不是字体的 CMAP 表中的字形时，应当提供字形索引。

字形位置的优化标记

当所需要的前进宽度的确是字体的 HMTX（水平度量）或 VMTX（垂直度量）表中的字形的前进宽度时，字形前进宽度可以从标记中省略。

当为零时，字形垂直偏移可以从标记中省略。这对于基础字符几乎总是真的，并且对于较简单的脚本中的组合记号，通常也是真。然而，对于诸如阿拉伯语和印度语等更复杂的脚本中的组合记号，这通常是假。

字形标志的优化标记

对于具有正常调整优先级的基础字形，字形标志可以被省略。

应用程序接口

以下描述了平台无关包装应用程序接口（API）的一个示例实施例。该 API 层由抽象类以及作为包装层的一部分包括在内的其它基类构成。该 API 包括以下讨论的类。

Container（容器）

容器是将部件集合容纳在一起的逻辑实体。

```
namespace System.IO.MMCF
{
  // The Container class represents a logical entity that holds together a collection of Parts.

  public class abstract Container : IDisposable
  {
    // Public Properties

    public virtual DateTime CreationTime { get; set; }    //File Properties.
    public virtual DateTime LastAccessTime { get; set; }    //File Properties.
    public virtual DateTime LastWriteTime { get; set; }    //File Properties.
    public FileAccess FileOpenAccess { get; }
    public abstract Part StartingPart { get; set; }

    // Public Static Methods

    public static Container OpenOnFile (string path);
    public static Container OpenOnFile (string path, FileMode mode);
    public static Container OpenOnFile (string path, FileMode mode, FileAccess access);
    public static Container OpenOnFile (string path,
```

```
                                        FileMode mode, FileAccess access, FileShare share);
    public static Container OpenOnUri (Uri uri);
    public static Container OpenOnUri (Uri uri, FileMode mode);
    public static Container OpenOnUri (Uri uri, FileMode mode, FileAccess access);

    public static Container OpenOnStream (Stream stream, string contentType);
    public static Container OpenOnStream (Stream stream, string contentType, FileMode mode);
    public static Container OpenOnStream (Stream stream, string contentType, FileMode mode,
                                          FileAccess access);
    // Public Methods

    public Part AddPart (MMCFUri uri, string contentType);
    public Part GetPart (MMCFUri uri);
    public virtual bool Exists (MMCFUri uri);
    public void DeletePart (MMCFUri uri);
    public PartCollection GetParts ();
    public void Flush ();
    public void Close ();
    public virtual void Dispose ();

    public Relationship AddRelationship (Uri uri);
    public void DeleteRelationship (Relationship relationship);
    public RelationshipCollection GetRelationships ();

    // Protected Methods - For Custom Implementation

    protected abstract Part AddPartCore (MMCFUri uri, string contentType);
    protected abstract Part GetPartCore (MMCFUri uri);
    protected abstract void DeletePartCore (MMCFUri uri);
    protected abstract Part [] GetPartsCore ();
    protected abstract void DeleteCore ();
    protected abstract void FlushCore ();

    // Protected constructor
    protected Container (FileInfo fileInfo, FileAccess access);

  }
}
```

构造函数

protected Container(FileInfo fileInfo, FileAccess access)

基类的受保护构造函数。明确地在该类中定义，因此能更容易地证明和维护该构造函数。如果不满足，则编译器将添加一构造函数。同样，这是抽象类和子类之间的当前合约。当 fileInfo 对象为空时，它定义了容器在流上被打开或创建。

属性

public virtual DateTime CreationTime{get; set}

获取或设置该容器的创建时间。当该值被设置时，也应当将 LastAccessTime（最后一次访问时间）和 LastWriteTime（最后一次写时间）更新为同一值。System_IO_FileInfo 对象用于操纵该值。

异常

－InvalidArgumentException（无效自变量异常）－如果 CreationTime 被设为大于 LastAccessTime 或 LastWriteTime 的值。

－InvalidOperationException（无效操作异常）－如果容器在流上打开，则无法获取

这一属性。

public virtual DateTime LastAccessTime{get; set;}

获取或设置该容器被最后一次打开的时间。System_IO_FileInfo 对象用于操纵该值。

异常

－InvalidArgumentException－如果 LastAccessTime 被设为小于 CreationTime 或 LastWriteTime 的值。

－InvalidOperationException－如果容器在流上打开，则无法获取该属性。

public virtual DateTime LastWriteTime{get; set}

获取或设置最后一次修改该容器的时间。同样，当更新 LastWriteTime 时，应当将 LastAccessTime 更新到同一值。System_IO_FileInfo 对象用于操纵该值。

异常

－InvalidArgumentException－如果 LastWriteTime 被设为小于 CreationTime 的值。

－InvalidOperationException－如果容器在流上打开，则无法获取该属性。

public FileAccess FileOpenAccess{get;}

获取用于打开容器的 FileAccesss（文件访问权限）。这是只读属性。该属性在打开容器时被设置。

public abstract Part StartingPart{get; set}

获取或设置容器的 StartingPart（起始部件）

方法

public static Container OpenOnFile(string path)

OpenOnFile 方法的该重载版本将返回在给定路径上指定的容器。该方法调用接受具有以下默认值的所有参数的重载。

FileMode（文件模式）－FileMode.OpenOrCreate

FileAccess（文件访问权限）－FileAccess.ReadWrite

FileShare（文件共享）－FileShare.None

public static Container OpenOnFile(string path, FileMode mode)

OpenOnFile 方法的这一重载版本将以指定的文件模式返回在给定路径上指定的容器。该方法调用接受具有以下默认值的所有参数的重载

FileAccess－FileAccess.ReadWrite

FileShare－FileShare.None

public static Container OpenOnFile(string path, FileMode mode, FileAccess access)

OpenOnFile 的该重载版本将以指定的文件模式和文件访问返回在给定路径上指定的容器。该方法调用接受具有以下默认值的所有参数的重载

FileShare－FileShare.None

public static Container OpenOnFile(string path, FileMode mode, FileAccess access, FileShare share)

OpenOnfile 方法的该重载版本将用设为提供的值的模式、访问和共享来打开给定路径上的容器。

异常

－InvalidArgumentException－如果 FileMode、FileAccess 和 FileShare 参数的组合无意义。

public static Container OpenOnUri(Uri uri)

OpenOnUri 方法的该重载版本将返回在给定 uri 处指定的容器。该方法调用接受具有以下默认值的所有参数的重载

FileMode－FileMode.Open

FileAccess－FileAccess.Read

public static Container OpenOnUri(Uri uri, FileMode mode)

OpenOnUri 的该重载版本将以指定的文件模式返回给定 uri 处指定的容器。该方法调用接受具有以下默认值的所有参数的重载

FileAccess－FileAccess.Read

public static Container OpenOnUri(Uri uri, FileMode mode, FileAccess access)

OpenOnUri 方法的该重载版本将用设为提供的值的模式和访问打开给定 uri 处的容器。WebRequest/WebResponse（web 请求/web 响应）机制将用于获取容器。FileMode 和 FileAccess 参数将应用于要打开的容器。该方法调用具有正确内容类型的 OpenOnStream 方法。

异常

－InvalidArgumentException－如果 FileMode、FileAccess 和 FileShare 参数的组合无意义。

public static Container OpenOnStrem(Stream stream, string contentType)

OpenOnStream 的该重载版本将在提供的流上返回容器。该方法调用接受具有以下默认值的所有参数的重载

FileMode－FileMode.Open

FileAccess－FileAccess.Read

public static Container OpenOnStream(Stream stream, string contentType, FileMode mode)

OpenOnStream 的该重载版本将以指定的文件模式在提供的流上返回容器。该方法调用接受具有以下默认值的所有参数的重载

FileAccess－FileAccess,Read

public static Container OpenOnStream(Stream stream, string contentType, FileMode mode, FileAccess access)

OpenOnStream 的该重载版本将用设为提供的值的模式和访问在提供的流上打开容器。FileMode 和 FileAccess 参数将被应用于要打开的容器。contentType 参数用于例示适当的子类对象。

异常

－InvalidArgumentException－如果 FileMode、FileAccess 和 FileShare 参数额度组合无意义。

public Part AddPart(MMCFUri uri, string contentType)

给定 Uri 的部件被添加到容器。如果未作出明确的调用来读取或写入流，则该方法将用空流来添加部件。该方法调用完成与物理实现相关的实际工作的 AddPartCore。

异常

－InvalidArgumentException－如果对应于该 Uri 的部件已在容器中存在。

public Part GetPart(MMCFUri uri)

返回给定 Uri 的部件。uri 相对于容器的根。该方法调用实际获取部件的 GetPartCore。

异常

－InvalidArgumentException－如果对应于该 Uri 的部件在容器中不存在。

public virtual bool Exists(MMCFUri uri)

由于可能令关系指向仍不存在的目标，因此该方法提供了一种方便的方法来找出部件是否在底层容器中实际存在。该 uri 应当相对于容器的根。

public void DeletePart(MMCDUri uri)

该方法将从当前容器中删除容器部件。对于其该部件是源部件的所有关系也被删除。该方法将删除底层的流，并且将处置对象。同样，如果打开了该部件的多个实例，则处置该部件的所有打开的实例。该方法将做必要的清理来实施这一行为，然而流的实际删除对于底层物理实现是专用的，并因此调用了删除实际流的 DeletePartCore 方法。未完成的部件枚举器将被无效。

public PartCollection GetParts ()

这返回容器内所有部件的集合。不返回关系。

public void Flush()

该方法在打开的个别部件上调用清洗，由此强制了所有的部件和关系被清洗到底层容器。本质上该类将维护它分发的所有部件的数组，然后将在所有的部件上调用 Flush。它然后调用完成对整个容器专用的工作的 FlushCore()。

public virtual void Dispose()

所有打开的部件和关系都被清洗到底层容器。由于该类维护分发的所有部件的数组，因此该方法将在分发的所有部件上调用 Dispose()。如果任一其它资源需要被清洗，则子类应当覆盖该方法以完成附加的清洗。

public void Close()

Close 方法与处置相同，因此它内部地调用了 Dispose()方法。

public Relationship AddRelationship(Uri uri)

该方法添加了容器和由 URI 指定的部件之间的关系。它返回 Relationship 对象。该改变进在调用了 Flush()方法之后被清洗到底层容器。在同一源和目标之间可以有多个关系。所有未完成的关系枚举器将被无效。

public void DeleteRelationship(Relationship relationship)

该方法删除由 Relationship 对象指定的目标关系。该改变进在调用了 Flush()方法之后被清洗到底层容器。删除操作基于对象的“名字”来完成，并且因此，每一对象被唯一地标识。所有未完成的关系枚举器被无效。

异常

－InvalidArgumentException－如果关系的源不与当前部件相同。

public RelationshipCollection GetRelationships()

这从容器返回所有目标关系的集合。当在公知的 uri 上定位了容器的目标关系，则可能提供一默认实现，它将打开关系部件，然后从流中读取 xml 并创建集合对象（异常－如果从底层容器读取的 XML 是畸形的。）

protected abstract Part AddPartCore(MMFCUri uri, string contentType)

该方法用于底层文件格式的自定义实现。它将从 AddPart 方法中调用。这将实际在底层容器中创建部件。空部件应当作为该调用的结果而被创建。

protected abstract Part GetPartCore(MMCFUri uri)

该方法用于底层文件格式的自定义实现。它将从 GetPart 方法中调用。该方法从底层容器中取出实际的部件。如果部件不存在，则它返回“空”。

protected abstract void DeletePartCore(MMCFUri uri)

该方法用于底层文件格式的自定义实现。它应当实际删除对应于该部件的流。同样，如果不存在对应于给定 URI 的部件，则它应当不抛出。

protected abstract Part[] GetPartsCore()

该方法用于底层文件格式的自定义实现。它应当返回容器中所有部件的数组。由于获取容器中所有部件的方法对实际物理格式是专用的，因此该方法是游泳的。提供了该方法，使得实际的 GetParts 调用仅将该数组传递到 PartCollection，并且可提供其上的枚举器。以此方式，PartCollection 类可以是有形的类。同样，如果容器中没有部件，则 GetPartsCore 应当返回空数组。

protected abstract void FlushCore()

该方法用于底层文件格式的自定义实现。该方法将所有的内容清洗到盘。

protected abstract void DisposeCore()

该方法用于底层文件格式的自定义实现。该方法应当释放对应于实际物理格式的资源。

Part（部件）

部件包括三个部分：

- URI－相对于容器的根
- ContentType－它是由该部件表示的流的模仿类型
- Stream－对应于该部件的实际流

另外部件可用关系被链接到其它部件。关系中的 SourcePart（源部件）拥有该关系。

```
namespace System.IO.MMCF
{
  // This class represents a Part which consists of an Uri, ContentType and an underlying stream.
  // A part can be connected to other part using Relationships.

  public class abstract Part
  {
    //Public Properties

    public MMCFUri Uri { get; }
    public string ContentType { get; }
    public Container Container { get; }

    // Public Methods

    public Stream GetStream ();
    public Stream GetStream (FileMode mode);
    public Stream GetStream (FileMode mode, FileAccess access);
    protected abstract Stream GetStreamCore (FileMode mode, FileAccess access);


    public Relationship AddRelationship (Uri uri);
    public void DeleteRelationship (Relationship relationship);
    public RelationshipCollection GetRelationships ();

    // Protected constructor
    protected Part (Container container, MMCFUri uri, string contentType);

  }
}
```

构造函数

protected Part(Container container, MMCFUri uri, string contentType)

用于基类的受保护的构造函数。在该类中明确地定义，使得能更容易地证明和维护该构造函数。如果不满足，则编译器将添加一构造函数。这也是抽象类和子类之间的当前合约。

属性

public MMCFUri Uri{get;}

该属性返回部件的 MMCFUri。这是只读属性。

public string ContentType{get;}

该属性返回由部件表示的流的内容类型。这是只读属性。

public Container Container{get;}

该属性返回部件的父容器。这是只读属性。

方法

public Stream GetStream()

该方法返回对应于该部件的流。它调用接受具有以下默认值的所有参数的重载

FileMode－Open

FileAccess－ReadWrite

public Stream GetStream(FileMode mode)

该方法以指定的模式返回对应于该部件的流。它调用接受具有以下默认值的所有参数的重载

FileAccess－ReadWrite

public Stream GetStream(FileMode mode, FileAccess access)

该方法返回对应于该部件的流。它调用返回实际流的 GetStreamCore 方法。该方法完成所需的内务处理以跟踪所有打开的流。

public abstract Stream GetStreamCore(FileMode mode, FileAccess access)

该方法返回对应于该部件的流。该方法用于自定义实现。

public Relationship AddRelationship(Uri uri)

该方法添加指定的 URI 处的部件和当前部件之间的关系。它返回 Relationship 对象。该改变仅在调用了 Flush()方法之后被清洗到底层容器。在同一源和目标之间可以有多个关系。所有未完成的关系枚举器将被无效。

异常

－InvalidOperationException－如果当前部件是关系。

public void DeleteRelationship(Relationship relationship)

该方法删除由 Relationship 对象指定的目标关系。该改变仅在调用了 Flush()方法之后被清洗到底层容器。删除操作基于对象的“引用”来完成，并且因此每一对象被唯一地标识。所有未完成的关系枚举器被无效。

异常

－InvalidArgumentException－如果关系的源不与当前部件相同。

public RelationshipCollection GetRelationships()

这返回该部件的所有目标关系的集合。当在公知的 uri 处定位了该部件的目标关系时，可能提供一默认实现，它打开关系部件，然后从流中读取 xml 并创建对象集合（异常－如果从底层容器读取的 XML 是畸形的。）

Relationship（关系）

该类用于表达源和目标部件之间的关系。创建关系的唯一方法是调用 Part.AddRelationship(Uri uri)。关系由源部件拥有，因此如果源部件被删除，则它所拥有的所有关系也被删除。关系的目标不必要存在。

```
namespace System.IO.MMCF
{
  // This class represents a relationship between a source part and a target part. The only way
  // to create a Relationship is to call Part.AddTargetRelationship (Uri uri). A relationship is
  // owned by the source part. So if the source part is deleted all the relationships it owns
  // also get deleted.

  public class Relationship
  {
    //Public Properties

    public Part Source { get;  }
    public Uri TargetUri { get; }
    public string Name { get;}

    //internal Constructors
    internal Relationship (Part source, Uri target, string name);

  }
}
```

构造函数

internal Relationship(Uri source, Uri target, string name)

返回 Relationship 对象。

属性

public Part Source{get;}

获取关系的源部件。这是只读属性。当创建关系时设置该属性。

public Uri TargetUri{get;}

获取关系的目标 Uri。该 Uri 应当被看作相对于源 uri。

public string Name{get;}

获取对应于该关系的名字。

PartCollection（部件集合）

这是容器中部件的集合。

```
namespace System.IO.MMCF
{
  // This is a strongly typed collection of Parts

  public class PartCollection : IEnumerable
  {
    //IEnumerable Member
    public IEnumerator GetEnumerator ();

    //Internal Constructors
    internal PartCollection (Dictionary<MMCFUri, Part> partDictionary);

   }
}
```

构造函数

internal PartCollection(Dictionary<MMCFUri, Part> partDictionary)

基于 Part 对象的类属字典创建 PartCollection。

方法

public IEnumerator GetEnumerator()

IEnumerable 接口的成员。它返回部件集合上的枚举器。

RelationshipColection（关系集合）

这是与容器中部件相关联的关系的集合。在给定的源和目标部件之间可以有一个以上关系。

```
namespace System.IO.MMCF
{
  // This is a strongly typed collection of Relationships

  public class RelationshipCollection : IEnumerable
  {
    //IEnumerable Member
    public IEnumerator GetEnumerator();

    //Internal Constructors
    internal RelationshipCollection (Relationship [] relationships);

   }
}
```

构造函数

internal RelationshipCollection(Relationship[] relationships)

基于 Relationship 对象创建 RelationshipCollection

方法

public IEnumerator GetEnumerator()

MMFCUri

该类从 URI 类继承。该 Uri 类的主函数是确保指定的 URI 以“/”开始。该类的动机是：

1. 确保用于每一个别部件的 URI 以“/”开始。这确保所有的部件名相对于容器的根。
2. 由于 System.Uri 类不允许解析两个相对 URI，因此他们需要以不同的方式来解析，因此在一个地方有这一逻辑是较佳的。
3. 强制容器是授权机构这一事实。由此，任何相对引用应当不被解析成容器外部的位置。
4.

```
namespace System.IO.MMCF
{
  // This class is used to create URIs that always start with a "/"

  public class MMCFUri : Uri
  {

    //Public Constructors

    public MMCFUri (string uri);
    public MMCFUri (MMCFUri baseUri, string relativeUri);

   }
}
```

构造函数

public MMCFUri(string uri)

从提供的 uri 串创建 MMCFUri。确保该 Uri 是相对的且以“/”开始。

异常－InvalidArgumentException－如果 URI 包括主机名和协议，即，它是绝对 URI。

public MMCFUri(MMCFUri baseUri, string relativeUri)

从提供的 Uri 对象和 relativeUri（相对 URI）串创建 MMCFUri 对象。相对于 baseUri（基础 URI）解析相对 uri。确保 Uri 是相对的，且以“/”开始。

异常－InvalidArgumentException－如果 URI 包括主机名和协议，即它是绝对 URI。

代码示例

```
namespace System.IO.MMCF
{
  public class Sample :
  {

    // Creates a container and adds a part to it.

    Container c = new Container.OpenOnFile("myFilePath");
    Part p1 = c.AddPart("mypart1", "contentType"); //Creates empty stream
    c.Close();

    // Opens the same container and adds another part and a relationship

    c = new Container.OpenOnFile("myFilePath");

    Part p1 = c.GetPart("mypart1");
    Part p2 = c.AddPart("mypart2", "contentType"); //Creates empty stream

    RelationshipInfo ri1 = p1.AddRelationship(p2.Uri);

    c.Close();

  }
}
```

其它 API 细节

OpenOnFile、OpenOnUri 和 OpenOnStream 方法

这些方法具有硬编码的逻辑，并且这些方法知道的唯一物理容器格式是复合文件实现。由于拥有这些类，因此有关于从这些静态方法调用的子类构造函数的假设。同样，这些静态方法基于文件扩展名或当前流的内容类型例示正确的子类对象。

OpenOnStream 方法和对容器指定的访问的含义

当在流上创建容器时，重要的是确保对容器指定的 FileAccess 与提供的流兼容。以下表格列出了各种可能性以及如何处理它们的示例。

	当前流访问		需要的容器访问		
	能读	能写	读	写	读写
1.	真	真	限于只读	限于只写	OK
2.	假	真	抛出	OK	抛出
3.	真	假	OK	抛出	抛出

4. 假 假 抛出 抛出 抛出

在第一行中，流具有更多的访问，而希望创建更受限制的容器，因此用称为RestrictedStream 的私有流包装传入的流，该私有流具有适当的能读和能写值。

Part 和 Relationship 对象的存储器中高速缓存

字典维持所有的部件可访问，并且如果第二次要求部件，则返回从字典对该部件的引用。这是更有效的，并且因为 Part 对象是不变的，这可以没有任何问题地完成。它也应用于 Relationship 对象。然而，如果底层容器以共享的写模式打开，并且有第二用户向底层容器添加或删除了部件，则这些改变将不会在高速缓存中得到反映。

总结

上述模块化内容框架和文档格式方法和系统提供了一组构件块，用于组成、包装、分发和呈现以文档为中心的内容。这些构件块定义了用于文档格式的平台无关框架，使软件和硬件系统能够可靠和一致地生成、交换和显示文档。所示并描述的到达包格式提供了一种用于以可用各种各样环境中的设备和应用程序之间的完全保真度并且跨各种各样情形来显示或打印到达包的内容的方式储存已编页码或预编页码的文档的格式。尽管以对结构特征和/或方法步骤专用的语言描述了本发明，然而可以理解，所附权利要求书中定义的本发明不必要限于所描述的特征或步骤。相反，揭示了具体特征和步骤作为实现要求保护的本发明的较佳形式。

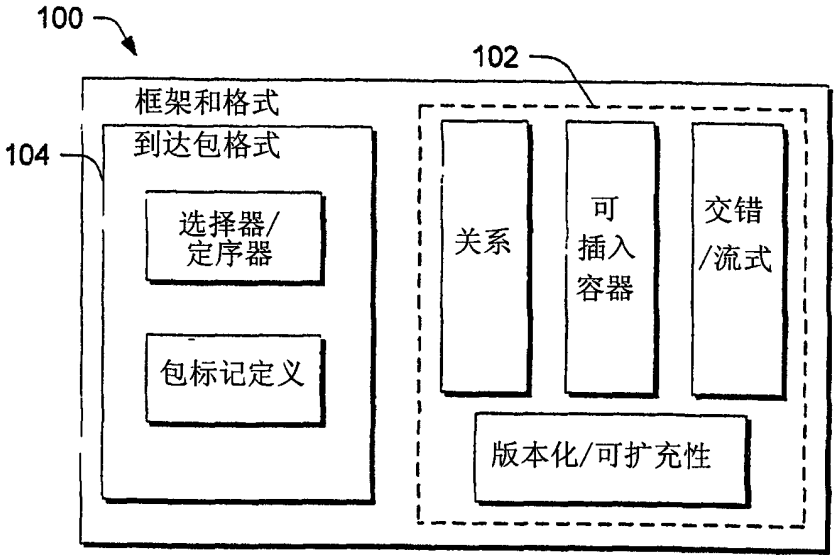

100
102
框架和格式
到达包格式
104
选择器/
定序器
包标记定义
关系
可
插入
容器
交错
/流式
版本化/可扩充性

图 1

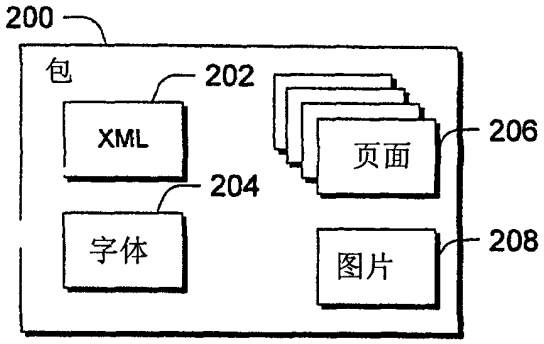

200
包
202
XML
204
字体
页面
206
图片
208

图 2

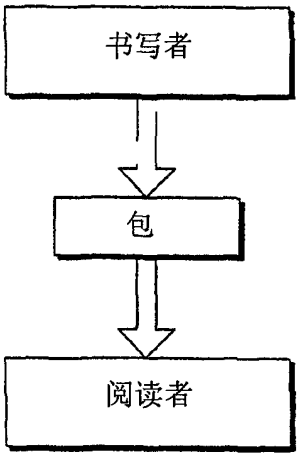


图 3

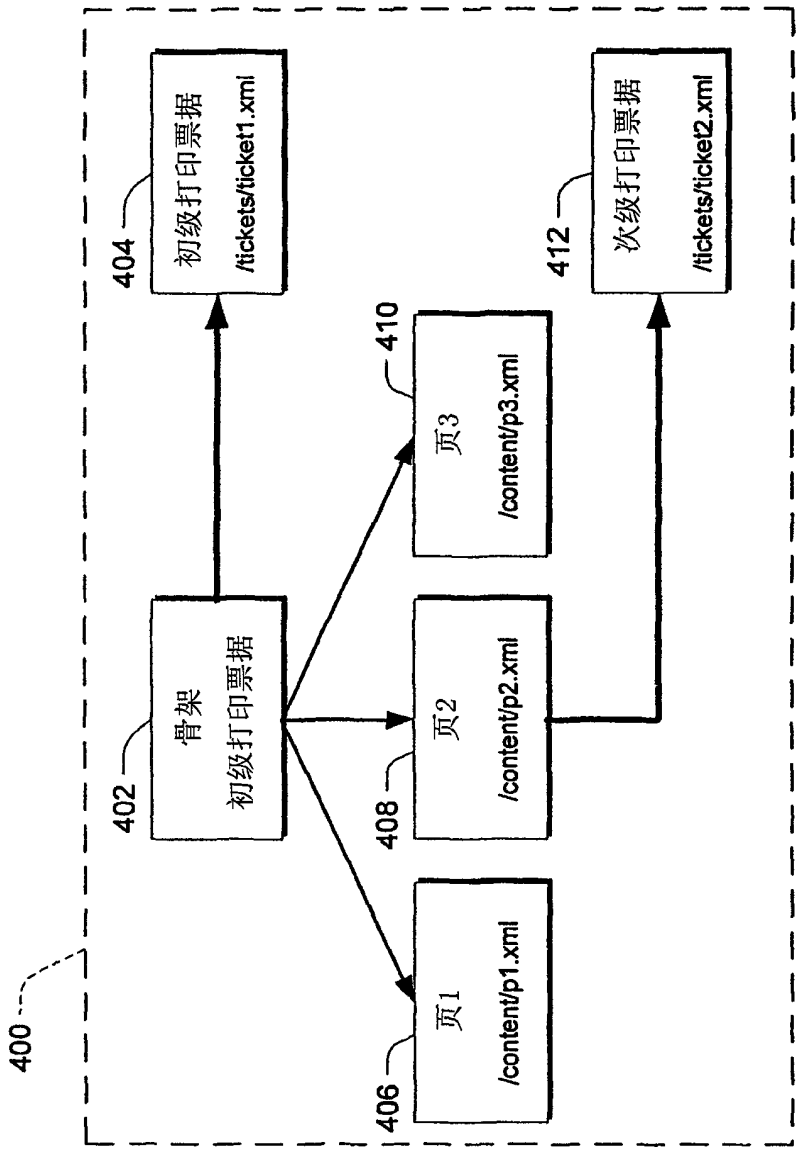

400
骨架
初级打印票据
402
初级打印票据
/tickets/ticket1.xml
404
页1
/content/p1.xml
406
页2
/content/p2.xml
408
页3
/content/p3.xml
410
次级打印票据
/tickets/ticket2.xml
412

图 4

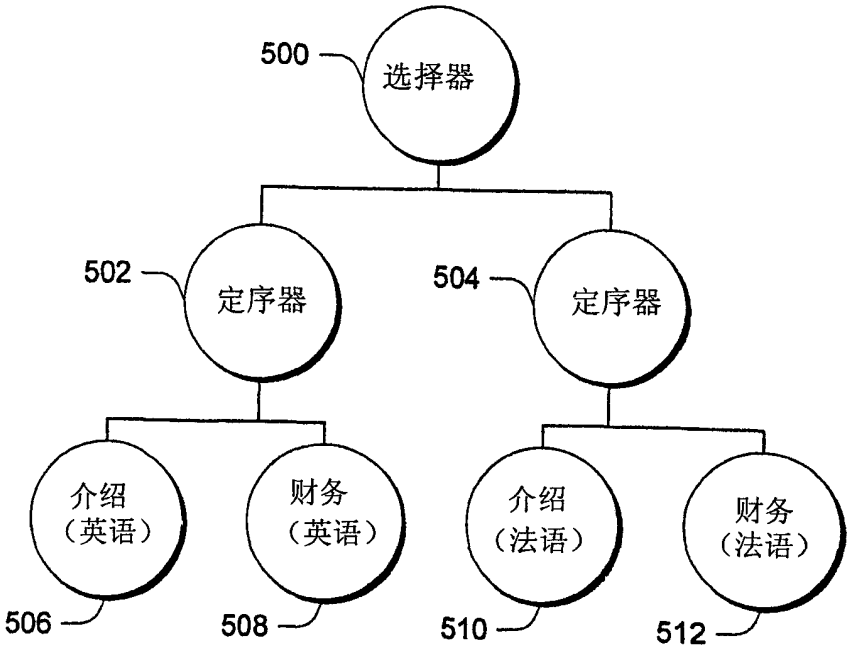


图 5

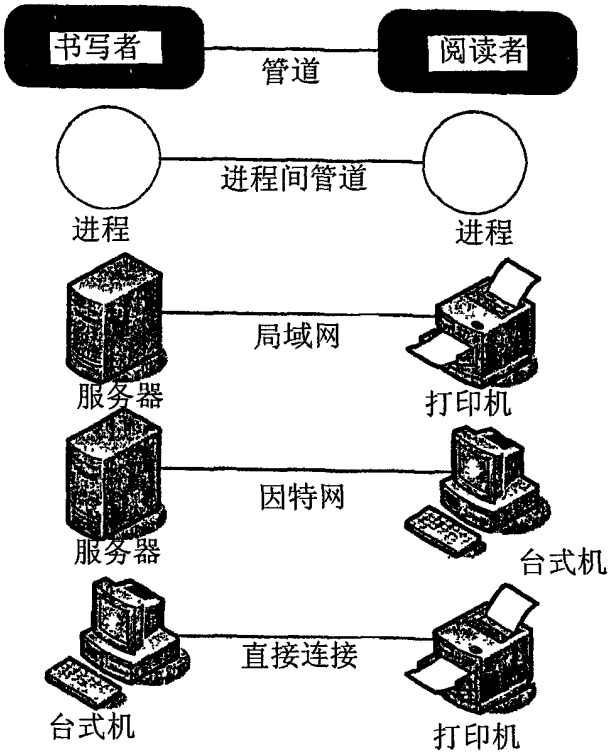


图 6

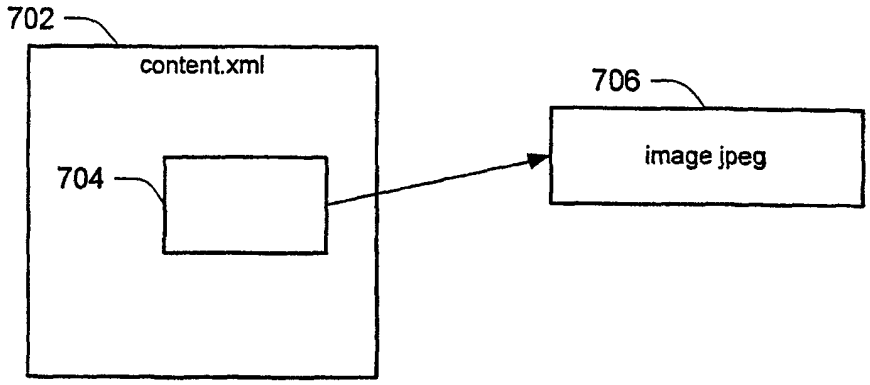

702
content.xml
706
704
image jpeg

图 7

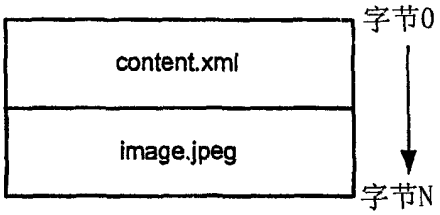

字节0
content.xml
image.jpeg
字节N

图 8

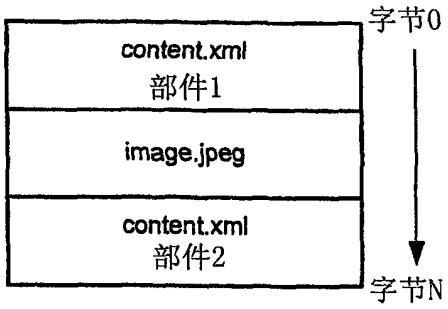

字节0
content.xml
部件1
image.jpeg
content.xml
部件2
字节N

图 9

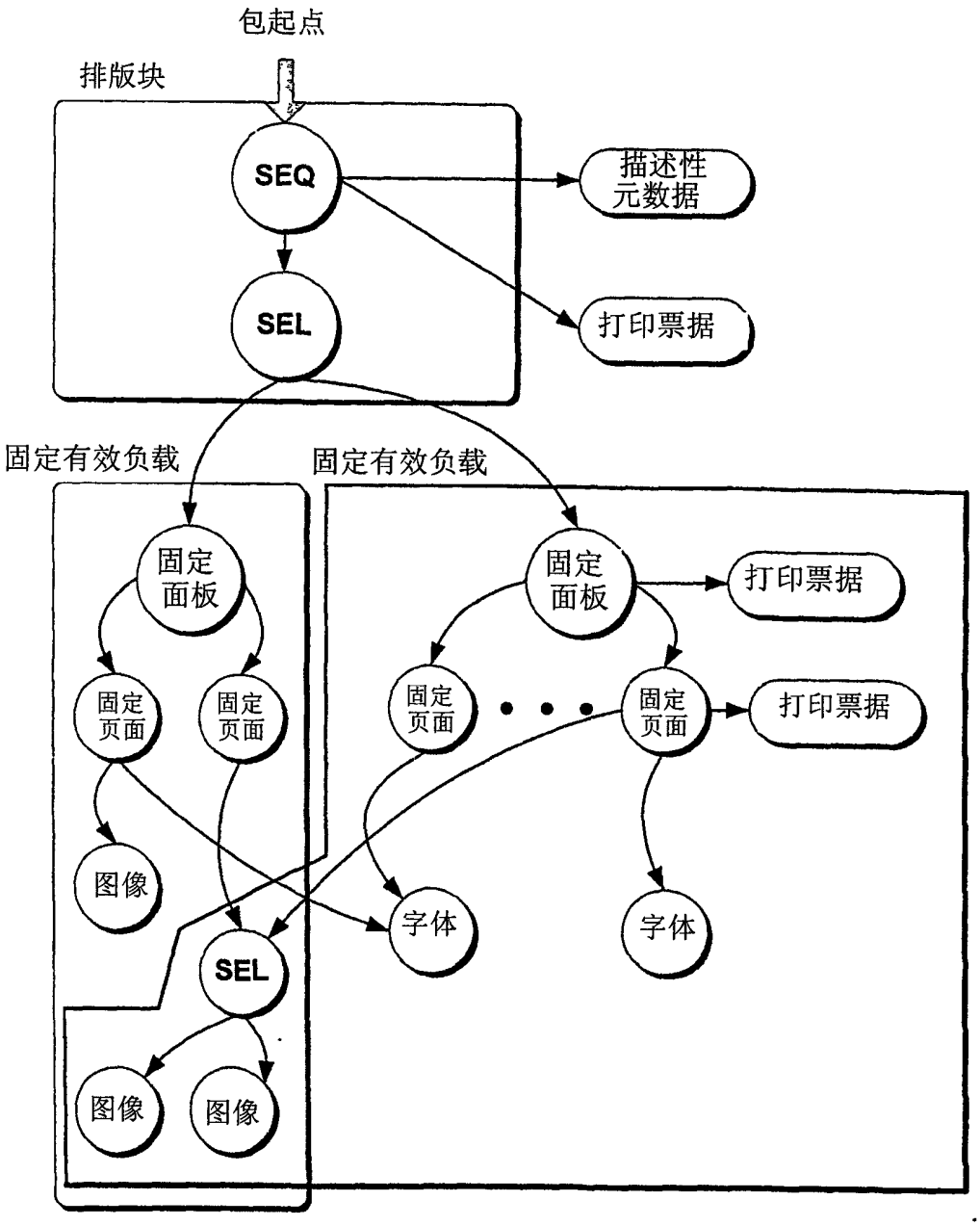

包起点
排版块
SEQ
描述性元数据
SEL
打印票据
固定有效负载
固定有效负载
固定面板
固定面板
打印票据
固定页面
固定页面
固定页面
打印票据
图像
字体
字体
SEL
图像
图像

图 10

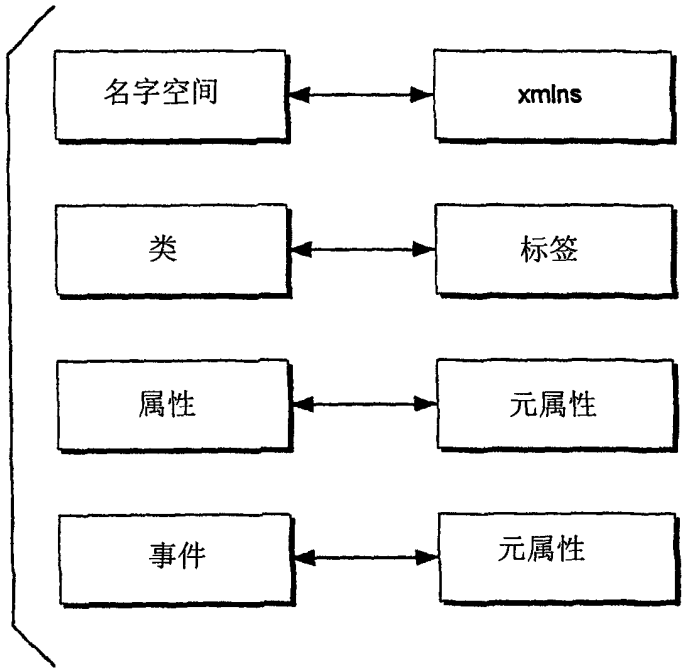


图 11

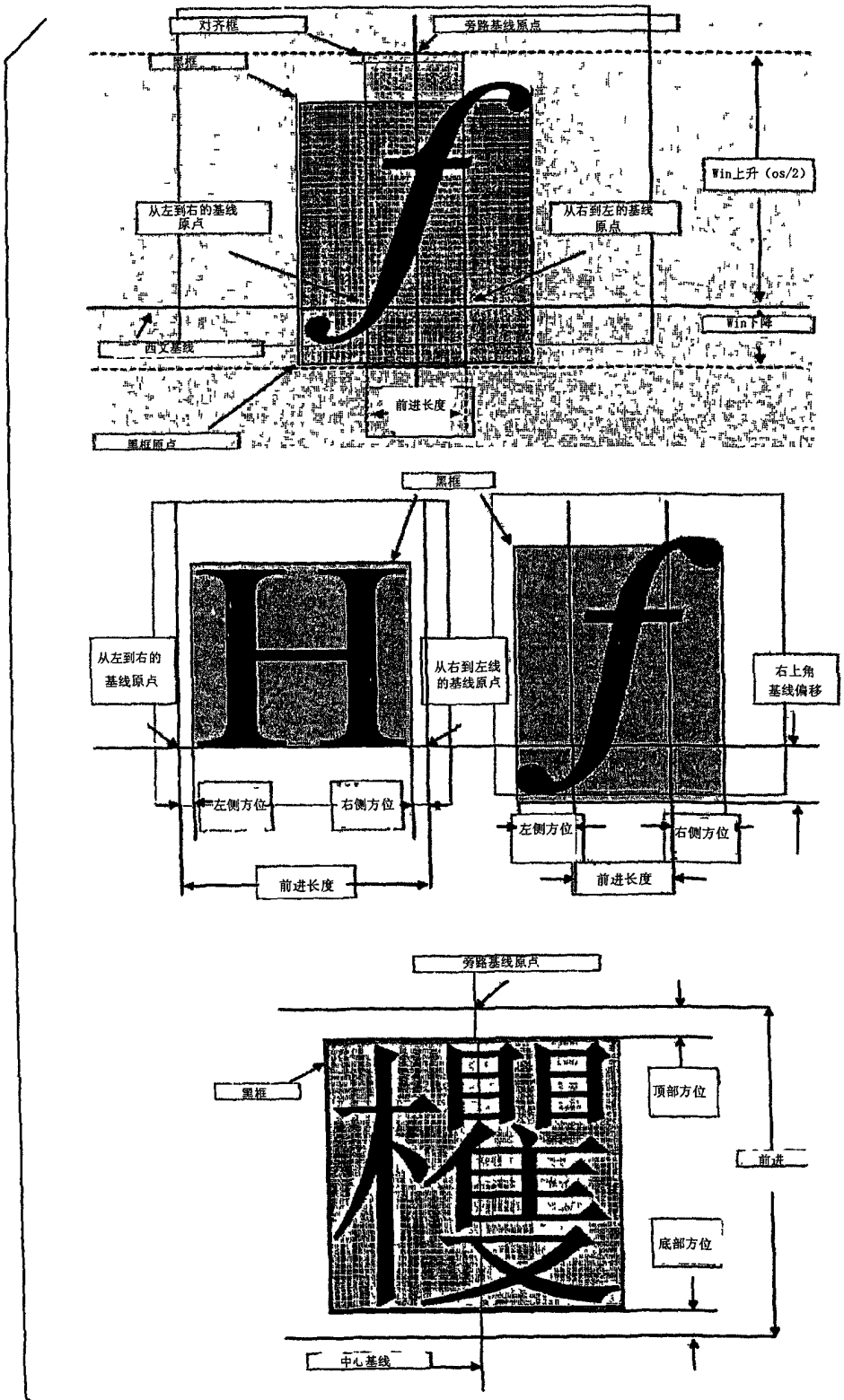

对齐框
旁路基线原点
黑框
Win上升（os/2）
从左到右的基线原点
从右到左的基线原点
Win下降
西文基线
前进长度
黑框原点
黑框
从左到右的基线原点
从右到左线的基线原点
右上角基线偏移
左侧方位
右侧方位
左侧方位
右侧方位
前进长度
前进长度
旁路基线原点
黑框
顶部方位
前进
底部方位
中心基线

图 12

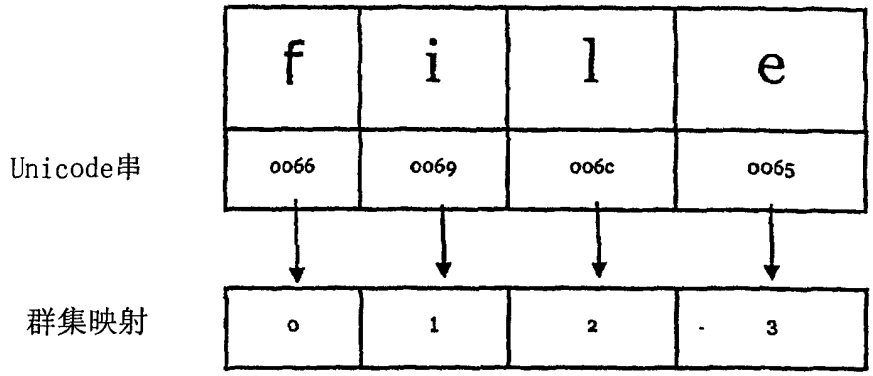

f
i
l
e
Unicode串
0066
0069
006c
0065
群集映射
0
1
2
3

图 13

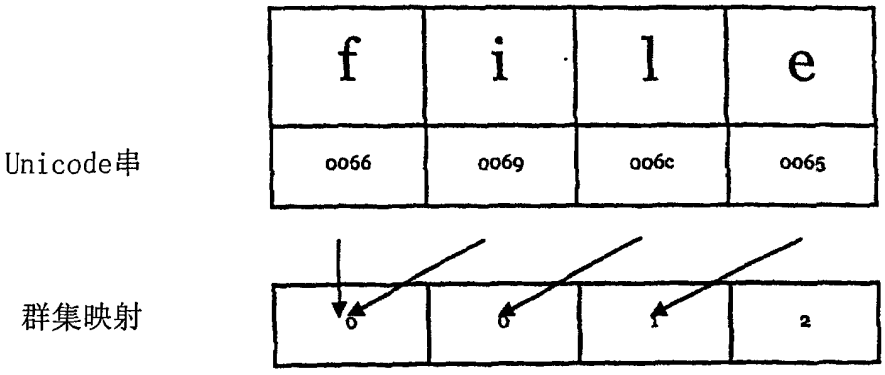

f
i
l
e
Unicode串
0066
0069
006c
0065
群集映射
0
0
1
2

图 14

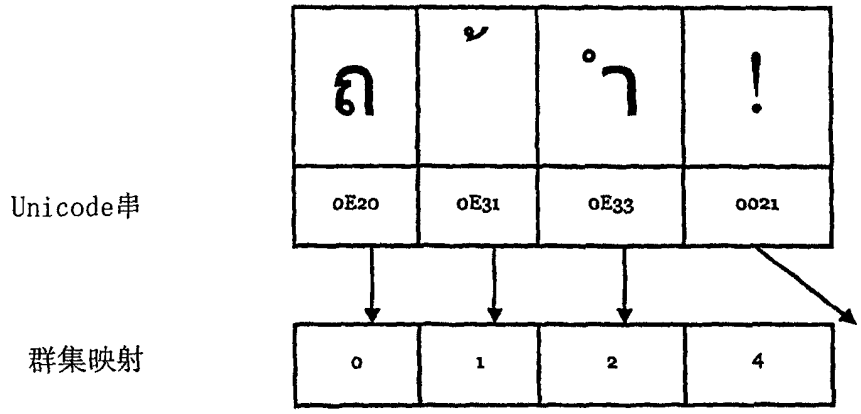

Unicode串
0E20
0E31
0E33
0021
群集映射
0
1
2
4

图 15

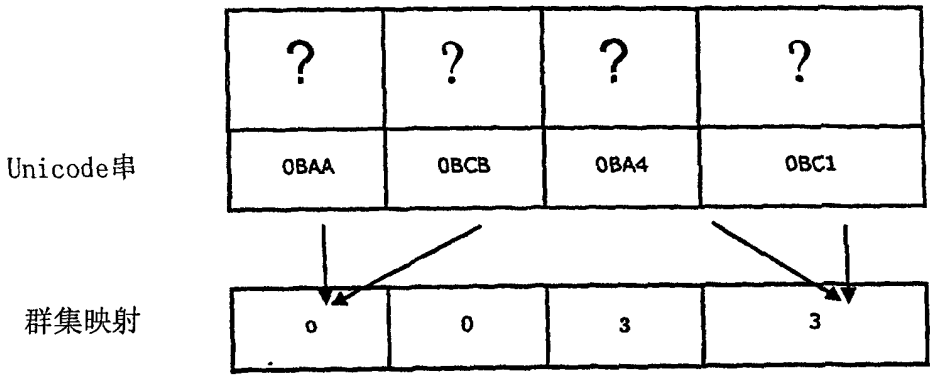

Unicode串
0BAA
0BCB
0BA4
0BC1
群集映射
0
0
3
3

图 16