РОССИЙСКАЯ ФЕДЕРАЦИЯ

ФЕДЕРАЛЬНАЯ СЛУЖБА
ПО ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ,
ПАТЕНТАМ И ТОВАРНЫМ ЗНАКАМ

(19) **RU** (11) **2 368 943** (13) **C2**

(51) МПК
G06F 15/00 (2006.01)
G06F 17/00 (2006.01)

(12) ОПИСАНИЕ ИЗОБРЕТЕНИЯ К ПАТЕНТУ

(21), (22) Заявка: **2005120697/09, 29.07.2004**

(24) Дата начала отсчета срока действия патента:
29.07.2004

(30) Конвенционный приоритет:
30.04.2004 US 10/836,326

(43) Дата публикации заявки: **10.04.2006**

(45) Опубликовано: **27.09.2009** Бюл. № **27**

(56) Список документов, цитированных в отчете о поиске: **US 2003/0233420 A1, 18.12.2003. RU 2222045 C2, 20.01.2004. US 2002/0099797 A1, 25.07.2002. WO 00/68817 A1, 16.11.2000.**

(85) Дата перевода заявки PCT на национальную фазу: **29.06.2005**

(86) Заявка PCT:
US 2004/024444 (29.07.2004)

(87) Публикация PCT:
WO 2005/111830 (24.11.2005)

Адрес для переписки:
129090, Москва, ул. Б.Спасская, 25, стр.3, ООО "Юридическая фирма Городисский и Партнеры", пат.пов. Ю.Д.Кузнецову, рег.№ 595

(72) Автор(ы):
ШУР Эндрю (US),
ДУНИТЦ Джерри (US),
ФЕР Оливер (US),
ЭМЕРСОН Дэниэл (US),
ХИЛЛБЕРГ Майк (US),
КИМ Янг Гах (US),
ПОЛЛОКК Джош (US),
ШИТ Сарджана (US),
ОРНСТАЙН Дэвид (US),
ПАОЛИ Джин (US),
ДЖОНС Брайан (US)

(73) Патентообладатель(и):
МАЙКРОСОФТ КОРПОРЕЙШН (US)

(54) МОДУЛЬНЫЙ ФОРМАТ ДОКУМЕНТОВ

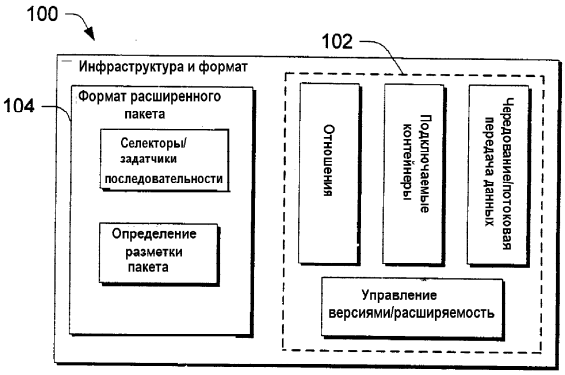
(57) Реферат:

Изобретение относится к способам и системам модульной инфраструктуры содержимого и формату документов. Изобретение обеспечивает платформно-независимую инфраструктуру для форматов документов, которая дает возможность генерировать и отображать документы надежным и единообразным образом. Описанные инфраструктура и формат определяют набор стандартных компоновочных блоков для составления, пакетирования, распространения и визуализации документно-ориентированного содержимого. Инфраструктура и формат были разработаны гибким и расширяемым образом. В дополнение к этим общим инфраструктуре и формату конкретный формат, известный как формат расширенного пакета, определяется с использованием общей инфраструктуры. Формат расширенного пакета представляет собой формат для хранения разбитых на страницы документов. Содержимое расширенного пакета может отображаться или печататься с высокой точностью воспроизведения на устройствах и в

RU 2368943 C2

RU 2368943 C2

приложениях в широком спектре сред и в широком спектре сценариев. 6 н. и 18 з.п. ф-лы, 16 ил.



ФИГ.1

RU 2368943 C2

RU 2368943 C2

RUSSIAN FEDERATION

FEDERAL SERVICE
FOR INTELLECTUAL PROPERTY,
PATENTS AND TRADEMARKS

(19) **RU** (11) **2 368 943** (13) **C2**

(51) Int. Cl.
G06F 15/00 (2006.01)
G06F 17/00 (2006.01)

(12) ABSTRACT OF INVENTION

(21), (22) Application: **2005120697/09, 29.07.2004**

(24) Effective date for property rights:
29.07.2004

(30) Priority:
30.04.2004 US 10/836,326

(43) Application published: **10.04.2006**

(45) Date of publication: **27.09.2009 Bull. 27**

(85) Commencement of national phase: **29.06.2005**

(86) PCT application:
US 2004/024444 (29.07.2004)

(87) PCT publication:
WO 2005/111830 (24.11.2005)

Mail address:
129090, Moskva, ul. B.Spasskaja, 25, str.3, OOO "Juridicheskaja firma Gorodisskij i Partnery", pat.pov. Ju.D.Kuznetsovu, reg.№ 595

(72) Inventor(s):
ShUR Ehndrju (US),
DUNITTs Dzherri (US),
FER Oliver (US),
EhMERSON Dehniehl (US),
KhILLBERG Majk (US),
KIM Jang Gakh (US),
POLLOKK Dzhosh (US),
ShIT Sardzhana (US),
ORNSTAJN Dehvid (US),
PAOLI Dzhin (US),
DZhONS Brajan (US)

(73) Proprietor(s):
MAJKROSOFT KORPOREJShN (US)

RU 2368943 C2

(54) MODULAR DOCUMENT FORMAT

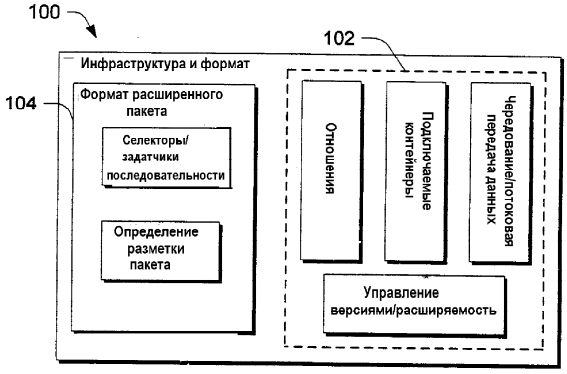
(57) Abstract:

FIELD: information technology.

SUBSTANCE: invention relates to methods and systems for modular framework of content and format of documents. Proposed framework and format define a set of standard assembly units for compiling, packetisation, distribution and imaging document-oriented content. The framework and format were designed in a flexible and extensible way. In addition to this general framework and format, a specific format, known as extended packet format, is determined using the general framework. The extended packet format is a format for storing paginated documents. Extended packet content can be displayed or printed with high reproduction accuracy on devices and in applications in a wide spectrum of media and in a wide spectrum of scenarios.

EFFECT: invention provides for platform-independent framework for document formats, which makes possible reliable and uniform generation and display of documents.

24 cl, 16 dwg



ФИГ.1

RU 2368943 C2

Область техники, к которой относится изобретение

Настоящее изобретение относится к инфраструктуре содержимого, формату документов и соответствующим способам и системам, которые могут их использовать.

Предшествующий уровень техники

В настоящее время обычно существует множество различных типов инфраструктур содержимого для представления содержимого и множество различных типов форматов документов для форматирования различных типов документов. Зачастую каждая из этих инфраструктур и форматов требует свое собственное сопоставленное программное обеспечение для сборки, получения, обработки или потребления ассоциированного документа. Для тех, кто установил конкретное ассоциированное программное обеспечение на соответствующем устройстве, сборка, получение, обработка или потребление ассоциированных документов не представляет собой большую проблему. Для тех, кто не имеет соответствующего программного обеспечения, обычно невозможна сборка, получение, обработка или потребление ассоциированных документов.

На фоне этого существует актуальная потребность в универсальности, насколько это касается получения и потребления документов.

Сущность изобретения

Описываются способы и системы модульной инфраструктуры содержимого и формата документов. Описываемая инфраструктура и формат определяют набор компоновочных блоков для составления, пакетирования, распространения и визуализации документно-ориентированного содержимого. Эти компоновочные блоки определяют платформно-независимую инфраструктуру для форматов документов, которые дают возможность программным и аппаратным системам генерировать, обмениваться и отображать документы наделым и единообразным образом. Упомянутые инфраструктура и формат были разработаны в гибком и расширяемом виде.

В дополнение к этим общим инфраструктуре и формату определен конкретный формат, известный как формат *расширенного пакета*, использующий общую инфраструктуру. Формат *расширенного пакета* представляет собой формат для хранения разбитых на страницы документов. Содержимое расширенного пакета может отображаться или печататься с высокой точностью воспроизведения на устройствах и в приложениях в широком спектре сред и в широком спектре сценариев.

Перечень фигур

Фиг.1 - блок-схема компонентов примерной инфраструктуры и формата согласно одному варианту выполнения.

Фиг.2 - блок-схема примерного пакета, хранящего документ, содержащий ряд частей, согласно одному варианту выполнения.

Фиг.3 - блок-схема, которая изображает примерного составителя, который создает пакет, и считывателя, который считывает пакет, согласно одному варианту выполнения.

Фиг.4 - примерная часть, которая связывает вместе три отдельные страницы.

Фиг.5 - диаграмма, которая изображает примерный селектор и последовательности, предназначенные для создания финансового отчета, содержащего как английское представление, так и французское представление отчета, согласно одному варианту выполнения.

Фиг.6 - некоторые примеры составителей и считывателей, работающих вместе и передающих данные о пакете, согласно одному варианту выполнения.

Фиг.7 - пример чередования многочисленных частей документа.

Фиг.8 и 9 - различные примеры пакетирования многочисленных частей документа, показанных на фиг.7.

Фиг.10 - примерный расширенный пакет и каждый из допустимых типов частей, которые могут составлять пакет или могут обнаруживаться в пакете, согласно одному варианту выполнения.

Фиг.11 - примерное отображение понятий общеязыковой среды выполнения в расширяемый язык разметки (XML) согласно одному варианту выполнения.

Фиг.12 - вертикальные метрики глифа и боковые метрики глифа согласно одному варианту выполнения.

Фиг.13 - отображение кластера “один в один” согласно одному варианту выполнения.

Фиг.14 - отображение кластера “многие в один” согласно одному варианту выполнения.

Фиг.15 - отображение кластера “один в многие” согласно одному варианту выполнения.

Фиг.16 - отображение кластера “многие в многие” согласно одному варианту выполнения.

Подробное описание предпочтительного варианта выполнения

Обзор

В настоящем документе описываются модульные инфрастрктура содержимого и формат документов. Инфраструктура и формат определяют набор стандартных компоновочных блоков для составления, пакетирования, распространения и визуализации документно-ориентированного содержимого. Эти стандартные компоновочные блоки определяют платформно-независимую инфраструктуру для форматов документов, которая дает возможность программным и аппаратным системам генерировать, обмениваться и отображать документы надежным и единообразным образом. Инфраструктура и формат были разработаны в гибком и расширяемом виде. В различных вариантах выполнения нет ограничения на тип содержимого, которое может быть включено, на то, как содержимое представляется, или на платформу, на которой должны создаваться клиенты для обработки содержимого.

В дополнение к этой общей инфраструктуре конкретный формат определяется с использованием общей инфраструктуры. Данный формат упоминается как формат *расширенного пакета* в настоящем документе и представляет собой формат для хранения разбиваемых на страницы или предварительно разбитых на страницы документов. Содержимое расширенного пакета может отображаться или печататься с высокой точностью воспроизведения на устройствах и в приложениях в широком спектре сред и в широком спектре сценариев.

Одной из целей описанной ниже инфраструктуры является обеспечение возможности взаимодействия независимо написанных программных средств и аппаратных систем, считывающих или записывающих содержимое, созданное в соответствии с описанными ниже инфраструктурой и форматом. Для достижения такой возможности взаимодействия описанный формат определяет формальные требования, которые должны выполнять системы, которые считывают или записывают содержимое.

Описание ниже организовано в соответствии с нижеследующим и представлено в двух главных разделах - один озаглавлен “Инфраструктура”, и другой озаглавлен

“Формат расширенного пакета”.

Раздел, озаглавленный “Инфраструктура”, представляет иллюстративную модель пакетирования и описывает различные части и отношения, которые составляют пакеты инфраструктуры. Описывается информация об использовании описательных метаданных в пакетах инфраструктуры, а также процесс отображения в физические контейнеры, расширения разметки инфраструктуры и использование механизмов управления версиями инфраструктуры.

В разделе, озаглавленном “Формат расширенного пакета”, исследуется структура одного конкретного типа построенного на инфраструктуре пакета, упоминаемого как *расширенный пакет*. В данном разделе также описываются части пакета, характерные для фиксированной полезной нагрузки, и определяется модель разметки расширенного пакета и модель рисования. Данный раздел завершается примерными элементами расширенной разметки и их свойствами вместе с изображенными образцами.

В качестве высокоуровневого обзора нижеследующего описания рассмотрим фиг.1, на которой в целом под позицией 100 изображены аспекты инфраструктуры и формата, обладающие признаками изобретения. Некоторые примерные компоненты инфраструктуры изображены под позицией 102, и некоторые компоненты формата расширенного пакета изображены под позицией 104.

Инфраструктура 102 содержит примерные компоненты, которые включают в себя без ограничения компонент отношения, компонент подключаемых контейнеров, компонент чередования/потоковой передачи данных и компонент управления версиями/расширяемости, каждый из которых подробно исследуется ниже. Формат 104 расширенного пакета содержит компоненты, которые включают в себя компонент селектора/задатчика последовательности и компонент определения разметки пакета.

В нижеследующем описании выполняется периодическая ссылка на фиг.1, так что читатель может поддерживать в перспективе, где описываемые компоненты размещаются в инфраструктуре и формате пакета.

Инфраструктура

Ниже предлагается описание общей инфраструктуры. Отдельные основные подзаголовки включают в себя “Модель пакета”, “Композиционные части: селектор и последовательность”, “Описательные метаданные”, “Физическая модель”, “Физические отображения” и “Управление версиями и расширяемость”. Каждый основной подзаголовок имеет один или несколько зависимых подзаголовков.

Модель пакета

В данном разделе описывается модель пакета, и он включает в себя подзаголовки, под которыми описываются пакеты и части, драйверы, отношения, отношения пакета и начальная часть.

Пакеты и части

В изображенной и описанной модели содержимое хранится в пакете. *Пакет* представляет собой логическую сущность, которая хранит коллекцию связанных *частей*. Назначением пакета является сбор всех частей документа (или других типов содержимого) в один объект, с которым легко работать программистам и конечным пользователям. Например, рассмотрим фиг.2, на которой изображен примерный пакет 200, хранящий документ, содержащий ряд частей, включающих в себя часть 202 разметки XML, представляющую документ, шрифтовую часть 204, описывающую шрифт, который используется в документе, ряд страничных частей 206, описывающих страницы документа, и часть с изображением, представляющую изображение в

документе. асть 202 разметки XML, которая представляет документ, полезна тем, что она может предоставить легкий поиск и ссылку без необходимости синтаксического анализа всего содержимого пакета. Это станет очевидным ниже.

По всему данному документу вводятся и обсуждаются понятия “считыватели” (также упоминаемые как потребители) и “составители” (также упоминаемые как создатели). Считыватель в том виде, в котором этот термин используется в данном документе, ссылается на сущность, которая считывает файлы или пакеты, основанные на модульном формате содержимого. Составитель в том виде, в котором этот термин используется в данном документе, ссылается на сущность, которая записывает файлы или пакеты, основанные на модульном формате содержимого. В качестве примера рассмотрим фиг.3, на которой показан составитель, который создает пакет, и считыватель, который считывает пакет. Обычно составитель и считыватель реализуются в виде программного обеспечения. По меньшей мере в одном варианте выполнения большая часть обработки служебных данных и сложностей, связанных с созданием и форматированием пакетов, возлагается на составителя. Это, в свою очередь, снимает большую часть сложности, связанной с обработкой, и служебных данных со считывателей, что, как понятно для специалиста в данной области техники, представляет собой отход от многих широко распространенных моделей. Данный аспект станет очевидным ниже.

Согласно по меньшей мере одному варианту выполнения один пакет содержит одно или несколько представлений содержимого, хранимого в пакете. Часто пакетом является один файл, упоминаемый в данной заявке как контейнер. Это дает конечным пользователям, например, удобный путь для распространения их документов со всеми компонентами документа (изображения, шрифты, данные и т.д.). Хотя пакеты часто соответствуют непосредственно одному файлу, это не всегда должно быть так. Пакет представляет собой логическую сущность, которая может быть представлена физически многочисленными способами (например, без ограничения, одним файлом, коллекцией свободных файлов, в базе данных, непродолжительной передачей по сетевому соединению и т.д.). Эти контейнеры хранят пакеты, но не все пакеты хранятся в контейнерах.

Абстрактная модель описывает пакеты независимо от любого механизма физического хранения. Например, абстрактная модель не ссылается на “файлы”, “потоки” или другие физические термины, относящиеся к физическому миру, в котором располагается пакет. Как описано ниже, абстрактная модель дает возможность пользователям создавать драйверы для различных физических форматов, протоколов передачи данных и т.п. По аналогии, когда приложение хочет напечатать изображение, оно использует абстракцию принтера (представленного драйвером, который понимает конкретный вид принтера). Таким образом, приложению не нужно знать о конкретном печатающем устройстве или то, как передать данные на печатающее устройство.

Контейнер обеспечивает многие преимущества по сравнению с тем, что, иначе, могло представлять собой совокупность свободных, несвязанных файлов. Например, аналогичные компоненты могут быть сгруппированы, и содержимое может быть индексировано и сжато. Кроме того, могут быть идентифицированы отношения между компонентами, и к компонентам может применяться управление правами, цифровые подписи, шифрование и метаданные. Конечно, контейнеры могут осуществлять другие признаки, которые конкретно не перечислены выше.

Общие свойства частей

В изображенном и описанном варианте выполнения часть содержит общие свойства (например, имя) и поток байтов. Она аналогична файлу в файловой системе или ресурсу на сервере протокола передачи гипертекста (HTTP). В дополнение к ее содержимому каждая часть имеет некоторые *общие свойства части*. Они включают в себя *имя,* которое представляет собой имя части, и *тип содержимого,* который представляет собой тип содержимого, хранимого в части. Части также могут иметь одно или несколько ассоциированных отношений, как описано ниже.

Имена частей используются всякий раз, когда необходимо сослаться некоторым образом на часть. В изображенном и описанном варианте выполнения имена организованы в иерархию, аналогичную путям в файловой системе или путям в универсальных идентификаторах информационного ресурса (URI). Ниже представлены примеры имен частей:

/document.xml

/tickets/ticket.xml

/images/march/summer.jpeg

/pages/page4.xml

Как видно выше, в данном варианте выполнения имена частей имеют следующие характеристики:

Имена частей подобны именам файлов в традиционной файловой системе.

Имена частей начинаются с косой черты ('/').

Подобно путям в файловой системе или путям в URI имена частей могут быть организованы в иерархию посредством набора каталогоподобных имен (tickets, images/march и pages в вышеприведенных примерах).

Эта иерархия состоит из сегментов, ограниченных косыми чертами.

Последний сегмент в имени аналогичен имени файла в традиционной файловой системе.

Важно отметить, что правила именования частей, особенно допустимые символы, которые могут использоваться для имен частей, характерны для инфраструктуры, описанной в данном документе. Эти правила именования частей основываются на правилах именования URI по стандарту Интернет. В соответствии с данным вариантом выполнения грамматика, используемая для задания имен частей в данном варианте выполнения, точно соответствует синтаксису abs_path, определенному в Разделе 3.3 (Path component) (*Компонент пути*) и 5 (Relative URI References) (*Ссылки по относительным URI*) документа RFC2396, спецификации (Uniform Resource Identifiers (URI: Generic Syntax)) ((*Универсальные идентификаторы ресурса (URI: Общий синтаксис)))*.

Следующие дополнительные ограничения применяются к abs_path в качестве допустимого имени части:

Компонент запроса, как он определен в Разделе 3 (URI Syntactic Components) ((Синтаксические компоненты URI) и 3.4 (Query Component)(Компонент запроса)), не применим к имени части.

Идентификатор фрагмента, как он определен в Разделе 4.1 (Fragment Identifier) ((Идентификатор фрагмента)), не применим к имени части.

Недопустимо иметь любую часть с именем, созданным добавлением * (сегмент “/”) к имени части существующей части.

Грамматика для имен частей показана ниже:

```
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
```

Сегменты имен всех частей в пакете, как можно видеть, образуют дерево. Это аналогично тому, что имеет место в файловых системах, в которых все неконцевые узлы на дереве представляют собой папки, а концевые узлы представляют собой фактические файлы, содержащие содержимое. Эти папкоподобные узлы (т.е. неконцевые узлы) на дереве имен служат аналогичной функции организации частей в пакете. Важно запомнить, однако, что эти “папки” существуют только как понятие в иерархии присваивания имен - они не имеют другого проявления в формате сохраняемости.

Имена частей не могут действовать на уровне “папок”. Конкретно, неконцевые узлы в иерархии (“папке”) именования частей не могут содержать часть и подпапку с этим же именем.

В изображенном и описанном варианте выполнения каждая часть имеет *тип содержимого*, который идентифицирует, какой тип содержимого хранится в части. Примеры типов содержимого включают в себя:

image/jpeg

text/xml

text/plain; charset="us-ascii"

Типы содержимого используются в изображенной инфраструктуре, как определено в RFC2045 (Multipurpose Internet Mail Extensions; (MIME)). (Многоцелевые расширения почтовой службы в Интернет (MIME)). Конкретно, каждый тип содержимого включает в себя тип полезных данных (например, текст), подтип (например, открытый) и необязательный набор параметров в форме ключ=значение (например, charset="us-ascii"); многочисленные параметры разделены точкой с запятой.

Адресация части

Часто части содержат ссылки на другие части. В качестве простого примера представим контейнер с двумя частями: файл разметки и изображение. Файлу разметки потребуется сохранить ссылку на изображение, так чтобы при обработке файла разметки ассоциированное изображение можно было идентифицировать и локализовать. Проектировщики типов содержимого и XML-схем могут использовать URI для представления этих ссылок. Чтобы сделать это возможным, необходимо определить отображение между “миром” имен частей и “миром” URI.

Чтобы предоставить возможность использовать URI в пакете, должно использоваться специальное правило интерпретации URI для оценки URI в основанном на пакете содержимом: сам пакет должен рассматриваться как

“авторитетный источник” для ссылок URI, и компонент пути URI используется для навигации по иерархии имен частей в пакете.

Например, при URI пакета http://www.example.com/foo/something.package ссылка на /abc/bar.xml интерпретируется, означая, что часть, названная /abc/bar.xml, не является URI http://www.example.com/abc/bar.xml.

Относительные URI должны использоваться, когда необходимо иметь ссылку с одной части на другую в контейнере. Использование относительных ссылок дает возможность совместного перемещения содержимого контейнера в другой контейнер (или в контейнер, например, из файловой системы) без модифицирования ссылок между частями.

Относительные ссылки из части интерпретируются относительно “базового URI” части, содержащей ссылку. По умолчанию, базовый URI части представляет собой имя части.

Рассмотрим контейнер, который включает в себя части со следующими именами:

/markup/page.xml

/images/picture.jpeg

/images/other_picture.jpeg

Если часть “/markup/page.xml” содержит ссылку URI на “../images/picture.jpeg”, то тогда эта ссылка должна интерпретироваться как ссылающаяся на имя части “/images/picture.jpeg”, согласно вышеприведенным правилам.

Некоторые типы содержимого обеспечивают возможность переопределения базового URI по умолчанию посредством задания другой базы в содержимом. При присутствии одного из этих переопределений явно заданный базовый URI должен использоваться вместо URI по умолчанию.

Иногда полезно “адресовать” долю или конкретную точку в части. В “мире” URI используется идентификатор фрагмента (см., например, RFC2396). В контейнере механизм работает таким же образом. Конкретно, фрагментом является строка, которая содержит дополнительную информацию, которая понимается в контексте типа содержимого адресуемой части. Например, в видеофайле фрагмент может идентифицировать кадр, в XML-файле он может идентифицировать долю XML-файла при помощи XPath.

Идентификатор фрагмента используется совместно с URI, который адресует часть для идентификации фрагментов адресуемой части. Идентификатор фрагмента является необязательным и отделяется от URI знаком перекрестной штриховки (“#”). Как таковой, он не является частью URI, но часто используется совместно с URI.

Нижеследующее описание предоставляет некоторые рекомендации для именования частей, так как модель именования пакета и части является довольно гибкой. Данная гибкость допускает широкий спектр применений пакета инфраструктуры. Однако важно понимать, что инфраструктура разработана для того, чтобы сделать возможным сценарии, в которых многочисленные несвязанные программные системы могут манипулировать “своими собственными” частями пакета без конфликтования друг с другом. Чтобы сделать это возможным, предлагаются некоторые рекомендации, которые, если им следовать, сделают это возможным.

Приведенные здесь рекомендации описывают механизм для минимизирования или по меньшей мере уменьшения количества конфликтов при именовании частей и борьбы с ними, когда они действительно возникают. Составители, создающие части в пакете, должны принимать меры для обнаружения и обработки конфликтов при именовании с существующими частями в пакете. В случае возникновения конфликта

имен составители могут не вслепую заменять существующие части.

В тех ситуациях, где гарантируется, что пакетом манипулирует один составитель, этот составитель может отходить от этих рекомендаций. Однако если существует возможность того, что многочисленные независимые составители совместно используют пакет, то все составители должны следовать этим рекомендациям. Рекомендуется, однако, чтобы все составители следовали этим рекомендациям в любом случае.

Требуется, чтобы составители, добавляющие части в существующий контейнер, делали это в новой "папке" иерархии присваивания имен, а не помещали части непосредственно в корень или в уже существующую папку. Таким образом, возможность конфликтов имен ограничивается первым сегментом имени части. Частям, созданным в этой новой папке, можно присваивать имена без риска конфликта с существующими частями.

В случае, если "предпочтительное" имя для папки уже используется существующей частью, составитель должен применить некоторую стратегию для выбора альтернативных имен папок. Составители должны использовать стратегию присоединения цифр к предпочтительному имени до тех пор, пока не будет обнаружено доступное имя папки (возможно прибегая к глобально уникальному идентификатору (GUID) после некоторого числа неуспешных повторений).

Одним результатом такой политики является то, что считыватели не должны предпринимать попытку локализации части при помощи "магического" или "общеизвестного" имени части. Вместо этого, составители должны создавать отношение пакета по меньшей мере с одной частью в каждой папке, которую они создают. Считыватели должны использовать эти отношения пакета для локализации частей, а не полагаясь на общеизвестные имена.

Если считыватель обнаружил по меньшей мере одну часть в папке (при помощи одного из вышеприведенных отношений пакета), он может использовать соглашения об общеизвестных именах частей в этой папке, чтобы найти другие части.

Драйверы

Описанный в данной заявке формат файлов может использоваться различными приложениями, различными типами документов и т.д., многие из которых имеют конфликтующие использования, конфликтующие форматы и т.п. Один или несколько *драйверов* используются для разрешения разнообразных конфликтов, таких как различия в форматах файлов, различия в протоколах передачи данных и т.п. Например, различные форматы файлов включают в себя свободные файлы и составные файлы, и различные протоколы передачи данных включают в себя HTTP, сетевые и беспроводные протоколы. Группа драйверов реализует абстрактное представление различных форматов файлов и протоколов передачи данных в единую модель. Могут быть предусмотрены многочисленные драйверы для различных сценариев, различных требований потребителей, различных физических конфигураций и т.д.

Отношения

Части в пакете могут содержать ссылки на другие части в этом пакете. Вообще, однако, эти ссылки представлены внутри ссылающейся части так, что они являются характерными для типа содержимого части; т.е. при произвольной разметке или при характерном для применения кодировании. Они эффективно скрывают внутреннее связывание между частями от считывателей, которые не понимают типы содержимого частей, содержащих такие ссылки.

Даже для общих типов содержимого (таких как разметка Фиксированной полезной нагрузки в разделе Расширенный пакет) считывателю потребуется провести синтаксический анализ всего содержимого в части, чтобы обнаружить и разрешить ссылки на другие части. Например, при реализации печатающей системы, которая печатает документы по одной странице за раз, может быть желательным идентифицировать изображения и шрифты, содержащиеся на конкретной странице. Существующие системы должны проводить синтаксический анализ всей информации для каждой страницы, что может отнимать много времени, и должы понимать язык каждой страницы, что может быть не так с некоторыми устройствами или считывателями (например, которые выполняют промежуточную обработку в отношении документа, когда он проходит по конвейеру процессоров на пути к устройству). Вместо этого, системы и способы, описанные в данной заявке, используют отношения для идентификации отношений между частями и описания природы этих отношений. Язык отношений простой и определяется один раз, так что считыватели могут понимать отношения без необходимости знания многочисленных различных языков. В одном варианте выполнения отношения представляются в XML как индивидуальные части. Каждая часть имеет ассоциированную часть отношения, которая содержит отношения, для которых часть является источником.

Например, приложение электронных таблиц использует этот формат и запоминает различные электронные таблицы в виде частей. Приложение, которое ничего не знает о языке электронных таблиц, все же может обнаружить различные отношения, ассоциированные с электронными таблицами. Например, приложение может обнаружить изображения в электронных таблицах и метаданные, ассоциированные с электронными таблицами. Примерная схема отношений приведена ниже:

```
<?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>
```

Данная схема определяет два XML-элемента, один назван “отношения” (“Relationships”) и другой назван “отношение” (“Relationship”). Этот элемент “отношение” используется для описания единственного отношения, как описано в данной заявке, и имеет следующие атрибуты: (1) “target” (цель), который указывает часть, к которой относится часть источника, (2) “name” (имя), который указывает тип или природу отношения. Элемент “отношения” определяется так, чтобы он имел возможность хранить ноль или более элементов “отношение” и служит просто для сбора этих элементов “отношение” вместе в блок.

Системы и способы, описанные в данной заявке, вводят высокоуровневый механизм для разрешения таких проблем, называемых “отношениями”. Отношения

обеспечивают дополнительный путь для представления вида связи между частью источника и частью цели в пакете. Отношения делают связи между частями непосредственно “обнаруживаемыми” без просмотра содержимого частей, так что они являются независимыми от характерной для содержимого схемы и более быстрыми для разрешения. Дополнительно, эти отношения являются независимыми от протокола. Множество различных отношений могут быть ассоциированы с конкретной частью.

Отношения обеспечивают вторую важную функцию: возможность связывания частей без их модификации. Ингда эта информация служит в качестве формы “аннотации”, где тип содержимого “аннотированной” части не определяет путь присоединения заданной информации. Потенциальные примеры включают в себя присоединенные описательные метаданные, этикетки печати и действительные аннотации. Наконец, некоторые сценарии требуют информации для присоединения к существующей части специально без модифицирования этой части, например, когда часть зашифрована и не может быть расшифрована, или когда часть подписана цифровой подписью, и ее изменение делало бы подпись недействительной. В другом примере пользователь может захотеть присоединить аннотацию к файлу с изображением в формате JPEG (формат объединенной группы экспертов в области фотографии). Формат изображения JPEG в настоящее время не обеспечивает поддержки для идентификации аннотаций. Изменение формата JPEG для выполнения этого желания пользователя не является целесообразным. Однако системы и способы, описанные в данной заявке, позволяют пользователю обеспечить аннотацию к файлу JPEG без модифицирования формата изображения JPEG.

В одном варианте выполнения отношения представляются с использованием XML в частях отношения. Каждая часть в контейнере, которая является источником одного или нескольких отношений, имеет ассоциированную часть отношения. Эта часть отношения хранит (выраженный в XML с использованием типа содержимого *application/PLACEHOLDER*) список отношений для этой части источника.

На фиг.4 внизу показана среда 400, в которой часть 402 “корешка” (аналогично FixedPanel) связывает вместе три страницы 406, 408 и 410. Набор страниц, связанных вместе корешком, имеет ассоциированную “этикетку 404 печати”. Дополнительно, страница 2 имеет свою собственную этикетку 412 печати. Связи от части 402 корешка к ее этикетке 404 печати и от страницы 2 к ее этикетке 412 печати представлены с использованием отношений. В компоновке по фиг.4 часть 402 корешка имеет ассоциированную часть отношения, которая содержит отношение, которое соединяет корешок с ticket1, как показано в примере ниже.

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

Отношения представлены с использованием элементов <Relationship>, вложенных в единственный элемент <Relationships>. Эти элементы определены в пространстве имен http://mmcfrels (PLACEHOLDER). Смотрите вышеприведенную примерную схему и относящееся к ней описание для примерных отношений.

Элемент отношения имеет следующие дополнительные атрибуты:

Атрибут	Требуется	Значение

Target (цель)	Да	URI, который указывает на часть на другом конце отношения. Относительные URI ДОЛЖНЫ интерпретироваться относительно части источника.
Name (имя)	Да	Абсолютный URI, который уникально определяет роль или назначение отношения.

Атрибут Name не является обязательно действительным адресом. Различные типы отношений идентифицируются при помощи их Name. Эти имена определяются аналогично тому, как пространства имен определяются для пространств имен XML. Конкретно, посредством использования имен, сформированных по пространству имен домена Интернета, некоординирующие части могут безопасно создавать неконфликтующие имена отношений, как они могут для пространств имен XML.

Части отношений не разрешается участвовать в других отношениях. Однако она является первоклассной частью во всех других смыслах (например, она является адресуемой по URI, она может быть открыта, считана, удалена и т.д.). Отношения обычно не указывают на предметы вне пакета. URI, используемые для идентификации целей отношения, не включают в себя, в основном, схему URI.

Часть и ассоциированная с ней часть отношения связаны соглашением об именах. В данном примере часть отношения для корешка будет храниться в /content/_rels/spine.xml.rels, и отношения для страницы 2 будут храниться в /content/_rels/p2.xml.rels. Заметьте, что здесь используются два специальных соглашения об именах. Во-первых, часть отношения для некоторой (другой) части в заданной “папке” в иерархии имен хранится в “подпапке”, названной _rels (для идентификации отношений). Во-вторых, имя этой хранящей отношение части образовано посредством присоединения расширения .rels к имени исходной части. В конкретных вариантах выполнения части отношений являются типа содержимого application/xml+relationshipsPLACEHOLDER.

Отношение представляет направленную связь между двумя частями. Вследствие способа, которым представлено отношение, можно эффективно прослеживать отношения от их частей источника (так как тривиально найти часть отношения для любой заданной части). Однако не является эффективным прослеживать отношения обратно от цели отношения (так как способом поиска всех отношений с частью является просмотр всех отношений в контейнере).

Чтобы выполнить обратное прослеживание возможного отношния, используется новое отношение для представления другого (прослеживаемого) направления. Это представляет собой моделирующую методику, которую может использовать проектировщик типа отношения. Следуя вышеприведенному примеру, если было бы важным иметь возможность найти корешок, который имеет прикрепленную этикетку ticket1, то использовалось бы второе отношение, связывающее этикетку с корешком, такое как:

В *content/_rels/p1.xml.rels*:

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

Отношения пакета

“Отношения пакета” используются для нахождения общеизвестных частей в пакете. Этот способ исключает зависимость от соглашений об именах для нахождения частей

в пакете и гарантирует, что не будет конфликтов между идентичными именами частей в различных полезных нагрузках.

Отношения пакетов представляют собой специальные отношения, целью которых является часть, но источником которых часть не является: источник представляет собой пакет в целом. Обладание “общеизвестной” частью действительно представляет собой обладание “общеизвестным” именем отношения, которое помогает находить эту часть. Это работает, так как существует четко определенный механизм, позволяющий присваивать имена отношениям некоординирующими частями, тогда как некоторые варианты выполнения не содержат такого механизма для имени части - эти варианты выполнения ограничиваются набором рекомендаций. Отношения пакета обнаруживаются в части отношений пакета, и им присваиваются имена с использованием стандартных соглашений об именах для частей отношения. Таким образом: ей присваивается имя “/_rels/.rels”.

Отношения в этой части отношений пакета полезны при нахождении общеизвестных частей.

Начальная часть

Одним примером общеизвестной части уровня пакета является “начальная” часть пакета. Это часть, которая обычно обрабатывается при открытии пакета. Она представляет логический корень содержимого документа, хранимого в пакете. Начальная часть пакета локализуется в результате следования общеизвестному отношению пакета. В одном примере это отношение имеет следующее имя: http://mmcf-start-part-PLACEHOLDER.

Композиционные части: селектор и последовательность

Описанная инфраструктура определяет два механизма для построения высокоуровневых структур из частей: *селекторов* и *последовательностей*.

Селектор представляет собой часть, которая “осуществляет выбор” из некоторого количества других частей. Например, часть селектора может “выбрать” из части, представляющей английскую версию документа, и части, представляющей французскую версию документа. Последовательность представляет собой часть, которая “устанавливает последовательность” из некоторого количества других частей. Например, часть последовательности может комбинировать (в линейную последовательность) две части, одна из которых представляет пятистраничный документ, а другая из которых представляет десятистраничный документ.

Эти два типа композиционных частей (последовательность и селектор) и правила для их сборки составляют *композиционную модель*. Композиционные части могут составлять другие композиционные части, таким образом, можно иметь, например, селектор, который выбирает из двух композиций. В качестве примера рассмотрим фиг.5, на которой показан пример финансового отчета, содержащего как английское представление, так и французское представление. Каждое из этих представлений дополнительно состоит из введения (титульного листа), за которым следуют финансовые документы (электронная таблица). В данном примере селектор 500 выбирает между английским и французским представлением отчета. Если выбрано английское представление, последовательность 502 устанавливает последовательность английской вводной части 506 с английской финансовой частью 508. Альтернативно, если выбрано французское представление, последовательность 504 устанавливает последовательность французской вводной части 510 с французской финансовой частью 512.

XML композиционной части

В изображенном и описанном варианте выполнения композиционные части описываются с использованием небольшого количества XML-элементов, причем все они выведены из общего пространства имен композиции. В качестве примера рассмотрим следующее:

Элемент: <selection>
Атрибуты: Нет
Разрешенные дочерние элементы: <item>
Элемент: <sequence>
Атрибуты: Нет
Разрешенные дочерние элементы: <item>
Элемент: <item>
Атрибуты: Target - имя части для части в композиции

В качестве примера приведен XML для примера по фиг.5 выше:

MainDocument. XML

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

EnglishRollup. XML

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

В этом XML MainDocument.xml представляет всю часть в пакете и указывает благодаря тегу (неотображаемому элементу разметки) “selection”, что выбор должен выполняться между различными предметами, инкапсулированными тегом “item”, т.е. “EnglishRollup.xml” и “FrenchRollup.xml”.

EnglishRollup.xml и FrenchRollup.xml представляют собой благодаря тегам “sequence” последовательности, которые устанавливают совместную последовательность соответствующих предметов, инкапсулированных их соответствующими тегами “item”.

Таким образом, обеспечивается простая грамматика XML для описания селекторов и последовательностей. Каждая часть в этом композиционном блоке создается и выполняет одну операцию - или выбор, или установление последовательности. Посредством использования иерархии частей могут быть построены различные надежные коллекции выборов и последовательностей.

Композиционный блок

Композиционный блок пакета содержит набор всех композиционных частей (селектор или последовательность), которые являются достижимыми из начальной части пакета. Если начальная часть пакета не является ни селектором, ни последовательностью, то тогда композиционный блок считается пустым. Если начальная часть представляет собой композиционную часть, то тогда дочерние <item> в этих композиционных частях рекурсивно прослеживаются для получения ориентированного ациклического графа композиционных частей (останавливая прослеживание, когда встречается некомпозиционная часть). Этот граф представляет собой композиционный блок (и он должен в соответствии с данным вариантом выполнения быть ациклическим для пакета, чтобы он был допустимым).

Определение семантики композиции

Установив выше относительно простую грамматику XML, последующее описание описывает то, как представить информацию, так чтобы выборы можно было

осуществить, основываясь на типе содержимого. Т.е. описанный выше XML предоставляет достаточно информации, позволяющей считывателям локализовать части, которые собраны вместе в композицию, но не предоставляет достаточной информации, чтобы помочь считывателю узнать больше о природе композиции. Например, при наличии выбора, который составляет две части, как считыватель может знать, на каком основании сделать выбор (например, языка, размера бумаги и т.д.)? Ответом является то, что эти правила ассоциированы с *типом содержимого* композиционной части. Таким образом, часть селектора, которая используется для отбора из представлений, основанных на языке, будет иметь отличающийся ассоциированный тип содержимого из части селектора, которая отбирает из представлений на основе размеров бумаги.

Общая инфраструктура определяет общую форму для этих типов содержимого:

Application/XML+Selector-SOMETHING

Application/XML+Sequence-SOMETHING

SOMETHING в этих типах содержимого заменяется словом, которое указывает природу выбора или последовательности, например размер бумаги, цвет, язык, резидентное программное обеспечение на устройстве считывателя и т.п. В этой инфраструктуре тогда можно придумывать всевозможные виды селекторов и последовательностей, и каждый может иметь очень разнообразную семантику.

Описываемая инфраструктура также определяет следующие общеизвестные типы содержимого для селекторов и последовательностей, которые все считыватели или считывающие устройства должны понимать.

Тип содержимого	Правила
Application/XML + Selector+SupportedContentType	Отбирает из предметов, основываясь на их типах содержимого. Выбирает первый предмет, для которого доступно программное обеспечение, которое понимает данный тип содержимого.

В качестве примера рассмотрим следующее. Предположим, что пакет содержит документ, который имеет страницу, и в центре страницы находится область, в которой должно появиться видео. В данном примере часть с видеостраницы может содержать видео в виде формата видео QuickTime. Одной проблемой с таким сценарием является то, что видео в формате QuickTime не везде понимается. Предположим, однако, что в соответствии с данной инфраструктурой и, в частности, описанным ниже форматом расширенного пакета существует везде понимаемый формат изображения - JPEG. При создании пакета, который содержит описанный выше документ, создатель может, в дополнение к определению видео в качестве части пакета, определить изображение в формате JPEG для страницы и вставить селектор SupportedContentType, так что если компьютер пользователя имеет программное обеспечение, которое понимает видео в формате QuickTime, то выбирается видео в формате QuickTime, иначе выбирается изображение в формате JPEG.

Таким образом, как описано выше, компоненты селектора и последовательности уровня инфраструктуры позволяют создать надежную иерархию, которая в этом примере определена в XML. Кроме того, существует четко определенный путь идентификации характера изменения селекторов и последовательностей с использованием типов содержимого. Кроме того, согласно одному варианту выполнения общая инфраструктура содержит один конкретный тип содержимого, который определяется предварительно и который позволяет производить обработку и использование пакетов, основываясь на том, что потребитель (например, считыватель или считывающее устройство) действительно понимает и не понимает.

Другие типы содержимого композиционной части могут быть определены с использованием подобных правил, примеры которых описаны ниже.

Описательные метаданные

Согласно одному варианту выполнения части с описательными метаданными предоставляют составителям или создателям пакетов путь для хранения значений свойств, которые дают возможность считывателям пакетов надежно обнаруживать значения. Эти свойства обычно используются для записи дополнительной информации о пакете в целом, а также отдельных частях в контейнере. Например, часть с описательными метаданными в пакете может хранить информацию, такую как автор пакета, ключевые слова, краткое изложение и т.п.

В изображенном и описанном варианте выполнения описательные метаданные выражаются в XML, хранятся в частях с общеизвестными типами содержимого и могут быть обнаружены с использованием общеизвестных типов отношений.

Описательные метаданные хранят *свойства метаданных*. Свойства метаданных представлены именем свойств и одним или многими значениями свойств. Значения свойств имеют простые типы данных, таким образом каждый тип данных описывается одним qname XML. Тот факт, что свойства описательных метаданных имеют простые типы, не означает, что нельзя хранить данные со сложными типами XML в пакете. В этом случае необходимо хранить информацию в виде полной части XML. Когда делается так, то снимаются все ограничения на использование только простых типов, но теряется простота “плоской” модели свойств описательных метаданных.

В дополнение к общему механизму назначения для определения наборов свойств существует характерный, четко определенный набор базовых *свойств документа*, хранимый с использованием данного механизма. Эти базовые свойства документа обычно используются для описания документов и включают в себя свойства, такие как заголовок, ключевые слова, автор и т.д.

Наконец, части с метаданными, хранящие эти базовые свойства документа, также могут хранить дополнительные специализированные свойства в дополнение к базовым свойствам документа.

Формат метаданных

Согласно одному варианту выполнения части с описательными метаданными имеют тип содержимого и цель, определяемую отношениями согласно следующим правилам:

Правила обнаружения описательных метаданных	Используя специализированные свойства	Используя базовые свойства документа
Тип содержимого части с описательными метаданными ДОЛЖЕН быть:	application/xml-SimpleTypeProperties-PLACEHOLDER	
Тип содержимого части источника, которая может иметь отношение, целью которого является часть с описательными метаданными, может быть:	Любой	Любой
Имя отношения, целью которого является часть с описательными метаданными, может быть одно из двух:	*специализированное пространство имен Uri*	http://mmcf-DocumentCore-PLACEHOLDER
Количество частей с описательными метаданными, которые могут быть присоединены к части источника, может быть:	UNBOUNDED (не ограничено)	0 или 1
Количество частей источника, которые могут иметь такую же прикрепленную часть с описательными метаданными, ДОЛЖНО быть	UNBOUNDED	UNBOUNDED

Следующий шаблон XML используется для представления описательных метаданных согласно одному варианту выполнения. Подробности каждого

компонента разметки приведены в таблице после образца.

```
<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>
```

Компонент разметки	Описание
xmlns="http://mmcf-common-services/PLACEHOLDER"	Определяет общее пространство имен служб организации по выработке стандартов в области мультимедиа (MMCF)
xmlns:xsd="http://www.w3.org/2001/XMLSchema"	Определяет пространство имен XML-схемы. Многие специализированные свойства и большая часть базовых свойств документа будут иметь встроенные типы данных, определяемые с использованием определения XML-схемы (XSD). Хотя каждое свойство может иметь свое собственное пространство имен, пространство имен XSD размещается в корне XML описательных метаданных.
mcs:properties	Корневой элемент XML описательных метаданных
mcs:property	Элемент свойств. Элемент свойств хранит свойство qname и значение. Может быть неограниченное количество элементов свойств. Элементы свойств рассматриваются непосредственными дочерними элементами корневого элемента.
xmlns:prns	Пространство имен свойств: для базовых свойств документа им является http://mmcf-DocumentCore-PLACEHOLDER. Для специализированных свойств им будет специализированное пространство имен.
prns:name	Имя свойств: строковый атрибут, который хранит имя свойств.
mcs:type="*datatype*"	Тип представляет собой строковый атрибут, который хранит определение типа данных свойств, например xsd:string.
mcs:value	Данный компонент задает значение свойства. Элементы значения представляют собой непосредственные дочерние элементы элементов свойств. Если mcs: multivalued="true", то тогда может быть неограниченное количество элементов значений

Базовые свойства документа

Ниже приведена таблица базовых свойств документа, которые включают в себя имя свойства, тип свойства и описание.

Имя	Тип		Описание
Comments	String, необязательный, однозначный		Комментарий к документу в целом, который включает автор (Author). Им может быть краткое содержание документа.
Copyright	String, необязательный, однозначный		Строка авторского права для данного документа.
EditingTime	Int64, необязательный, однозначный		Время, потраченное на редактирование данного документа в секундах. Устанавливается логическими средствами приложения. Данное значение должно иметь соответствующий тип.
IsCurrentVersion	Boolean, необязательный, однозначный		Указывает, является ли данный экземпляр текущей версией документа или устаревшей версией. Данное поле может быть извлечено из VersionHistory, но процесс извлечения может быть дорогостоящим.
Language	Keyword (=string256), необязательный, многозначный		Язык документа (английский, французский и т.д.). Данное поле устанавливается логическими средствами приложения.
RevisionNumber	String, необязательный, однозначный		Редакция документа.
Subtitle	String, необязательный, однозначный		Дополнительный или объяснительный заголовок документа.
TextDataProperties	TextDataProperties, необязательный, однозначный		Если данный документ имеет текст, то данное свойство определяет коллекцию свойств текста документа, таких как число абзацев, число строк и т.д.
	CharacterCount	int64	
	LineCount	int64	
	PageCount	int64	
	ParagraphCount	int64	
	WordCount	int64	

TimeLastPrinted	datetime, необязательный, однозначный	Дата и время, когда данный документ был последний раз напечатан.
Title	String, необязательный, однозначный	Заголовок документа, понимаемый приложением, которое обрабатывает документ. Он отличается от имени файла, который содержит пакет.
TitleSortOrder	String необязательный, однозначный	Порядок сортировки заголовка (например, “The Beatles” будет иметь SortOrder “Beatles” без находящегося впереди “The”).
ContentType	Keyword (= string256), необязательный, многозначный	Тип документа, установленный логическими средствами приложения. Тип, который здесь хранится, должен быть распознанным “типом MIME”. Данное свойство может быть полезным для категоризации или поиска документов некоторых типов.

Физическая модель

Физическая модель определяет различные пути, которыми пакет используется составителями и считывателями. Данная модель основывается на трех компонентах: *составителе, считывателе* и *программном канале* между ними. На фиг.6 показаны некоторые примеры составителей и считывателей, работающих вместе и передающих данные о пакете.

Программный канал переносит данные от составителя к считывателю. Во многих сценариях программный канал может просто содержать вызовы интерфейса прикладного программирования (ИПП, API), которые считыватель выполняет для считывания пакета из локальной файловой системы. Это упоминается как *прямой доступ*.

Часто, однако, считыватель и составитель должны передавать данные друг другу по протоколу некоторого типа. Эта передача данных происходит, например, через границу процесса или между сервером и настольным компьютером. Это упоминается как *сетевой доступ* и является важным из-за характеристик передачи данных программного канала (особенно быстродействие и задержка запроса).

Чтобы получить максимальные рабочие характеристики, разработки физического пакета должны принимать во внимание поддержку в трех важных областях: *стиль доступа*, *стиль компоновки* и *стиль передачи данных*.

Стиль доступа

Потребление потока

Так как передача данных между составителем и считывателем, использующими сетевой доступ, не является мгновенной, важно предоставить возможность последовательного создания и потребления пакетов. В частности, рекомендуется, в соответствии с данным вариантом выполнения, чтобы любой физический формат пакета был разработан так, чтобы обеспечить возможность считывателю начинать интерпретирование и обработку данных, когда он принимает данные (например, части), до того как все биты пакета будут доставлены по программному каналу. Эта возможность называется *потреблением потока*.

Создание потока

Когда составитель начинает создавать пакет, он не всегда знает, что он поместит в пакет. В качестве примера, когда приложение начинает создавать пакет файла спулинга печати, он может не знать, сколько страниц необходимо будет поместить в пакет. В качестве другого примера, программа на сервере, которая динамически генерирует отчет, может не понимать, какую продолжительность отчет будет иметь, или сколько изображений будет иметь отчет, пока он полностью не сгенерирует отчет. Чтобы принимать во внимание таких составителей, физические пакеты должны предоставлять составителям возможность динамически добавлять части, после того

как другие части уже были добавлены (например, составитель не должен устанавливать заранее, сколько частей он будет создавать, когда он начинает запись). Кроме того, физические пакеты должны давать возможность составителю начинать запись содержимого части без знания окончательной продолжительности этой части. Вместе данные требования дают возможность осуществлять *создание потока*.

Одновременное создание и потребление

В высоко конвейеризованной архитектуре создание потока и потребление потока может происходить одновременно для конкретного пакета. При конструировании физического пакета поддержка создания потока и поддержка потребления потока могут толкать конструирование в разных направлениях. Однако часто можно найти конструкцию, которая поддерживает обоих. Из-за преимуществ в конвейерной архитектуре рекомендуется, чтобы физические пакеты поддерживали *одновременное создание и потребление.*

Стили раскладки

Физические пакеты хранят коллекцию частей. Эти части могут компоноваться по одному из двух стилей: простой упорядоченности и с чередованием. С простой упорядоченностью части в пакете компонуются с определенной упорядоченностью. Когда такой пакет доставляется простым линейным образом, начиная с первого байта в пакете до последнего, все байты первой части поступают первыми, затем все байты второй части и т.д.

При компоновке с чередованием байты многочисленных частей чередуются, позволяя получить улучшенные характеристики в некоторых сценариях. Двумя сценариями, которые имеют существенные преимущества от чередования, являются воспроизведение мультимедиа (например, одновременная доставка видео и аудио) и ссылка на встроенные ресурсы (например, ссылка в середине файла разметки на изображение).

Чередование обрабатывается по специальному соглашению для организации содержимого чередуемых частей. Посредством разбиения частей на порции и чередования этих порций можно достичь требуемых результатов чередования, в то же самое время все же делая возможным легкое восстановление исходной большей части. Чтобы понять, как работает чередование, на фиг.7 изображается простой пример, включающий в себя две части: content.xml 702 и image.jpeg 704. Первая часть content.xml описывает содержимое страницы, и в середине этой страницы находится ссылка на изображение (image.jpeg), которое должно появиться на странице.

Чтобы понять, почему чередование является полезным, рассмотрим, как эти части были бы расположены в пакете с использованием простой упорядоченности, как показано на фиг.8. Считыватель, который обрабатывает этот пакет (и последовательно принимает байты), не сможет отобразить изображение до тех пор, пока он не примет всю часть content.xml, а также image.jpeg. При некоторых обстоятельствах (например, малых или простых пакетах или быстродействующих каналах передачи данных) это может не быть проблемой. При других обстоятельствах (например, если content.xml был очень большим, или очень медленный канал передачи данных) необходимость считывания всей части content.xml для перехода к изображению будет приводить к недопустимым рабочим характеристикам или накладывать чрезмерные требования по памяти на систему считывателя.

Чтобы приблизиться к идеальным рабочим характеристикам, хорошо было бы иметь возможность разбить часть content.xml и вставить часть image.jpeg в середину,

сразу после того, откуда происходит ссылка на изображение. Это позволило бы считывателю раньше начать обработку изображения: как только он встречает ссылку, за которой следуют данные изображения. Это привело бы, например, к компоновке пакета, показанной на фиг.9. Из-за преимуществ в рабочих характеристиках часто желательно, чтобы физические пакеты поддерживали чередование. В зависимости от вида используемого физического пакета чередование может поддерживаться или может не поддерживаться. Различные физические пакеты могут обрабатывать внутреннее представление чередования по-разному. Независимо от того, как физический пакет обрабатывает чередование, важно запомнить, что чередование представляет собой оптимизацию, которая происходит на физическом уровне, и часть, которая разбивается на многочисленные порции в физическом файле, все же представляет собой одну логическую часть; сами порции не являются частями.

Стили передачи данных

Передача данных между составителем и считывателем может быть основана на *последовательной доставке* частей или посредством *произвольного доступа* к частям, предоставляя возможность доступа к ним вне упорядоченности. Какой из этих стилей передачи данных используется зависит от возможностей как программного канала, так и формата физического пакета. В общих чертах, все программные каналы поддерживают последовательную доставку. Физические пакеты должны поддерживать последовательную доставку. Чтобы поддерживать сценарии произвольного доступа, как используемый программный канал, так и физический пакет должны поддерживать произвольный доступ. Некоторые программные каналы основываются на протоколах, которые могут делать возможным произвольный доступ (например, HTTP 1.1 с поддержкой байтового диапазона). Чтобы сделать возможным максимальные рабочие характеристики, когда используются такие программные каналы, рекомендуется, чтобы физические пакеты поддерживали произвольный доступ. При отсутствии такой поддержки считыватели будут просто ожидать до тех пор, пока не будут последовательно доставлены части, которые им нужны.

Физические отображения

Логическая модель пакетирования определяет абстракцию пакета; фактический экземпляр пакета основывается на некотором конкретном физическом представлении пакета. Модель пакетирования может отображаться в физические форматы сохраняемости, а также на различные средства транспортировки (например, основанные на сети протоколы). Физический формат пакета может описываться как отображение из компонентов абстрактной модели пакетирования в признаки конкретного физического формата. Модель пакетирования не задает, какие физические форматы пакета должны использоваться для архивирования, распространения или спулинга пакетов. В одном варианте выполнения задается только логическая структура. Пакет может быть “физически” реализован посредством коллекции свободных файлов, архива файлов .ZIP, составного файла или некоторого другого формата. Выбранный формат поддерживается целевым устройством потребления или драйвером для устройства.

Отображаемые компоненты

Каждый физический формат пакета определяет отображение для следующих компонентов. Некоторые компоненты являются необязательными, и конкретный физический формат пакета может не поддерживать эти необязательные компоненты.

	Компонент	Описание	Требуется или необязательный
Части	Имя	Именует часть	Требуется
	Тип содержимого	Идентифицирует вид содержимого, хранимого в части	Требуется
	Содержимое части	Хранит фактическое содержимое части	Требуется

Общие шаблоны отображения			
Стили доступа	Потребление потока	Дает возможность считывателям начать обработку частей до получения всего пакета	Необязательный
	Создание потока	Дает возможность составителям начать запись частей в пакет без знания заранее всех частей, которые будут записаны	Необязательный
	Одновременное создание и потребление	Позволяет созданию потока и потреблению потока происходить одновременно и в отношении одного и того же пакета	Необязательный
Стили компоновки	Простое упорядочение	Все байты для части N появляются в пакете до байтов для части N+1	Необязательный
	С чередованием	Байты для многочисленных частей чередуются	Необязательный
Стили передачи данных	Последовательная доставка	Вся часть N доставляется считывателю до части N+1	Необязательный
	Произвольный доступ	Считыватель может запросить доставку части вне последовательного упорядочения	Необязательный

Существуют многие физические форматы хранения, признаки которых частично соответствуют компонентам модели пакетирования. При определении отображений из модели пакетирования в такие форматы хранения может быть желательным использовать преимущества любых подобий в возможностях между моделью пакетирования и физической запоминающей средой, тогда как использование уровней отображения для обеспечения дополнительных возможностей по своей природе не присутствует в физической запоминающей среде. Например, некоторые физические форматы пакета могут хранить отдельные части в качестве отдельных файлов в файловой системе. В таком физическом формате было бы естественным отобразить многие имена частей непосредственно в имена идентичных физических файлов. Имена частей, использующие символы, которые представляют собой недопустимые имена файлов файловой системы, могут требовать некоторый вид механизма введения управляющих символов.

Во многих случаях проектировщики различных физических форматов пакета могут столкнуться с одной общей проблемой отображения. Два примера общих проблем отображения возникают при сопоставлении произвольных Типов содержимого с частями и при поддержке стиля компоновки с чередованием. Эта спецификация предлагает общие решения таких общих проблем отображения. Проектировщикам конкретных физических форматов пакета может быть рекомендовано, но это не является для них необходимым, использовать общие решения отображения, определенные в данной заявке.

Идентификация типов содержимого частей

Отображения физического формата пакета определяют механизм для хранения типа содержимого для каждой части. Некоторые физические форматы пакета имеют собственный механизм для представления типов содержимого (например, заголовок “Content-Type” в MIME). Для таких физических пакетов рекомендуется, чтобы отображение использовало собственный механизм для представления типов содержимого для частей. Для других физических форматов пакета используется некоторый другой механизм для представления типов содержимого. Рекомендуемым механизмом для представления типов содержимого в этих пакетах является механизм включения специально именованного XML-потока в пакет, известного как поток

типов. Этот поток не является частью, и поэтому сам по себе не является адресуемым по URI. Однако он может чередоваться в физическом пакете, используя такие же механизмы, что и используемые для частей с чередованием.

Поток типов содержит XML с элементом “Types” верхнего уровня, и один или несколько подчиненных элементов “Default” и “Override”. Элементы “Default” определяют отображения по умолчанию из расширений имени части в типы содержимого. Это имеет преимущества в том, что расширения файлов часто соответствуют типу содержимого. Элементы “Override” используются для задания типов содержимого на частях, которые не охватываются отображениями по умолчанию или не являются совместимыми с ними. Составители пакета могут использовать элементы “Default” для уменьшения количества элементов “Override” на часть, но им не требуется это делать.

Элемент “Default” имеет следующие атрибуты:

Имя	Описание	Требуется
Extension	Расширение имени части. Элемент “Default” совпадает с любой частью, имя которой оканчивается точкой, за которой следует это значение атрибута.	Да
ContentType	Тип содержимого, определенный в RFC2045. Указывает тип содержимого любых совпадающих частей (если только не переопределены элементом “Override”; см. ниже).	Да

Элемент “Override” имеет следующие атрибуты:

Имя	Описание	Требуется
PartName	URI имени части. Элемент “Override” совпадает с частью, имя которой равно этому значению атрибута.	Да
ContentType	Тип содержимого, определенный в RFC2045. Указывает тип содержимого совпадающей части.	Да

Нижеследующее представляет собой пример XML, содержащегося в потоке типов:

```
<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>
```

В нижеследующей таблице показан образец списка частей и их соответствующие типы содержимого, определенные вышеупомянутым потоком типов:

Имя части	Тип содержимого
/a/b/sample1.txt	plain/text
/a/b/sample2.jpeg	image/jpeg
/a/b/sample3.picture	image/gif
/a/b/sample4.picture	image/jpeg

Для каждой части в пакете поток типов содержит или (a) один совпадающий элемент “Default”, или (b) один совпадающий элемент “Override”, или (c) как совпадающий элемент “Default”, так *и* совпадающий элемент “Override” (в этом случае элемент “Override” имеет приоритет). В общем случае, имеется максимум один элемент “Default” для любого заданного расширения и один элемент “Override” для любого заданного имени части.

Не является важным порядок элементов “Default” и “Override” в потоке типов.

Однако в пакетах с чередованием элементы “Default” и “Override” появляются в физическом пакете *перед* частью (частями), которой они соответствуют.

Чередование

Не все физические пакеты поддерживают чередование потоков данных частей естественным образом. В одном варианте выполнения отображение в любой такой физический пакет использует общий механизм, описанный в этом разделе, позволяющий получить чередование частей. Общий механизм работает посредством разбиения потока данных части на многочисленные порции, которые затем могут чередоваться с порциями других частей или целых частей. Отдельные порции части существуют в физическом отображении и не являются адресуемыми в логической модели пакетирования. Порции могут иметь нулевой размер.

Определяется следующее уникальное отображение из имени части в имена для отдельных порций части, так что считыватель может сшить вместе порции в их исходном порядке, образуя поток данных части.

Грамматика для получения имен порций для заданного имени части:

piece_name=part_name “/” “[” 1*digit “]” [“.last”] “.piece”

Существуют следующие ограничения на допустимость для имен порций piece_name, сгенерированных при помощи грамматики:

- Номера порций начинаются с 0 и являются положительными последовательными целыми числами. Номера порций могут дополняться нулями слева.

- Последняя порция набора порций части содержит “.last” в имени порции перед “.piece”.

- Имя порции генерируется из имени логической части перед отображением в имена в физическом пакете.

Хотя необязательно хранить порции в их естественном порядке, такое хранение может обеспечить оптимальную эффективность. Физический пакет, содержащий части с чередованием (разбитые на порции), также может содержать части без чередования (из одной порции), поэтому будет действительным следующий пример:

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

Конкретные отображения

Нижеследующее определяет конкретные отображения для следующих физических форматов: Свободные файлы в файловой системе Windows.

Отображение в свободные файлы в файловой системе Windows.

Чтобы лучше понять, как отображать элементы логической модели в физический формат, рассмотрим базовый случай представления пакета Metro в качестве коллекции свободных файлов в файловой системе Windows. Каждая часть в логическом пакете будет содержаться в отдельном файле (потоке). аждое имя части в логической модели соответствует имени файла.

Логический компонент	Физическое представление
Часть	Файл(ы)
Имя части	Имя файла с путем (который должен выглядеть аналогично URI, изменяет косую черту на обратную косую черту и т.д.).
Тип содержимого части	Файл, содержащий XML, выражающий простой список имен файлов и ассоциированных с ними типов.

Имена частей переводятся в допустимые имена файлов Windows, как изображено в нижеследующей таблице.

Ниже приведены два набора символов, которые являются допустимыми для сегментов логического имени части (сегментов URI) и для имен файлов Windows. Эта таблица раскрывает два важных обстоятельства:

Существуют два допустимых для URI символа, двоеточие (:) и звездочка (*), для которых необходимо ввести управляющий символ при преобразовании URI в имя файла.

Существуют допустимые для имен файлов символы, ^ { }[]#, которые не могут присутствовать в URI (они могут использоваться для специальных целей отображения, подобно чередованию).

“Введение управляющего символа” используется в качестве методики для получения символов допустимых имен файлов, когда имя части содержит символ, который не может использоваться в имени файла. Чтобы ввести управляющий символ, используется символ вставки (^), за которым следует шестнадцатеричное представление символа.

Чтобы отобразить из abs_path (имени части) в имя файла:

удалите первую /

преобразуйте все / в \

введите управляющий символ для символов двоеточия и звездочки

Например, имя части **/a:b/c/d*.xaml** становится следующим именем файла **a^25b\c\d^2a.xaml**.

Для выполнения обратного отображения:

преобразуйте все \ в /

добавьте / в начало строки

уберите управляющие символы заменой ^[hexCode] на соответствующий символ

Из правил грамматики УИР (RFC2396)	Символы, которые допустимы для именования файлов, папок или ярлыков
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"	Символ | ^ ударения (вставки) & Амперсанд ' Апостроф (одинарная кавычка) @ Знак "а" в кружочке ("собака") { Левая фигурная скобка } Правая фигурная скобка [Открывающая квадратная скобка] Закрывающая квадратная скобка , Запятая $ Знак доллара = Знак равенства ! Восклицательный знак - Дефис # Знак номера (Открывающая круглая скобка) Закрывающая круглая скобка % Процент .] Точка + Плюс ~ Тильда _ Символ подчеркивания

Управление версиями и расширяемость

Аналогично другим техническим спецификациям спецификация, содержащаяся в данной заявке, может развиваться с будущими усовершенствованиями. Конструкция первого издания данной спецификации включает в себя планы на будущий обмен документами между программными системами, написанными на основе первого издания, и программными системами, написанными для будущих изданий. Аналогично, данная спецификация дает возможность третьим сторонам создавать расширения к спецификации. Такие расширения могут, например, допускать конструкцию документа, который использует признак некоторого конкретного принтера, в то же самое время все же сохраняя совместимость с другими считывателями, которые не имеют сведений о существовании этого принтера.

Документы, использующие новые версии разметки Fixed.Payload или расширения разметки третьих сторон, требуют от считывателей принять соответствующие решения в отношении поведения (например, как воспроизвести что-либо визуально). Чтобы предоставить рекомендации считывателям, автор документа (или инструментальное средство, которое сгенерировало документ) должен идентифицировать соответствующее поведение для считывателей, встречающих иначе нераспознаваемые элементы или атрибуты. Для расширенных документов является важным этот тип рекомендаций.

Новые принтеры, браузеры и другие клиенты могут реализовывать разнообразную поддержку будущих признаков. Авторы документов, использующие новые версии или расширения, должны осторожно принимать во внимание поведение считывателей,

незнакомых с этими версиями расширений.

Пространство имен управления версиями

Распознавание разметки XML основывается на идентификаторах URI пространства имен. Для любого пространства имен XML, как ожидается, считыватель распознает или все, или ни один из XML-элементов и XML-атрибутов, определенных в этом пространстве имен. Если считыватель не распознает новое пространство имен, то считывателю необходимо выполнить операции визуализации с откатом, как определено в документе.

URI пространства имен XML 'http://PLACEHOLDER/version-control' включает в себя XML-элементы и атрибуты, используемые для создания разметки фиксированной полезной нагрузки, которая является адаптивной к версиям и адаптивной к расширениям. Фиксированным полезным нагрузкам необязательно иметь в себе элементы управления версиями. Чтобы создать адаптивное содержимое, однако, необходимо использовать по меньшей мере один из XML-элементов <ver: Compatibility.Rules> и <ver:AlternativeContent>.

Данная спецификация разметки Fixed-Payload имеет URI xmln, ассоциированный с ней: 'http://PLACEHOLDER/pdl'. Использование этого пространства имен в фиксированной полезной нагрузке будет указывать приложению считывателя, что будут использоваться только элементы, определенные в данной спецификации. Будущие версии данной спецификации будут иметь свои собственные пространства имен. Приложения считывателей, знакомые с новыми пространствами имен, будут знать, как поддерживать расширенный набор элементов атрибутов, определенных в предыдущих версиях. Приложения считывателей, которые не знакомы с новой версией, будут рассматривать URI новой версии, как если бы он был URI некоторого неизвестного расширения языка описания страниц (PDL). Эти приложения могут не знать, что существует отношение между пространствами имен, в которых одно является расширенным набором другого.

Обратная и “прямая” совместимость

В контексте приложений или устройств, поддерживающих системы и способы, описанные в данной заявке, совместимость указывается способностью клиентов производить синтаксический разбор и отображение документов на дисплее, которые были составлены автором с использованием предыдущих версий спецификации или неизвестных расширений или версий спецификации. Различные механизмы управления версиями ориентированы на “обратную совместимость”, предоставляя будущим реализациям клиентов возможность поддерживать документы, основанные на версиях нижнего уровня спецификации, как изображено ниже.

Когда реализованный клиент, такой как принтер, принимает документ, созданный с использованием будущей версии языка разметки, клиент сможет производить синтаксический анализ и понимать доступные варианты визуализации. Способность клиентского программного обеспечения, написанного в соответствии с более старой версией спецификации, обрабатывать *некоторые* документы с использованием признаков более новой версии часто называется “прямой совместимостью”. Документ, написанный с возможностью прямой совместимости, описывается как “адаптивный к версиям”.

Далее, так как реализованным клиентам также необходимо будет поддерживать документы, которые имеют неизвестные расширения, представляющие новые элементы или свойства, различные семантики поддерживают более общий случай документов, которые являются “адаптивными к расширениям”.

Если принтер или программа просмотра встречает расширения, которые являются незнакомыми, она ищет информацию, встроенную вместе с использованием расширения для рекомендации об адаптивной визуализации окружающего содержимого. Эта адаптация включает в себя замену неизвестных элементов или атрибутов содержимым, которое является понятным. Однако адаптация может принимать другие формы, включая полное игнорирование неизвестного содержимого. В отсутствие явно заданных рекомендаций считыватель должен рассматривать присутствие нераспознанного расширения в разметке как состояние ошибки. Если рекомендации не предусмотрены, предполагается, что расширение является существенным для понимания содержимого. Неуспешная визуализация будет зафиксирована и сообщена пользователю.

Чтобы поддерживать эту модель, новые и расширенные версии языка разметки должны логически группировать относящиеся расширения в пространствах имен. Таким образом, авторы документов смогут воспользоваться расширенными признаками, используя минимальное количество пространств имен.

Разметка управления версиями

Словарь XML для поддержки адаптивного к расширениям поведения включает в себя следующие элементы:

Элемент и иерархия управления версиями	Описание
<Compatibility.Rules>	Управляет тем, как синтаксический анализатор реагирует на неизвестный элемент или атрибут.
<Ignorable>	Объявляет, что ассоциированный URI пространства имен является игнорируемым.
<ProcessContent>	Объявляет, что если элемент проигнорирован, то содержимое элемента будет обрабатываться, как если бы оно содержалось в контейнере проигнорированного элемента.
<CarryAlong>	Указывает инструментальным средствам редактирования документа, должно ли быть сохранено игнорируемое содержимое, когда документ модифицируется.
<MustUnderstand>	Изменяет на обратное влияние элемента, объявленного игнорируемым.
<AlternateContent>	В разметке, которая использует признаки управления версиями/расширения, элемент <AlternateContent> ассоциирует разметку замены “fallback” для использования приложениями считывателя, которые не могут обрабатывать разметку, заданную как Preferred.
<Prefer>	Задает предпочтительное содержимое. Этим содержимым будет то, что клиент знаком с признаками управления версиями/расширением.
<Fallback>	Для клиентов нижнего уровня задает, что содержимое “нижнего уровня” должно быть заменено предпочтительным содержимым.

Элемент <Compatibility.Rules>

Compatibility.Rules может быть присоединен к любому элементу, который может хранить присоединенный атрибут, а также к корневому элементу Xaml. Элемент <Compatibility.Rules> управляет тем, как синтаксический анализатор реагирует на неизвестные элементы или атрибуты. Обычно для таких предметов выдается информация об ошибке. Добавление элемента Ignorable к свойству Compatibility.Rules информирует компилятор о том, что предметы из некоторых пространств имен можно игнорировать.

Compatibility.Rules может содержать элементы Ignorable и MustUnderstand. По умолчанию все элементы и атрибуты, как предполагается, являются MustUnderstand. Элементы и атрибуты могут быть сделаны Ignorable посредством добавления элемента Ignorable в свойство Compatibility.Rules его контейнера. Элемент или свойство может быть сделано снова MustUnderstand посредством добавления элемента MustUnderstand к одному из вложенных контейнеров. Один Ignorable или MustUnderstand ссылается на конкретный URI пространства имен внутри этого же

элемента Compatibility.Rules.

Элемент <Compatibility.Rules> влияет на содержимое контейнера, а не на собственный тег или атрибуты контейнера. Чтобы оказывать воздействие на тег или атрибуты контейнера, их контейнер должен содержать правила совместимости. Корневой элемент Xaml может использоваться для задания правил совместимости для элементов, которые иначе были бы корневыми элементами, такие как Canvas. Составной атрибут Compatibility.Rules представляет собой первый элемент в контейнере.

Элемент <Ignorable>

Элемент <Ignorable> объявляет, что можно игнорировать включенный URI пространства имен. Предмет может считаться игнорируемым, если тег <Ignorable> объявлен перед этим предметом в текущем блоке или блоке контейнера и URI пространства имен неизвестен для синтаксического анализатора. Если URI неизвестен, тег Ignorable не принимается во внимание и все предметы становятся понятными. В одном варианте выполнения должны быть понятны все предметы, не объявленные явно как Ignorable. Элемент Ignorable может содержать элементы <ProcessContent> и <CarryAlong>, которые используются для модифицирования того, как элемент игнорируется, а также для предоставления рекомендаций инструментальным средствам редактирования документа, как такое содержимое должно быть сохранено в отредактированных документах.

Элемент <ProcessContent>

Элемент <ProcessContent> объявляет, что, если элемент проигнорирован, содержимое элемента будет обрабатываться так, как если бы оно содержалось в контейнере проигнорированного элемента.

Атрибуты <ProcessContent>

Атрибут	Описание
Elements	Список с разделителями-пробелами имен элементов, для которых необходимо обрабатывать содержимое, или “*”, указывающий, что должно обрабатываться содержимое всех элементов. Атрибут Elements по умолчанию устанавливается в “*”, если он не задан.

Элемент <CarryAlong>

Необязательный элемент <CarryAlong> указывает инструментальным средствам редактирования документа на то, должно ли быть сохранено игнорируемое содержимое, когда модифицируется документ. Способ, которым инструментальное средство редактирования сохраняет или не принимает во внимание игнорируемое содержимое, находится в сфере действия инструментального средства редактирования. Если многочисленные элементы <CarryAlong> ссылаются на один и тот же элемент или атрибут в пространстве имен, последний заданный <CarryAlong> имеет приоритет.

Атрибуты <CarryAlong>

Атрибут	Описание
Elements	Список с разделителями-пробелами имен элементов, которые запрашиваются для сохранения, когда документ редактируется, или “*”, указывающий, что должно быть сохранено содержимое всех элементов в пространстве имен. Атрибут Elements по умолчанию устанавливается в “*”, если он не задан.
Attributes	Список с разделителями-пробелами имен атрибутов в элементах, которые должны быть сохранены, или “*”, указывающий, что все атрибуты элементов должны быть сохранены. Когда элемент проигнорирован и сохранен, все атрибуты сохраняются независимо от содержимого этого атрибута. Этот атрибут имеет влияние только тогда, когда заданный атрибут используется в элементе, который не игнорируется, как в примере ниже. По умолчанию Attribute равен “*”.

Элемент <MustUnderstand>

<MustUnderstand> представляет собой элемент, который изменяет на обратное

воздействие элемента Ignorable. Этот прием полезен, например, когда он объединяется с альтернативным содержимым. Вне области действия, определенной элементом <MustUnderstand>, элемент остается Ignorable.

Атрибуты <MustUnderstand>

Атрибут	Описание
NamespaceUri	URI пространства имен, предметы которого должны быть поняты.

Элемент <AlternateContent>

Элемент <AlternateContent> дает возможность предусматривать альтернативное содержимое, если любая часть заданного содержимого не понята.

Блок AlternateContent использует как блок <Prefer>, так и <Fallback>. Если что-либо в блоке <Prefer> не понято, то тогда используется содержимое блока <Fallback>. Пространство имен объявляется <MustUnderstand>, чтобы указать, что откат должен использоваться. Если пространство имен объявляется игнорируемым и это пространство имен используется в блоке <Prefer>, то не будет использоваться содержимое в блоке <Fallback>.

Примеры разметки управления версиями

Использование <Ignorable>

В этом примере используется фиктивное пространство имен разметки, http://PLACEHOLDER/Circle, которое определяет элемент Circle в его исходной версии и использует атрибут Opacity для Circle, вводимый в будущей версии разметки (версия 2), и свойство Luminance, вводимое в еще более поздней версии разметки (версия 3). Эта разметка остается загружаемой в версиях 1 и 2, а также 3 и позже. Кроме того, элемент <CarryAlong> задает, что v3:Luminance ДОЛЖНА быть сохранена при редактировании, даже когда редактор не понимает v3:Luminance.

Для считывателя версии 1 игнорируются Opacity и Luminance.

Для считывателя версии 2 игнорируется только Luminance.

Для считывателя версии 3 и позже используются все атрибуты.

```
<FixedPanel
     xmlns="http://PLACEHOLDER/fixed-content"
     xmlns:v="http://PLACEHOLDER/versioned-content"
     xmlns:v1="http://PLACEHOLDER/Circle/v1"
     xmlns:v2="http://PLACEHOLDER/Circle/v2"
     xmlns:v3="http://PLACEHOLDER/Circle/v3" >
  <v:Compatibility.Rules>
    <v:Ignorable NamespaceUri=" http://PLACEHOLDER/Circle/v2" />

     <v:Ignorable NamespaceUri=" http://PLACEHOLDER/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>
```

Использование <MustUnderstand>

Следующий пример демонстрирует использование элемента <MustUnderstand>.

```
<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>
```

Использование элемента <MustUnderstand> вызывает появление ошибки при ссылке на v3:Luminance, даже если он был объявлен Ignorable в корневом элементе. Этот прием является полезным при объединении с альтернативным содержимым, которое использует, например, свойство Luminance для Canvas, добавленное взамен в Версии 2 (см. ниже). Вне области действия элемента Canvas свойство Luminance для Circle снова является игнорируемым.

```
<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>
```

Использование <AlternateContent>

Если любой элемент или атрибут объявлен как <MustUnderstand>, но не понят в блоке <Prefer> блока <AlternateContent>, блок <Prefer> пропускается полностью, и блок <Fallback> обрабатывается нормальным образом (т.е. о любых встречаемых предметах MustUnderstand сообщается как об ошибках).

```
<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>
```

Формат расширенного пакета

В нижеследующем объяснении представлено описание конкретного формата файлов. Отдельные основные подзаголовки в этом разделе включают в себя “Введение в формат расширенного пакета”, “Структура расширенного пакета”, “Части фиксированной полезной нагрузки”, “Основы разметки FixedPage”, “Элементы

и свойства фиксированной полезной нагрузки” и “Разметка FixedPage”. Каждый основной подзаголовок имеет один или несколько соответствующих подзаголовков.

Введение в формат расширенного пакета

Описав выше примерную инфраструктуру, описание, которое следует ниже, представляет собой описание конкретного формата, который предусматривается с использованием инструментальных средств, описанных выше. Необходимо оценить и понять, что нижеследующее описание составляет только один примерный формат и, как предполагается, не ограничивает применение заявленного объекта изобретения.

Согласно данному варианту выполнения отдельный пакет может содержать многочисленные полезные нагрузки, причем каждая действует в качестве разных представлений документа. *Полезная нагрузка* представляет собой коллекцию частей, включающую в себя идентифицируемую “корневую” часть, и все части, требуемые для достоверной обработки этой корневой части. Например, полезная нагрузка может быть фиксированным представлением документа, переформатируемым представлением или любым произвольным представлением.

Нижеследующее описание определяет конкретное представление, называемое *фиксированной полезной нагрузкой*. Фиксированная полезная нагрузка имеет корневую часть, которая содержит разметку FixedPanel, которая, в свою очередь, ссылается на части FixedPage. Вместе они описывают точную визуализацию многостраничного документа.

Пакет, который хранит по меньшей мере одну фиксированную полезную нагрузку и следует другим описанным ниже правилам, как известно, упоминается как *расширенный пакет*. Считыватели и составители расширенных пакетов могут реализовывать свои собственные синтаксические анализаторы и средства визуализации, основанные на спецификации формата расширенного пакета.

Прзнаки расширенных пакетов

Согласно описанному варианту выполнения расширенные пакеты обращаются к требованиям, которые информационные работники имеют для распространения, архивирования и визуализации документов. Используя известные правила визуализации, расширенные пакеты могут однозначно и точно воспроизводиться или печататься из формата, в котором они сохранены, без привязывания клиентских устройств или приложений к определенным операционным системам или библиотекам служб. Кроме того, так как расширенная полезная нагрузка выражается нейтральным независимым от приложения образом, документ обычно может просматриваться и печататься без приложения, использованного для создания пакета. Для получения такой возможности понятие *фиксированной полезной нагрузки* вводится и содержится в расширенном пакете.

Согласно описанному варианту выполнения фиксированная полезная нагрузка имеет фиксированное количество страниц и всегда одинаковое количество разделителей страниц. Предварительно определяется раскладка всех элементов на странице в фиксированной полезной нагрузке. Каждая страница имеет фиксированный размер и ориентацию. Как таковые не должны выполняться никакие вычисления раскладки на стороне потребления, и содержимое может просто визуализироваться. Это применяется не только к графике, но также к тексту, который представляется в фиксированной полезной нагрузке с точным типографским размещением. Содержимое страницы (текст, графика, изображения) описывается с использованием мощного, но простого набора визуальных примитивов.

Расширенные пакеты поддерживают многочисленные механизмы организации

страниц. Группа страниц “склеивается” вместе одна за другой в “FixedPanel”. Эта группа страниц грубо эквивалента традиционному многостраничному документу. FixedPanel затем может дополнительно участвовать в композиции - процессе построения последовательностей и выборов для сборки “составного” документа.

В изображенном и описанном варианте выполнения расширенные пакеты поддерживают определенный вид последовательности, названный *последовательностью FixedPanel*, которая может использоваться, например, для склеивания вместе набора FixedPanel в один большой “документ”. Вообразите, например, склеивание вместе двух документов, которые поступили из различных источников: двухстраничный сопроводительный меморандум (FixedPanel) и двадцатистраничный отчет (FixedPanel).

Расширенные пакеты поддерживают некоторое количество определенных селекторов, которые могут использоваться при построении пакетов документов, содержащих альтернативные представления “одного и того же” содержимого. В частности, расширенные пакеты допускают выбор, основанный на *языке*, *способности к воспроизведению цветов* и *размере страницы*. Таким образом, можно иметь, например, двуязычный документ, который использует селектор для выбора между английским представлением и французским представлением документа.

В дополнение к этим простым использованиям селектора и последовательности для составления расширенного пакета важно отметить, что селекторы и последовательности также могут ссылаться на другие селекторы и последовательности, таким образом позволяя строить составные иерархии. Точные правила в отношении того, что можно и что нельзя делать, в соответствии с данным вариантом выполнения, определяются ниже в разделе, озаглавленном “Структура расширенного пакета”.

Кроме того, расширенный пакет может содержать дополнительные полезные нагрузки, которые *не* являются фиксированными полезными нагрузками, но вместо этого имеют более богатые возможности и, возможно, редактируемые представления документа. Это дает возможность пакету содержать обогащенный редактируемый документ, который хорошо работает в приложении редактора, а также как представление, которое визуально точное и может просматриваться без редактирующего приложения.

Наконец, согласно данному варианту выполнения расширенные пакеты поддерживают то, что известно как *этикетка печати*. Этикетка печати обеспечивает установки, которые должны использоваться, когда пакет печатается. Эти этикетки печати могут быть прикреплены различным образом для достижения существенной гибкости. Например, этикетка печати может быть “прикреплена” ко всему пакету, и ее установки будут влиять на весь пакет. Этикетки печати могут дополнительно прикрепляться на более нижних уровнях в структуре (например, к отдельным страницам), и эти этикетки печати будут обеспечивать установки переопределения для использования при печати части, к которой они прикреплены.

Структура расширенного пакета

Как описано выше, расширенный пакет поддерживает набор признаков, включающий в себя “фиксированные” страницы, FixedPanel, композицию, этикетки печати и т.п. Эти признаки представлены в пакете с использованием основных компонентов модели пакета: частей и отношений. В данном разделе и относящихся к нему подразделах предусматривается полное определение “расширенного пакета”, включая описание того, как все эти части и отношения должны быть собраны,

связаны и т.д.

Обзор структуры расширенного пакета

На фиг.10 изображен примерный расширенный пакет и в данном варианте выполнения каждый из допустимых типов частей, который может составлять пакет или быть обнаружен в пакете. В таблице, предусмотренной ниже, перечисляется каждый допустимый тип части и предусматривается описание каждого:

FixedPage application/xml+FixedPage-PLACEHOLDER	Каждая часть FixedPage представляет содержимое страницы
FixedPanel application/xml+FixedPanel-PLACEHOLDER	Каждый FixedPanel склеивает вместе набор FixedPage по порядку
Font	Шрифты могут быть встроены в пакет, чтобы гарантировать надежное воспроизведение глифов документа
Image image/jpeg image/png	Могут быть включены части с изображением
Composition Parts application/xml+Selector+[XXX] Application/xml+Sequence+[XXX]	Селекторы и последовательности могут использоваться для построения “композиционного” блока, вводя высокоуровневую организацию в пакет
Descriptive Metadata application/xml+SimpleTypeProperties-PLACEHOLDER	Описательные метаданные (например, заголовок, ключевые слова) могут быть включены для документа
Print Ticket application/xml+PRINTTICKET-PLACEHOLDER	Этикетка печати может быть включена, чтобы предоставить установки для использования при печати пакета

Так как расширенный пакет был разработан, чтобы он был документом “с просмотром и печатью где угодно”, считыватели и составители расширенных пакетов должны совместно использовать общие, однозначно определенные ожидаемые результаты того, что составляет “допустимый” расширенный пакет. Чтобы получить определение “допустимого” расширенного пакета, ниже сначала определяются несколько понятий.

Расширенные композиционные части

Расширенный пакет должен содержать по меньшей мере один FixedPanel, который является “обнаруживаемым” в результате прослеживания композиционного блока с начальной части пакета. Согласно описанному варианту выполнения процесс обнаружения следует нижеприведенному алгоритму:

- Рекурсивный обход графа композиционных частей, начиная с начальной части пакета.

- При выполнении этого обхода обходить только композиционные части, оторые являются *расширенными композиционными частями* (описанные ниже).

- Локализовать все конечные узлы (те, которые без выходящих дуг) на ребре графа.

Эти конечные узлы ссылаются (посредством их элементов <item>) на набор частей, названный *корни расширенной полезной нагрузки*.

Фиксированная полезная нагрузка

Фиксированная полезная нагрузка представляет собой полезную нагрузку, корневой частью которой является часть FixedPanel. Например, каждая из фиксированных полезных нагрузок на фиг.10 имеет в качестве ее корневой части ассоциированную часть FixedPanel. Полезная нагрузка включает в себя полное замкнутое выражение всех частей, требуемых для допустимой обработки FixedPanel. Они включают в себя:

- саму FixedPanel;

- все FixedPage, на которые имеется ссылка из FixedPanel;
- все части с изображением, на которые ссылается (непосредственно или косвенно при помощи селектора) любая из FixedPage в полезной нагрузке;
- все *расширенные селекторы* (как описано ниже), на которые имеется ссылка непосредственно или косвенно из кистей изображения, используемых в любой FixedPage в полезной нагрузке;
- все шрифтовые части, на которые ссылается любая из FixedPage в полезной нагрузке;
- все части с описательными метаданными, присоединенные к любой части в фиксированной полезной нагрузке; и
- любые этикетки печати, присоединенные к любой части в фиксированной полезной нагрузке.

Правила допустимости для расширенного пакета

С вышеупомянутыми определениями на месте теперь описываются правила согласования, которые описывают “допустимый” расширенный пакет согласно описанному варианту выполнения:

- расширенный пакет должен иметь начальную часть, определенную с использованием стандартного механизма отношения пакета, как описано выше;
- начальной частью расширенного пакета должен быть или селектор, или последовательность;
- расширенный пакет должен иметь по меньшей мере один корень расширенной полезной нагрузки, которым является FixedPanel;
- части PrintTicket могут быть присоединены к любой из композиционных частей, частей PrintTicket или любой из частей FixedPage, идентифицированных в FixedPanel. В настоящем примере это выполняется при помощи отношения http://PLACEHOLDER/HasPrintTicketRel;
- PrintTicket могут быть присоединены к любой или всем из этих частей;
- любая заданная часть должна иметь не более одной присоединенной PrintTicket;
- часть Descriptive Metadata может быть присоединена к любой части в пакете;
- каждый объект Font в FixedPayload должен удовлетворять правилам формата шрифтов, определенным в разделе “Шрифтовые части”;
- ссылки на изображения из любой FixedPage в фиксированной полезной нагрузке могут указывать на селектор, который может выполнять выбор (потенциально рекурсивно через другие селекторы) для обнаружения фактической части с изображением, подлежащей визуализации;
- каждый объект Image, используемый в фиксированной полезной нагрузке, должен удовлетворять правилам формата шрифтов, определенным в разделе “Части с изображением”;
- для любой шрифтовой части, части с изображением или части селектора, ссылаемой из FixedPage (непосредственно или косвенно через селектор), должно быть отношение “требуемой части” (имя отношения = http://mmcf-fixed-RequiredResource-PLACEHOLDER) из ссылающейся FixedPage на часть, на которую сделана ссылка.

Расширенные композиционные части

Хотя расширенный пакет может содержать многие типы композиционной части, только четко определенный набор типов композиционных частей имеет четко определенное поведение согласно этому документу. Эти композиционные части с четко определенным поведением называются *расширенные композиционные части*. Части, отличные от этих, не являются релевантными при определении допустимости

расширенного пакета.

Следующие типы композиционных частей определяются как расширенные композиционные части:

Language Selector (селектор языка) application/xml+selector+language	Выбирает между представлениями, основываясь на их естественном языке.
Color Selector (селектор цвета) application/xml+selector+color	Выбирает между представлениями, основываясь на том, являются ли они монохромными или цветными.
Page Size Selector (селектор размера страницы) application/xml+selector+pagesize	Выбирает между представлениями, основываясь на их размере страницы.
Content Type Selector (селектор типа содержимого) application/xml+selector+contenttype	Выбирает между представлениями, основываясь на том, могут ли их типы содержимого быть поняты системой.
Fixed Sequence (фиксированная последовательность) application/xml+sequence+fixed	Объединяет дочерние элементы, которыми являются фиксированное содержимое, в последовательность.

Расширенные селекторы

Эти композиционные части селектора, определенные как расширенные композиционные части, называются *расширенные селекторы*. Как отмечено выше, *селектор языка* выбирает между представлениями, основываясь на их естественном языке, таком как английский или французский. Чтобы обнаружить этот язык, данный селектор инспектирует каждый из его объектов. Рассматриваются только те, которые являются XML. Для них корневой элемент каждого из них инспектируется с целью определения этого языка. Если не присутствует атрибут xml:lang, то часть игнорируется. Селектор затем рассматривает каждую из этих частей по очереди, выбирая первую, язык которой совпадает с языком системы, используемым по умолчанию.

Селектор цвета выбирает между представлениями, основываясь на том, являются ли они монохромными или цветными. *Селектор размера страницы* выбирает между представлениями, основываясь на размере их страницы. *Селектор типа содержимого* выбирает между представлениями, основываясь на том, могут ли их типы содержимого быть поняты системой.

Расширенные последовательности

Эти композиционные части последовательности, определенные как расширенные композиционные части, называются *расширенные последовательности*. *Фиксированная последовательность* комбинирует дочерние элементы, которые являются фиксированным содержимым, в последовательность.

Части фиксированных полезных нагрузок

Фиксированная полезная нагрузка может содержать следующие виды частей: часть FixedPanel, часть FixedPage, части с изображением, шрифтовые части, части этикетки печати и части с описательными метаданными, каждая из которых описывается ниже под своим собственным подзаголовком.

Часть FixedPanel

Структура документа Fixed-Payload идентифицирует FixedPage как часть корешка, как показано ниже. Отношения между частью корешка и частями страницы определяются в потоке отношений для корешка. Часть FixedPanel является типом содержимого application/xml+PLACEHOLDER.

Корешок содержимого Fixed-Payload задается в разметке посредством включения элемента <FixedPanel> в элемент <Document>. В примере ниже элемент <FixedPanel> задает источники страниц, которые хранятся в корешке.

<!- Корешок -->

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

Элемент <Document>

Элемент <Document> не имеет атрибутов и должен иметь только один дочерний элемент: <FixedPanel>.

Элемент <FixedPanel>

Элемент <FixedPanel> представляет собой корешок документа, логически связывающий упорядоченную последовательность страниц вместе в один многостраничный документ. Страницы всегда задают их собственные ширину и высоту, но элемент <FixedPanel> также может, в необязательном порядке, задавать высоту и ширину. Эта информация может использоваться для множества целей, включая, например, выбор между альтернативными представлениями, основываясь на размере страницы. Если элемент <FixedPanel> задает высоту и ширину, он обычно будет выравниваться по ширине и высоте страниц в <FixedPanel>, но эти размеры не задают высоту и ширину отдельных страниц.

В нижеследующей таблице просуммированы атрибуты FixedPanel согласно описанному варианту выполнения.

Атрибут <FixedPanel>	Описание
PageHeight	Типовая высота страниц, содержащихся в <FixedPanel>. Необязательный.
PageWidth	Типовая ширина страниц, содержащихся в <FixedPanel>. Необязательный.

Элемент <PageContent> является единственным допустимым дочерним элементом элемента <FixedPanel>. Элементы <PageContent> находятся в последовательном порядке разметки, соответствующем порядку страниц документа.

Элемент <PageContent>

Каждый элемент <PageContent> ссылается на источник содержимого для одной страницы. Для определения количества страниц в документе нужно подсчитать количество дочерних элементов <PageContent>, содержащихся в <FixedPanel>.

Элемент <PageContent> не имеет допустимых дочерних элементов и имеет единственный требуемый атрибут Source, который ссылается на часть FixedPage для содержимого страницы.

Как и с элементом <FixedPanel>, элемент <PageContent> может, в необязательном порядке, включать в себя атрибут PageHeight и PageWidth, при этом отражая размер одной страницы. Требуемый размер страницы задается в части FixedPage; необязательный размер на <PageContent> является только рекомендательным. Атрибуты размера <PageContent> дают возможность приложениям, таким как программы просмотра документов, быстро выполнять оценки визуальной раскладки для документа без загрузки и синтаксического анализа всех отдельных частей FixedPage.

В таблице, представленной непосредственно ниже, просуммированы атрибуты <PageContent> и представлено описание атрибутов.

Атрибут <PageContent>	Описание
Source	Строка URI, которая ссылается на содержимое страницы, хранящееся в отдельной части в пакете. Содержимое идентифицируется как часть в пакете. Требуется.

PageHeight	Необязательный
PageWidth	Необязательный

Строка URI содержимого страницы должна ссылаться на расположение части содержимого относительно пакета.

Часть FixedPage

Каждый элемент <PageContent> в <FixedPanel> ссылается посредством имени (URI) на часть FixedPage. Каждая часть FixedPage содержит разметку FixedPage, описывающую визуализацию одной страницы содержимого. Часть FixedPage является Content Type application/xml+PLACEHOLDER-FixedPage.

Описание FixedPage в разметке

Ниже приведен пример того, как разметка содержимого источника может производить поиск страницы, на которую имеется ссылка в иллюстративной разметке корешка выше (<PageContent Source="p1.xml" />).

```
//    /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>
```

В таблице внизу просуммированы свойства FixedPage и представлено описание свойств.

Свойство FixedPage	Описание
PageHeight	Требуется
PageWidth	Требуется

Порядок считывания в разметке FixedPage

В одном варианте выполнения порядок разметки в дочерних элементах Glyph, содержащихся в FixedPage, должен быть такой же, что и требуемый порядок считывания текстового содержимого страницы. Этот порядок считывания может использоваться как для интерактивного выбора/копирования последовательного текста из FixedPage в программу просмотра, так и для обеспечения возможности доступа к последовательному тексту при помощи технологии доступности. Ответственностью приложения, генерирующего разметку FixedPage, является обеспечение этого соответствия между порядком разметки и порядком считывания.

Части с изображением

Поддерживаемые форматы

Согласно описанному варианту выполнения части с изображением, используемые FixedPage в расширенном пакете, могут быть в фиксированном количестве формаов, например в формате PNG (переносимая сетевая графика) или в формате JPEG, хотя могут использоваться другие форматы.

Шрифтовые части

Согласно описанному варианту выполнения расширенные пакеты поддерживают ограниченное количество форматов шрифтов. В изображенном и описанном варианте выполнения поддерживаемый формат шрифтов включает в себя формат TrueType и формат OpenType.

Как понятно для специалиста в данной области техники, формат шрифта OpenType

является расширением формата шрифта TrueType, добавляя поддержку данных шрифтов PostScript и сложной типографской раскладки. Файл шрифта OpenType содержит данные в табличном формате, которые содержат или контурный шрифт TrueType, или контурный шрифт PostScript.

Согласно описанному варианту выполнения следующие форматы шрифтов не поддерживаются в расширенном пакете: шрифт Type 1 компании Adobe, растровый шрифт, шрифт со скрытым атрибутом (использует флаг системы для принятия решения в отношении того, перечислять его или нет), векторные шрифты и шрифт EUDC (определяемый конечным пользователем шрифт) (именем семейства шрифтов которых является EUDC).

Разделение шрифтов на подгруппы

Фиксированные полезные нагрузки представляют весь текст с использованием элементов *Glyphs*, подробно описанных ниже. Так как в данном варианте выполнения формат является фиксированным, то можно разделить шрифты на подгруппы, чтобы содержать только глифы, требуемые для FixedPayload. Поэтому шрифты в расширенных пакетах могут быть разделены на подгруппы, основываясь на использовании глифов. Хотя разделенный на подгруппы шрифт не будет содержать все глифы в исходном шрифте, разделенный на подгруппы шрифт должен быть действительным файлом шрифта OpenType.

Части этикетки печати

Части этикетки печати обеспечивают установки, которые могут использоваться, когда пакет печатается. Эти этикетки печати могут присоединяться различным образом для достижения существенной гибкости. Например, этикетка печати может “присоединяться” ко всему пакету, и ее установки будут воздействовать на весь пакет. Этикетки печати, кроме того, могут прикрепляться на более нижних уровнях в структуре (например, к отдельным страницам), и эти этикетки печати будут обеспечивать установки переопределения для использования при печати части, к которой они прикреплены.

Описательные метаданные

Как отмечено выше, части описательных метаданных предоставляют составителям или создателям пакетов средство для хранения значений свойств, которые дают возможность считывателям пакетов надежно обнаруживать значения. Эти свойства обычно используются для записи дополнительной информации о пакете в целом, а также отдельных частей в контейнере.

Основы разметки FixedPage

В этом разделе описывается некоторая основная информация, связанная с разметкой FixedPage, и он включает в себя следующие разделы: “Фиксированная полезная нагрузка и другие стандарты разметки”, “Модель разметки FixedPage”, “Ресурсы и ссылки на ресурсы” и “Модель рисования FixedPage”.

Фиксированная полезная нагрузка и другие стандарты разметки

Разметка FixedPanel и FixedPage для фиксированной полезной нагрузки в расширенном пакете представляет собой поднабор из разметки расширяемого языка разметки приложений (XAML) Avalon системы Windows® Longhorn. Т.е. хотя разметка фиксированной полезной нагрузки стоит отдельно как независимый формат разметки XML (как документировано в данном документе), она загружается так же, как и в системах Longhorn, и визуализирует воспроизведение в режиме WYSIWYG (“что видишь, то и получаешь”) исходного многостраничного документа.

В качестве некоторых основных принципов разметки XAML рассмотрим

следующее. Разметка XAML представляет собой механизм, который дает возможность пользователю задавать иерархию объектов и логику программирования после объектов в качестве основанного на XML языка разметки. Это обеспечивает возможность описания объектной модели в XML. Это дает возможность доступа в XML к классам расширения, таким как классы в общеязыковой среде выполнения (OCB, CLR) .NET Framework корпорации Microsoft. Механизм XAML обеспечивает прямое отображение тегов XML в объекты OCB и возможность представлять соответствующий код в разметке. Необходимо оценить и понять, что различным реализациям нет необходимости специально использовать основанную на OCB реализацию XAML. Скорее, основанная на OCB реализация представляет только один путь, по которому XAML может использоваться в контексте вариантов выполнения, описанных в данном документе.

Более конкретно, рассмотрим следующее в связи с фиг.11, на которой изображено примерное отображение принципов OCB (компоненты левой стороны) в XML (компоненты правой стороны). Пространства имен обнаруживаются в объявлении xmlns с использованием принципа OCB, названного отражением. Классы отображаются непосредственно в теги XML. Свойства и события отображаются непосредственно в атрибуты. Используя данную иерархию, пользователь может задавать дерево иерархии любых объектов OCB в файлах разметки XML. Файлы xaml представляют собой файлы xml с расширением .xaml и тип среды application/xaml+xml. Файлы xaml имеют один корневой тег, который обычно задает пространство имен, используя атрибут xmlns. Пространство имен может задаваться в других типах тегов.

Продолжая, теги в файле xaml обычно отображаются в объекты OCB. Тегами могут быть элементы, составные свойства, определения или ресурсы. Элементы представляют собой объекты OCB, экземпляры которых обычно создаются во время выполнения и образуют иерархию объектов. Теги составных свойств используются для установки свойства в родительском теге. Теги определения используются для добавления кода в страницу и определения ресурсов. Тег ресурса обеспечивает возможность повторного использования дерева объектов простым заданием дерева в качестве ресурса. Теги определения также могут определяться внутри другого тега в качестве атрибута xmlns.

Если документ соответствующим образом описывается разметкой (обычно составителем), разметка может быть синтаксически проанализирована и обработана (обычно считывателем). Сконфигурированный соответствующим образом синтаксический анализатор определяет из корневого тега, в каких сборках и пространствах имен OCB должен производится поиск для обнаружения тега. При многих экземплярах синтаксический анализатор ищет и находит файл определения пространства имен в URI, заданном атрибутом xmlns. Файл определения пространства имен предоставляет имя сборок и их путь установки и список пространств имен OCB. Когда синтаксический анализатор встречает тег, синтаксический анализатор определяет, на какой класс OCB ссылается тег, используя xmlns тега и файл определения xmlns для этого xmlns. Синтаксический анализатор производит поиск в том порядке, в каком сборки и пространства имен заданы в файле определения. Когда он обнаруживает совпадение, синтаксический анализатор создает экземпляр объекта класса.

Таким образом, механизм, описанный непосредственно выше и более полно в заявке, включенной выше в качестве ссылки, дает возможность представлять объектные модели в основанном на XML файле с использованием тегов разметки. Эта

возможность представления объектных моделей в качестве тегов разметки может использоваться для асинхронного или синхронного создания рисунков векторной графики, документов с фиксированным форматом, адаптивных к потоку документов и пользовательских интерфейсов (ПИ) приложения.

В изображенном и описанном варианте выполнения разметка фиксированной полезной нагрузки представляет собой очень минимальный, почти полностью экономичный поднабор примитивов визуализации XAML Avalon. Она представляет визуально все, что может быть представлено в Avalon, с высокой точностью воспроизведения. Разметка фиксированной полезной нагрузки представляет собой поднабор элементов и свойств XAML Avalon - плюс дополнительные соглашения, канонические формы или ограничения в использовании по сравнению с XAML Avalon.

Радикально минимальный определенный набор разметки фиксированной полезной нагрузки снижает затраты, связанные с реализацией и тестированием считывателей расширенных пакетов, таких как процессоры растрирования изображений (ПРИ, RIP) принтера или интерактивные приложения просмотра, а также снижает сложность и объем памяти, занимаемый ассоциированным синтаксическим анализатором. Экономичный набор разметки также минимизирует возможности для разделения на подгруппы, ошибки или несоответствия между составителями и считывателями расширенных пакетов, делая формат и его экосистему по своей природе более надежными.

В дополнение к минимальной разметке фиксированной полезной нагрузки расширенный пакет задает разметку для дополнительной семантической информации для поддержки программ просмотра или представлений документов с расширенными пакетами с признаками, такими как гиперссылки, структура и навигация разделов/состава текста, выбор текста и доступность документа.

Наконец, используя описанные выше механизмы управления версиями и расширяемости, можно дополнить минимальную разметку фиксированной полезной нагрузки боле богатым набором элементов для конкретных целевых приложений, программ просмотра или устройств потребления.

Модель разметки FixedPage

В изображенном и описанном варианте выполнения часть FixedPage выражается основанным на XML языком разметки, основанным на XML-элементах, XML-атрибутах и пространствах имен XML. Три пространства имен XML определяются в данном документе для включения в разметку FixedPage. Одно такое пространство имен ссылается на элементы и атрибуты управления версиями, определенные в другом месте в данном описании изобретения. Основным пространством имен, используемым для элементов и атрибутов в разметке FixedPage, является “http://schemas.microsoft.com/MMCF-PLACEHOLDER-FixedPage”. И, наконец, разметка FixedPage вводит понятие “Ресурсы”, которое требует третье пространство имен, описанное ниже.

Хотя разметка FixedPage выражается с использованием XML-элементов и XML-атрибутов, ее спецификация основывается на высокоуровневой абстрактной модели “Содержимого” и “Свойств”. Элементы FixedPage все выражаются как XML-элементы. Только небольшое количество элементов FixedPage может хранить “Содержимое”, выраженное как дочерние XML-элементы. Но значение свойства может быть выражено с использованием XML-атрибута или с использованием дочернего XML-элемента.

Разметка FixedPage также зависит от двойных принципов Словаря рсурсов и

Ссылки на ресурсы. Комбинация Словаря ресурсов и многочисленных Ссылок на ресурсы дает возможность совместного использования единственного значения свойства многочисленными свойствами многочисленных элементов разметки FixedPage.

Свойства в разметке FixedPage

В изображенном и описанном варианте выполнения существуют три формы разметки, которые могут использоваться для задания значения свойства разметки FixedPage.

Если свойство задается с использованием ссылки на ресурсы, то тогда имя свойства используется в качестве имени XML-атрибута, и специальный синтаксис для значения атрибута указывает присутствие ссылки на ресурсы. Синтаксис для выражения ссылок на ресурсы описывается в разделе, озаглавленном “Ресурсы и ссылки на ресурсы”.

Любое значение свойства, которое не задается в качестве ссылки на ресурсы, может быть выражено в XML с использованием вложенного дочернего XML-элемента, идентифицирующего свойство, значение которого устанавливается. Этот “Синтаксис составного свойства” описывается ниже.

Наконец, некоторые значения свойств без ссылки на ресурсы могут быть выражены как строки простого текста. Хотя все такие значения свойства могут выражаться с использованием Синтаксиса составного свойства, они также могут выражаться с использованием синтаксиса простого XML-атрибута.

Для любого заданного элемента любое свойство может быть установлено не более одного раза, независимо от синтаксиса, используемого для задания значения.

Синтаксис простого атрибута

Для значения свойства, выражаемого простой строкой, может использоваться синтаксис XML-атрибута для задания значения свойства. Например, принимая во внимание элемент разметки FixedPage, названный “SolidColorBrush”, со свойством, названным “Color”, следующий синтаксис может использоваться для задания значения свойства:

<!-- Синтаксис простого атрибута -->

<SolidColorBrush Color=”#FF0000” />

Синтаксис составного свойства

Значения некоторых свойств не могут быть выражены в виде простой строки, например, XML-элемент используется для описания значения свойства. Такое значение свойства не может быть выражено с использованием синтаксиса простого атрибута. Но они могут быть выражены с использованием синтаксиса составного свойства.

В синтаксисе составного свойства используется дочерний XML-элемент, но имя XML-элемента выводится из комбинации имени родительского элемента и имени свойства, разделенных точкой. Принимая во внимание элемент разметки FixedPage <Path>, который имеет свойство “Fill”, которое может быть утановлено в <SolidColorBrush>, следующая разметка может использоваться для установки свойства “Fill” элемента <Path>:

<!-- Синтаксис составного свойства -->

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

Синтаксис составного свойства может использоваться даже в тех случаях, когда

был бы достаточным синтаксис простого атрибута для выражения значения свойства. Поэтому пример предыдущего раздела

```
<!-- Синтаксис простого атрибута -->
<SolidColorBrush Color="#FF0000" />
```

Может быть выражен вместо этого синтаксисом составного свойства:

```
<!- Синтаксис составного свойства -->
<SolidColorBrush>
<SolidColorBrush.Color>#FF0000</SolidColorBrush.Color>
</SolidColorBrush>
```

При задании значения свойства с использованием синтаксиса составного свойства дочерние XML-элементы, представляющие “Property”, должны появляться перед дочерними XML-элементами, представляющими “Contents”. Не является важным порядок отдельных дочерних XML-элементов составного свойства, только то, что они появляются вместе перед любым “Contents” родительского элемента.

Например, при использовании свойств как Clip, так и RenderTransform элемента <Canvas> (описанного ниже), оба должны появляться перед любым содержимым <Path> и <Glyphs> элемента <Canvas>:

```
<Canvas>
        <!--  Сначала, относящиеся к свойству дочерние элементы  -->
        <Canvas.RenderTransform>
                <MatrixTransform Matrix="1,0,0,1,0,0">
        </Canvas.RenderTransform>
        <Canvas.Clip>
                <PathGeometry>
                        …
                </PathGeometry>
        </Canvas.Clip>

        <!-- Затем "Contents"  -->
        <Path …>
                …
        </Path>
        <Glyphs …>
                …
        </Glyphs>
</Canvas>
```

Ресурсы и ссылки на ресурсы

Словари ресурсов могут использоваться для хранения совместно используемых значений свойств, причем каждое называется ресурс. Любое значение свойства, которое само представляет собой элемент разметки FixedPage, может содержаться в Словаре ресурсов. Каждый ресурс в Словаре ресурсов имеет имя. Имя ресурса может использоваться для ссылки на ресурс из XML-атрибута свойства.

В изображенном и описанном варианте выполнения элементы <Canvas> и <FixedPage> могут нести Словарь ресурсов. Словарь ресурсов выражается в разметке как свойство элементов <Canvas> и <FixedPage> в свойстве, названном “Resources”. Однако отдельные значения ресурсов встраиваются непосредственно в XML-элемент <FixedPage.Resources> или <Canvas.Resources>. Синтаксически разметка для <Canvas.Resources> и <FixedPage.Resources> напоминает разметку для элементов разметки с “Contents”.

Согласно данному варианту выполнения <Canvas.Resources> или <FixedPage.Resources> должны предшествовать любому значению свойства синтаксиса составного свойства в <Canvas> или <FixedPage>. Они, аналогично, должны предшествовать любому “Contents” в <Canvas> или <FixedPage>.

Определение словарей ресурсов фиксированной полезной нагрузки

Любой <FixedPage> или <Canvas> может нести Словарь ресурсов, выраженный с

использованием XML-элемента <Canvas.Resources>. Каждому элементу в единственном словаре ресурсов дается уникальное имя, идентифицированное с использованием XML-атрибута, ассоциированного с этим элементом. Чтобы отличить этот атрибут “Name” от тех атрибутов, которые соответствуют свойствам, атрибут Name берется из пространства имен, отличного от пространств имен элементов FixedFormat. URI для этого пространства имен XML представляет собой “http: //schemas.microsoft.com/PLACEHOLDER-for-resources”. В примере ниже определяются две геометрические формы: одна для прямоугольника и другая для окружности.

```
<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>
```

Ссылка на ресурсы

Чтобы установить значение свойства для одного из ресурсов, определенных выше, следует использовать значение XML-атрибута, которое охватывает имя ресурса скобками {}. Например, “{Rectangle}” будет означать, что используется геометрическая форма. В образце разметки ниже регион прямоугольника, определенный геометрическими объектами в словаре, будет закрашиваться посредством SolidColorBrush.

```
<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>
```

Согласно данному варианту выполнения ссылка на ресурсы не должна происходить в пределах определения ресурса в Словаре ресурсов.

Правила ограничения для разрешения ссылок на ресурсы

Хотя взятое в отдельности имя Name не может использоваться дважды в одном и том же Словаре ресурсов, одно и то же имя может использоваться в двух различных Словарях ресурсов в одной части FixedPage. Кроме того, Словарь ресурсов внутреннего <Canvas> может повторно использовать Name, определенное в Словаре ресурсов некоторого внешнего <Canvas> или <FixedPage>.

Когда ссылка на ресурсы используется для установки свойства элемента, в различных Словарях ресурсов производится поиск ресурса с данным именем. Если элементом, несущим свойство, является <Canvas>, то тогда производится поиск ресурса с требуемым именем в Словаре ресурсов (если он присутствует) этого <Canvas>. Если элементом не является <Canvas>, то тогда поиск начинается с ближайшего содержащего <Canvas> или <FixedPage>. Если требуемое имя не

определено в Словаре ресурсов первоначального поиска, то тогда принимается во внимание следующий ближайший содержащий <Canvas> или <FixedPage>. Происходит ошибка, если поиск продолжался до корневого элемента <FixedPage>, и ресурс с требуемым именем не был обнаружен в Словаре ресурсов, ассоциированном с этим <FixedPage>.

Пример ниже демонстрирует эти правила.

```
<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>
              <!--  Следующая Path будет закрашиваться цветом #000000-->
              <Path Fill="{FavoriteColorFill}">
                     <Path.Data>
                            ...
                     </Path.Data>
              </Path>

              <Canvas>
                     <!-- Следующая траектория будет закрашиваться цветом #808080 -->
                     <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>
```

Модель рисования FixedPage

Элемент FixedPage (или вложенный дочерний элемент Canvas) представляет собой элемент, на котором визуализируются другие элементы. Управление размещением содержимого осуществляется посредством свойств, заданных для FixedPage (или Canvas), свойств, заданных для элементов на FixedPage (или Canvas), или композиционных правил, определенных для пространства имен Fixed-Payload.

Использование Canvas для расположения элементов

При фиксированной разметке все элементы располагаются относительно текущего начала (0,0) системы координат. Текущее начало координат может перемещаться в результате применения атрибута RenderTransform к каждому элементу FixedPage или Canvas, который содержит элемент.

Нижеследующий пример иллюстрирует расположение элементов при помощи RenderTransform.

```
<Canvas>
        <Canvas.Resources>
                <PathGeometry def:Name="StarFish">
                        <!-- Различные PathFigure в этом месте -->
                        ...
                </PathGeometry>
                <PathGeometry def:Name="LogoShape">
                        <!-- Различные PathFigure в этом месте -->
                        ...
                </PathGeometry>
        </Canvas.Resources>

        <!-- Нарисовать зеленый StarFish и красный LogoShape, смещенные на 100 вправо и
    и на 50 вниз -->
        <Canvas>
                <Canvas.RenderTransform>
                        <MatrixTransform Matrix="1,0,0,1,100,50"/>
                </Canvas.RenderTransform>
                <Path Fill="#00FF00" Data="{StarFish}"/>
                <Path Fill="#FF0000" Data="{LogoShape}"/>
        </Canvas>

        <!-- Нарисовать зеленый StarFish и красный LogoShape, смещенные на 200 вправо и на
             250 вниз -->
        <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>
```

Системы координат и единицы

Согласно изображенному и описанному варианту выполнения система координат первоначально устанавливается так, что одна единица в этой системе координат равняется 1/96 дюйма, выраженной в виде значения с плавающей точкой, начало (0,0) системы координат находится в левом верхнем углу элемента FixedPage.

Атрибут RenderTransform может задаваться на каждом дочернем элементе для применения аффинного преобразования к текущей системе координат.

Размеры страницы

Размеры страницы задаются параметрами “PageWidth” и “PageHeight” на элементе FixedPage.

Правила композиции

FixedPage используют модель рисования с альфа-каналом. Согласно описанному варианту выполнения композиция должна происходить согласно этим правилам и в следующем порядке:

- FixedPage (или любой вложенный Canvas) предполагается в виде неограниченной поверхности, на которой рисуются дочерние элементы в том порядке, в каком они появляются в разметке. Альфа-канал этой поверхности инициализируется в “0,0” (все прозрачно). На практике идеальная неограниченная поверхность может предполагаться в виде буфера растра, достаточно большого, чтобы хранить все метки, созданные визуализацией всех дочерних элементов.

- Содержимое поверхности преобразуется с использованием аффинного преобразования, задаваемого свойством RenderTransform, FixedPage (или Canvas).

- Все дочерние элементы визуализируются на поверхность, отсекаемую свойством Clip (которое также преобразуется с использованием свойства RenderTransform), FixedPage (или Canvas). FixedPage дополнительно отсекает до прямоугольника, задаваемого (0,0,PageWidth,PageHeight). Если дочерний элемент имеет свойство Opacity или свойство OpacityMask, оно применяется к дочернему элементу перед его визуализацией на поверхность.

- Наконец, содержимое FixedPage (или Canvas) визуализируется на содержащий его элемент. В случае FixedPage содержащий элемент представляет собой физическую поверхность формирования изображения.

Визуализация происходит согласно этим правилам:

- Единственными элементами, которые создают метки на поверхности, являются “Glyphs” и “Path”.

- Все другие эффекты визуализации могут достигаться расположением элементов “Glyphs” и “Path” на “Canvas” и применением их различных допустимых атрибутов.

Элементы и свойства фиксированной полезной нагрузки

Фиксированная полезная нагрузка, согласно изображенному и описанному варианту выполнения, включает в себя небольшой набор XML-элементов, используемых в разметке, для представления страниц и их содержимого. Разметка в части FixedPanel сводит страницы документа вместе в общий, легко индексируемый корень, использующий элементы <Document>, <FixedPanel> и <PageContent>. Каждая часть FixedPage представляет содержимое страницы в элементе <FixedPage> с единственными элементами <Path> и <Glyphs> (которые вместе выполняют весь рисунок) и элементом <Canvas> для их группирования.

Иерархия элемента разметки фиксированной полезной нагрузки резюмирована в следующих разделах, озаглавленных “Элементы верхнего уровня”, “Геометрическая форма для Path, Clip”, “Кисти, используемые для закрашивания Path, Glyphs или OpacityMask”, “Словари ресурсов для FixedPage или Canvas”, “Маски непрозрачности для прозрачности альфа-канала”, “Траектории отсечения” и “Преобразования”.

Элементы верхнего уровня

<Document> [явно один на часть FixedPanel]

Атрибуты:

[нет]

Дочерние элементы:

<FixedPanel> [точно один]

<FixedPanel>

Атрибуты:

PageHeight [Необязательный]

PageWidth [Необязательный]

Дочерние элементы:

<PageContent> [1-N из этих дочерних элементов]

<PageContent>

Атрибуты:

Source [требуется]

PageHeight [необязательный]

PageWidth [необязательный]

Дочерние элементы:

[нет]

<FixedPage>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

PageHeight [требуется (здесь или в качестве дочернего элемента)]

PageWidth [требуется (здесь или в качестве дочернего элемента)]

Словарь ресурсов, выраженный сам как дочерний элемент XML:

<FixedPage.Resources>

Свойства, выраженные при помощи дочерних элементов XML:
<FixedPage.PageHeight> [требуется (здесь или в качестве атрибута)]
<FixedPage.PageWidht> [требуется (здесь или в качестве атрибута)]
Содержимое при помощи дочерних элементов XML:
<Canvas>
<Path>
<Glyphs>

<Canvas>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Opacity

Свойства, выраженные при помощи ссылки словаря ресурсов:

Clip
RenderTransform
OpacityMask

Словарь ресурсов, сам выраженный как дочерний элемент XML:

<Canvas.Resources>

Свойства, выраженные при помощи дочерних элементов XML:

<Canvas.Opacity>
<Canvas.Clip>
<Canvas.RenderTransform>
<Canvas.OpacityMask>

Содержимое при помощи дочерних элементов XML:

<Canvas>
<Path>
<Glyphs>

<Path>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Opacity

Свойства, выраженные при помощи ссылки словаря ресурсов:

Clip
RenderTransform
OpacityMask
Fill

Содержимое при помощи дочерних элементов XML

<Path.Opacity>
<Path.Clip>
<Path.RenderTransform>
<Path.OpacityMask>
<Path.Fill>
<Path.Data>

<Glyphs>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Opacity
BidiLevel
FontFaceIndex
FontHintingEmSize
FontRenderingEmSize
FontUri
Indices
OriginX
OriginY
Sideways
StyleSimulations
UnicodeString

Свойства, выраженные при помощи ссылки словаря ресурсов:

Clip
RenderTransform
OpacityMask
Fill

Содержимое при помощи дочерних элементов XML:

<Glyphs.Clip>
<Glyphs.RenderTransform>
<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>

Геометрическая форма для Path, Clip

<Path.Data>
Атрибуты:
[нет]
Значение свойства, выраженное в виде единственного дочернего элемента XML:
[Path.Data имеет ровно один из этих дочерних элементов]
<GeometryCollection>
<PathGeometry>

<GeometryCollection>
Атрибуты:
CombineMode
Дочерние элементы:
[1-N-дочерних элементов]
<GeometryCollection>
<PathGeometry>

<PathGeometry>
Атрибуты:
FillRule
Дочерние элементы:
[1-N-дочерних элементов]
<PathFigure>

<PathFigure>
Атрибуты:
[нет]
Дочерние элементы:
[StartSegment поступает первым, CloseSegment - последним, 1-N из Poly* между ними]
<StartSegment>
<PolyLineSegment>
<PolyBezierSegment>
<CloseSegment>

<StartSegment>
Свойства, выраженные непосредственно при помощи простых XML-атрибутов:
Point
Свойства, выраженные при помощи дочерних элементов XML
<StartSegment.Point>

<PolyLineSegment>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Points

Свойства, выраженные при помощи дочерних элементов XML

<PolyLineSegment.Points>

<PolyBezierSegment>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Points

Свойства, выраженные при помощи дочерних элементов XML

<PolyBezierSegment.Points>

Кисти, используемые для закрашивания Path, Glyphs или OpacityMask

<Path.Fill>

Атрибуты:

[нет]

Значение свойства, выраженное в виде единственного дочернего элемента XML:

[Path.Data имеет ровно один из этих дочерних элементов]

<SolidColorBrush>

<ImageBrush>

<DrawingBrush>

<LinearGradientBrush>

<RadialGradientBrush>

<Glyphs.Fill>

Атрибуты:

[нет]

Значение свойства, выраженное в виде единственного дочернего элемента XML:

Glyphs.Fill имеет ровно один из этих дочерних элементов

<SolidColorBrush>

<ImageBrush>

<DrawingBrush>

<LinearGradientBrush>

<RadialGradientBrush>

<SolidColorBrush>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Opacity

Color

Свойства, выраженные при помощи дочерних элементов XML

<SolidColorBrush.Opacity>

<SolidColorBrush.Color>

<ImageBrush>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Opacity
HorizontalAlignment
VerticalAlignment
ViewBox
ViewPort
Stretch
TileMode
ContentUnits
ViewportUnits
ImageSource

Свойства, выраженные при помощи ссылки словаря ресурсов:

Transform

Свойства, выраженные при помощи дочерних элементов XML

<ImageBrush.Opacity>
<ImageBrush.Transform>
<ImageBrush.HorizontalAlignment>
<ImageBrush.VerticalAlignment>
<ImageBrush.ViewBox>
<ImageBrush.ViewPort>
<ImageBrush.Stretch>
<ImageBrush.TileMode>
<ImageBrush.ContentUnits>
<ImageBrush.ViewportUnits>
<ImageBrush.ImageSource>

<DrawingBrush>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Opacity
HorizontalAlignment
VerticalAlignment
ViewBox
ViewPort
Stretch
TileMode
ContentUnits
ViewportUnits

Свойства, выраженные при помощи ссылки словаря ресурсов:

Transform
Drawing

Свойства, выраженные при помощи дочерних элементов XML

<DrawingBrush.Opacity>
<DrawingBrush.Transform>
<DrawingBrush.HorizontalAlignment>
<DrawingBrush.VerticalAlignment>
<DrawingBrush.ViewBox>
<DrawingBrush.ViewPort>
<DrawingBrush.Stretch>
<DrawingBrush.TileMode>
<DrawingBrush.ContentUnits>
<DrawingBrush.ViewportUnits>
<DrawingBrush.Drawing>

<DrawingBrush.Drawing>

Содержимое при помощи дочерних элементов XML :

<Canvas>
<Path>
<Glyphs>

<LinearGradientBrush>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов :

Opacity
MappingMode
SpreadMethod
StartPoint
EndPoint

Свойства, выраженные при помощи ссылки словаря ресурсов:

Transform
GradientStops

Свойства, выраженные при помощи дочерних элементов XML

<LinearGradientBrush.Opacity>
<LinearGradientBrush.Transform>
<LinearGradientBrush.MappingMode>
<LinearGradientBrush.SpreadMethod>
<LinearGradientBrush.StartPoint>
<LinearGradientBrush.EndPoint>
<LinearGradientBrush.GradientStops>

<RadialGradientBrush>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Opacity
Center
Focus
RadiusX
RadiusY

Свойства, выраженные при помощи ссылки словаря ресурсов:

Transform
GradientStops

Свойства, выраженные при помощи дочерних элементов XML

<RadialGradientBrush.Opacity>
<RadialGradientBrush.Transform>
<RadialGradientBrush.Center>
<RadialGradientBrush.Focus>
<RadialGradientBrush.RadiusX>
<RadialGradientBrush.RadiusY>
<RadialGradientBrush.GradientStops>

<GradientStops>

Содержимое при помощи дочерних элементов XML:

<GradientStop> 1-N из этих дочерних элементов

<GradientStop>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Color
Offset

Свойства, выраженные при помощи дочерних элементов XML

<GradientStop.Color>
<GradientStop.Offset>

Словари ресурсов для FixedPage или Canvas

<FixedPage.Resources>
<Canvas.Resources>

Эти элементы описаны выше в разделе, в котором описываются Словари ресурсов.

Маски непрозрачности для прозрачности альфа-канала

<Canvas.OpacityMask>

Атрибуты:

[нет]

Значение свойства, выраженное в виде единственного дочернего элемента XML:

[Canvas.OpacityMask имеет ровно один из этих дочерних элементов]

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

<Path.OpacityMask>

Атрибуты:

[нет]

Значение свойства, выраженное в виде единственного дочернего элемента XML:
[Path.Data имеет ровно один из этих дочерних элементов]

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

<Glyphs.OpacityMask>

Атрибуты:

[нет]

Значение свойства, выраженное в виде единственного дочернего элемента XML:

[Glyphs.OpacityMask имеет ровно один из этих дочерних элементов]

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

Траектории отсечения

<Canvas.Clip>

Атрибуты:

[нет]

Значение свойства, выраженное в виде единственного дочернего элемента XML:
[Canvas.Clip имеет ровно один из этих дочерних элементов]

<GeometryCollection>
<PathGeometry>

<Path.Clip>

Атрибуты:

[нет]

Значение свойства, выраженное в виде единственного дочернего элемента XML:

[Path.Data имеет ровно один из этих дочерних элементов]

<GeometryCollection>
<PathGeometry>

<Glyphs.Clip>

Атрибуты:

[нет]

Значение свойства, выраженное в виде единственного дочернего элемента XML:

[Glyphs.Clip имеет ровно один из этих дочерних элементов]

<GeometryCollection>
<PathGeometry>

Преобразования

<Canvas.RenderTransform>

Значение свойства, выраженное в виде единственного дочернего элемента XML:

<MatrixTransform> [требуется]

<Path.RenderTransform>

Значение свойства, выраженное в виде единственного дочернего элемента XML:

<MatrixTransform> [требуется]

<Glyphs.RenderTransform>

Значение свойства, выраженное в виде единственного дочернего элемента XML:

<MatrixTransform> [требуется]

<MatrixTransform>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов:

Matrix

Свойства, выраженные при помощи дочерних элементов XML

<MatrixTransform.Matrix>

<ImageBrush.Transform>

Свойства, выраженные непосредственно при помощи простых XML-атрибутов

MatrixTransform

Свойства, выраженные при помощи дочерних элементов XML :

<ImageBrush.Transform.MatrixTransform>

<DrawingBrush.Transform>
Свойства, выраженные непосредственно при помощи простых XML-атрибутов:
MatrixTransform
Свойства, выраженные при помощи дочерних элементов XML
<DrawingBrush.Transform.MatrixTransform>
<LinearGradientBrush.Transform>
Свойства, выраженные непосредственно при помощи простых XML-атрибутов:
MatrixTransform
Свойства, выраженные при помощи дочерних элементов XML
<LinearGradientBrush.Transform.MatrixTransform>
<RadialGradientBrush.Transform>
Свойства, выраженные непосредственно при помощи простых XML-атрибутов:
MatrixTransform
Свойства, выраженные при помощи дочерних элементов XML
< RadialGradientBrush.Transform.MatrixTransform>

Разметка FixedPage

Каждая часть FixedPage представляет содержимое страницы в разметке XML, расположенной в корне в элементе <FixedPage>. Эта разметка FixedPage обеспечивает точность воспроизведения WYSIWYG, ("что видишь, то и получаешь") документа между составителями и считывателями при помощи только небольшого набора элементов и свойств: элементы <Path> и <Glyphs> (которые вместе выполняют весь рисунок) и элемент <Canvas> для их группирования.

Свойства общих элементов

Перед описанием атрибутов, характерных для каждого элемента в разметке FixedPage, рассмотрим атрибуты, общие для элементов рисования и группирования: Opacity, Clip, RenderTransform и OpacityMask. Не только они являются единственными свойствами, общими для элементов верхнего уровня, они также являются единственными свойствами, которые "накапливают" свои результаты от родительского элемента к дочернему элементу, как описано в разделе "Правила композиции" выше. Накопление представляет собой результат применения Правил композиции. В таблице, которая следует ниже, представлено краткое описание этих общих атрибутов, за которой следует более подробное описание каждого атрибута.

Атрибут	Элементы	Описание
Opacity	Canvas, Path, Glyphs and SolidColorBrush, ImageBrush, DrawingBrush, LinearGradientBrush, RadialGradientBrush	Определяет равномерную прозрачность элемента

Дочерний элемент	Элементы	Описание
Clip	Canvas, Path, Glyphs	Определяет равномерную прозрачность элемента
RenderTransform	Canvas, Path, Glyphs	RenderTransform устанавливает для дочернего элемента новый координатный фрейм элемента. Поддерживается только MatrixTransform
OpacityMask	Canvas, Path, Glyphs	Задает прямоугольную маску альфа-факторов, которая применяется так же, как и атрибут Opacity, но допускает различные альфа-факторы на попиксельной основе

Атрибут Opacity

Opacity используется для прозрачного сопряжения двух элементов при визуализации (Alpha Blending). Атрибут Opacity заключен в пределах от 0 (полностью прозрачный) до 1 (полностью непрозрачный). Значения вне этого охватывающего диапазона ограничиваются этим диапазоном во время синтаксического анализа разметки. Поэтому фактически [-∞ … 0] является прозрачным, и [1 … ∞] является

непрозрачным.

Атрибут Opacity применяется при помощи следующих вычислений (предполагая исходный цвет и цвет назначения без умножения в обратном порядке, оба задаются как scRGB):

O_E: Атрибут Opacity элемента или альфа-фактора в соответствующем положении в OpacityMask

A_S: Альфа-фактор, присутствующий на поверхности источника

R_S: Значение Red, присутствующее на поверхности источника

G_S: Значение Green, присутствующее на поверхности источника

B_S: Значение Blue, присутствующее на поверхности источника

A_D: Альфа-фактор, уже присутствующий на поверхности назначения

R_D: Значение Red, уже присутствующее на поверхности назначения

G_D: Значение Green, уже присутствующее на поверхности назначения

B_D: Значение Blue, уже присутствующее на поверхности назначения

A*: Результирующее значение Alpha для поверхности назначения

R*: Результирующее значение Red для поверхности назначения

G*: Результирующее значение Green для поверхности назначения

B*: Результирующее значение Blue для поверхности назначения

Все значения, обозначаемые с нижним индексом T, являются временными значениями (например, R_{T1}).

Этап 1: умножить альфа-фактор источника на значение Opacity

$A_s=A_s \cdot O_E$

Этап 2: умножить в обратном порядке альфа-фактор источника

$A_{T1}=A_S$

$R_{T1}=R_S \cdot A_S$

$G_{T1}=G_S \cdot A_s$

$B_{T1}=B_s \cdot A_s$

Этп 3: умножить в обратном порядке альфа-фактор назначения

$A_{T2} = A_D$

$R_{T2} = R_D \cdot A_D$

$G_{T2} = G_D \cdot A_D$

$B_{T2} = B_D \cdot A_D$

Этап 3: сопряжение

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

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

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

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

Этап 4: обратное умножение в обратном порядке

Если $A_{T2} = 0$, установить все A*R*G*D* к 0.

Иначе:

$A^* = A_{T2}$

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

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

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

Свойство Clip

Свойство Clip задается в качестве одного из геометрических элементов <GeometryCollection> или <PathGeovetry> (см. Path.Data в отношении подробностей).

Свойство Clip применяется следующим образом:

Является видимым все визуализируемое содержимое, которое попадает внутрь геометрического элемента, описываемого дочерним элементом Clip.

Не является видимым все визуализируемое содержимое, которое оказывается снаружи геометрического элемента, описываемого дочерним элементом Clip.

Дочерний элемент RenderTransform

MatrixTransform является единственным атрибутом преобразования, доступным для элементов. Он выражает аффинное преобразование. Синтаксис следующий:

<X.RenderTransform>

<MatrixTransform Matrix="1,0,0,1,0,0"/>

</X.RenderTransform>

X представляет элемент, к которому применяется преобразование.

Шесть чисел, заданных в атрибуте Matrix, представляют собой m00, m01, m10, m11, dx, dy.

Полная матрица выглядит как:

m00	m01	0
m10	m11	0
dx	dy	1

Заданная координата X,Y преобразуется при помощи RenderTransform, получая результирующую координату X',Y', при помощи применения этих вычислений:

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

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

Дочерний элемент OpacityMask

OpacityMask задает Brush, но в противоположность Fill Brush только альфа-канал (см. выше атрибут Opacity) кисти используется в качестве дополнительного параметра для визуализации элемента. Каждый альфа-фактор для каждого пиксела элемента затем дополнительно умножается на альфа-фактор в соответствующем положении в OpacityMask Brush.

Элемент <Canvas>

Элемент <Canvas> используется для группирования вместе элементов. Обычно элементы FixedPage группируются вместе в <Canvas>, когда они совместно используют составной общий атрибут (т.е. Opacity, Clip, RenderTransform или OpacityMask). В результате группирования вместе этих элементов на Canvas общие атрибуты часто могут применяться к Canvas вместо отдельных элементов.

Атрибуты и дочерние элементы <Canvas>

Элемент <Canvas> имеет только общие атрибуты, описанные ранее: Opacity, Clip, RenderTransform и OpacityMask. Они используются с элементом <Canvas>, как описано в таблице ниже:

Атрибут	Воздействие на Canvas
Opacity	Определяет равномерную прозрачность Canvas

Дочерний элемент	Воздействие на Canvas

Clip	Clip описывает регион, в котором кисть может применяться дочерними элементами Canvas
RenderTransform	RenderTransform устанавливает новый координатный фрейм для дочерних элементов Canvas, например другой Canvas. Поддерживается только MatrixTransform
OpacityMask	Задает прямоугольную маску альфа-факторов, которые применяются так же, как и атрибут Opacity, но допускает другой альфа-фактор на попиксельной основе

Следующий пример разметки иллюстрирует использование <Canvas>.

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

В отношении порядка считывания в разметке Canvas рассмотрим следующее. Как и с FixedPage, порядок разметки дочерних элементов Glyphs, содержащихся в Canvas, должен быть тем же, что и требуемый порядок считывания текстового содержимого. Этот порядок считывания может использоваться как для интерактивного выбора/копирования последовательного текста из FixedPage в программе просмотра, так и для обеспечения возможности доступа к последовательному тексту технологией доступности. Ответственностью приложения, генерирующего разметку FixedPage, является обеспечение этого соответствия между порядком разметки и порядком считывания.

Дочерние элементы Glyphs, содержащиеся во вложенных элементах Canvas, упорядочиваются внутри среди сестринских элементов Glyphs, имеющих место перед и после Canvas.

Пример

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

Элемент <Path>

Элемент Path представляет собой основанный на XML элемент, который описывает геометрический регион. Геометрический регион представляет собой форму, которая может закрашиваться или использоваться в качестве траектории отсечения. Общие типы геометрических форм, такие как прямоугольник и эллипс, могут быть представлены с использованием геометрических форм Path. Траектория описывается посредством задания требуемого дочернего элемента Geometry.Data и атрибутов визуализации, таких как Fill или Opacity.

Свойства и дочерние элементы <Path>

Следующие свойства применимы к элементам <Path>, как описано ниже:

Свойства	Воздействие на Path
Opacity	Определяет равномерную прозрачность закрашиваемой траектории

Дочерний элемент	Воздействие на Path
Clip	Clip описывает регион, в котором кисть может применяться посредством геометрической формы траектории
RenderTransform	RenderTransform устанавливает новый координатный фрейм для дочерних элементов траектории, такой как геометрическая форма, определяемая посредством Path.Data. Поддерживается только MatrixTransform
OpacityMask	Задает прямоугольную маску альфа-факторов, которая применяется так же, как и атрибут Opacity, но допускает другой альфа-фактор для различных областей поверхности
Data	Описывает геометрическую форму траектории.
Fill	Описывает кисть, используемую для закрашивания геометрической формы траектории

Чтобы описать то, как закрашивать регион, описываемый геометрической формой дочернего элемента <Path.Data>, следует использовать свойство Fill. Чтобы ограничить регион, на котором могут рисоваться формы <Path.Data>, следует использовать свойство Clip.

Использование <Path> для описания геометрических форм

Геометрическая форма траектории задается в виде последовательности вложенных дочерних элементов <Path.Data>, как показано ниже. Геометрическая форма может быть представлена при помощи или <GeometryCollection>, содержащей набор дочерних элементов <PathGeometry>, или единственного дочернего элемента <PathGeometry>, содержащего <PathFigure>.

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

Одинаковые элементы <GeometryCollection> или <PathGeometry> определяют геометрическую форму для траектории отсечения, используемой в свойстве Clip в Canvas, Path или Glyphs.

В нижеследующей таблице введена иерархия дочерних элементов, определяющих геометрические формы Path.

Элементы Geometry	Описание
GeometryCollection	Набор элементов PathGeometry, визуализируемых с использованием операций Boolean CombineMode
PathGeometry	Набор элементов PathFigure, каждый из которых закрашивается с использованием одного и того же варианта FillRule
PathFigure	Набор одного или нескольких элементов сегмента
StartSegment, PolyLineSegment PolyBezierSegment CloseSegment	

GeometryCollection

GeometryCollection представляет собой набор геометрических объектов, которые скомбинированы вместе для визуализации согласно вариантам Boolean CombineMode. Элемент GeometryCollection представляет собой механизм разметки FixedPage для создания визуальных комбинаций геометрических форм.

Атрибуты	Воздействие на GeometryCollection
CombineMode	Задает различные режимы для комбинирования геометрических форм

Атрибут CombineMode задает операцию Boolean, используемую для комбинирования набора геометрических форм в GeometryCollection. В зависимости от режима будут включаться или исключаться различные регионы.

Варианты CombineMode	Описание
Complement	Задает, что существующий регион заменяется результатом удаления существующего региона из нового региона. Другими словами, существующий регион исключается из нового региона.
Exclude	Задает, что существующий регион заменяется результатом удаления нового региона из существующего региона. Другими словами, новый регион исключается из существующего региона
Intersect	Два региона комбинируются посредством взятия их пересечения
Union	Два региона комбинируются посредством взятия объединения их обоих
Xor	Два региона комбинируются посредством взятия только областей, охватываемых одним или другим регионом, но не обеими

CombineMode обрабатываются следующим образом:

NotCommutative Complement и Exclude не обладают свойством коммутативности и поэтому определяются среди первой геометрической формы в GeometryCollection и каждой отдельной оставшейся геометрической формой. Например, для набора { g1, g2, g3 } CombineMode Exclude применялся бы как ((g1 исключая g2) и (g1 исключая g3)).

Commutative Операции Boolean Union, Xor, Intersect обладают свойством коммутативности и поэтому применяют порядок, независимый от геометрических форм.

PathGeometry

Элемент PathGeometry содержит набор элементов PathFigure. Объединение PathFigure определяет внутреннюю область PathGeometry.

Атрибуты	Воздействие на GeometryCollection
FillRule	Задает альтернативные алгоритмы для закрашивания траекторий, которые описывают охватываемые области

В отношении атрибута FillRule рассмотрим следующее. Закрашиваемая область PathGeometry определяется принятием всех содержащихся PathFigure, атрибут Filled которых установлен в true, и применением FillRule для определения охватываемой области. Варианты FillRule задают как пересекающиеся области элементов Figure, содержащихся в Geometry, комбинируются, образуя результирующую область Geometry.

Согласно описанному варианту выполнения предусмотрены алгоритмы EvenOdd Fill и NonZero Fill.

Алгоритм EvenOdd Fill определяет “внутренность” точки на полотне посредством рисования лучевой линии из этой точки в бесконечность в любом направлении и затем исследования мест, где сегмент формы пересекает лучевую линию. Начиная подсчет с нуля, добавляют один всякий раз, когда Segment пересекает лучевую линию слева направо, и вычитают один всякий раз, когда сегмент траектории пересекает лучевую линию справа налево. После подсчета пересечений, если результат равен нулю, то тогда точка находится *снаружи* траектории. Иначе она находится *внутри*.

Алгоритм NonZero Fill определяет “внутренность” точки на полотне посредством рисования лучевой линии из этой точки в бесконечность в любом направлении и подсчета количества Segment траектории из заданной формы, которые пересекает лучевая линия. Если это количество нечетное, точка находится внутри, если четное, точка находится снаружи.

PathFigure

Элемент PathFigure состоит из набора одного или нескольких сегментов линий или кривых. Элементы сегмента определяют форму PathFigure. PathFigure всегда должен определять замкнутую форму.

Атрибуты	Воздействие на PathFigure
FillRule	Задает альтернативные алгоритмы для закрашивания траекторий, которые описывают охватываемую область

Фигура требует начальную точку, после которой каждый сегмент линии или кривой продолжается из последней добавленной точки. Первым сегментом в наборе PathFigure должен быть StartSegment, и CloseSegment должен быть последним сегментом. StartSegment имеет атрибут Point. CloseSegment не имеет атрибутов.

Атрибут StartSegment	Описание
Point	Расположение сегмента линии (начальная точка)

Разметка фиксированной полезной нагрузки для геометрических форм Path.Data

Нижеследующее обеспечивает разметку для рисования и закрашивания Path на Canvas. В конкретном примере ниже рисуется прямоугольная Path на Canvas и закрашивается кистью чистого зеленого цвета.

```
<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>
```

Нижеследующая разметка описывает рисование кубической кривой Безье. Т.е. в дополнение к PolyLineSegment разметка Fixed-Payload включает в себя PolyBezierSegment для рисования кубических кривых Безье.

```
<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>
```

Кисти

Кисть используется для заполнения краской внутренней части геометрических форм, определяемых элементом <Path>, и закрашивания растров символов, визуализируемых при помощи элемента <Glyphs>. Кисть также используется при определении маски прозрачности альфа-канала в <Canvas.OpacityMask>, <Path.OpacityMask> и <Glyphs.OpacityMask>. Разметка FixedPage включает в себя следующие кисти:

Тип кисти	Описание
SolidColorBrush	Закрашивает определенные геометрические регионы чистым цветом
ImageBrush	Закрашивает регион изображением
DrawingBrush	Закрашивает регион векторным рисунком
LinearGradientBrush	Закрашивает регион линейным градиентом
RadialGradientBrush	Закрашивает регион радиальным градиентом

Атрибуты изменяются в зависимости от кистей, хотя все кисти имеют атрибут Opacity. ImageBrush и DrawingBrush совместно используют возможности мозаичного размещения. Две кисти с градиентным закрашиванием также имеют общие атрибуты.

Использование дочернего элемента кисти в разметке показано ниже:

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

Общие свойства для кистей

Согласно описанному варианту выполнения следующие свойства применимы ко всем кистям, кроме простой кисти SolidColorBrush, которая имеет меньшее количество необязательных дочених элементов.

Атрибут	Тип кисти	Описание
Opacity	Все кисти	

Дочерний элемент	Тип кисти	Описание
Transform	Все кисти за исключением SolidColorBrush	Описывает MatrixTransform, применимый к координатному пространству кисти

Общие атрибуты для DrawingBrush и ImageBrush

HorizontalAlignment	DrawingBrush, ImageBrusch	Center, Left или Right
VerticalAlignment	DrawingBrush, ImageBrusch	Center, Bottom или Top
ViewBox	DrawingBrush, ImageBrush	
ViewPort	DrawingBrush, ImageBrusch	
Stretch	DrawingBrush, ImageBrusch	None, Fill, Uniform или UniformToFill
TileMode	DrawingBrusch, ImageBrush	None, Tile, FlipY, FlipX или FlipXY
ContentUnits	DrawingBrush, ImageBrusch	Absolute или RelativeToBoundingBox
ViewportUnits	DrawingBrush, ImageBrusch	Absolute или RelativeToBoundingBox

Атрибут Horizontal Alignment задает то, как кисть выравнивается в горизонтальном направлении в области, которую она закрашивает. Атрибут Vertical Alignment задает то, как кисть выравнивается в вертикальном направлении внутри области, которую она закрашивает. Атрибут ViewBox имеет значение по умолчанию (0,0,0,0), интерпретируемое как неустановленное. Когда оно является неустановленным, не выполняется корректировка, и атрибут Stretch игнорируется. Окно просмотра задает новую систему координат для содержимого, т.е. повторно определяет протяженность и начало порта просмотра. Атрибут Stretch способствует заданию того, как это содержимое отображается в порт просмотра. Значение атрибута viewBox представляет собой список “безразмерных” четырех чисел <min-x>, <min-y>, <width> и <height>, разделенных пробельным символом и/или запятой, и является типа Rect. Viewbox rect

задает прямоугольник в пользовательском пространстве, которое отображается в ограничивающий прямоугольник. Он работает аналогично введению scaleX и scaleY. Атрибут Stretch (в случае, если вариантом не является none) обеспечивает дополнительное управление для сохранения соотношения размеров графики. Дополнительное преобразование применяется ко всем потомкам данного элемента для достижения заданного эффекта. Если есть преобразование на Brush, оно применяется “над” отображением в ViewBox.

Атрибут Stretch имеет следующие режимы: None, Fill, Uniform, UniformToFill.

Вариант атрибута Stretch	Описание
None	По умолчанию. Сохраняет исходные размеры
Fill	Соотношение размеров не сохраняется, и содержимое масштабируется так, чтобы закрасить установленные границы
Uniform	Равномерно масштабирует размеры до тех пор, пока изображение не будет помещаться в установленные границы
UniformToFill	Равномерно масштабирует размеры, чтобы закрасить установленные границы и выполнить отсечение, если необходимо

Простые кисти и их атрибуты

Дочерние элементы Path.Brush и Canvas.Brush включают в себя следующие: SolidColorBrush, ImageBrush и DrawingBrush.

SolidColorBrush закрашивает определенные геометрические регионы чистым цветом. Если имеется альфа компонент цвета, он комбинируется мультипликативным образом с соответствующим атрибутом непрозрачности в Brush.

Атрибуты	Воздействие
Color	Задает цвет для закрашиваемых элементов

Нижеследующий пример иллюстрирует, как атрибуты цвета выражаются для SolidColorBrush.

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

ImageBrush может использоваться для закрашивания пространства изображением. Разметка для ImageBrush дает возможность задавать URI. Если все другие атрибуты установлены в их значения по умолчанию, изображение будет растягиваться для закрашивания ограничивающего прямоугольника региона.

Атрибуты	Воздействие
ImageSource	Задает URI ресурса изображения

Атрибут ImageSource должен ссылаться или на один из поддерживаемых Форматов расширенного изображения, или на селектор, который приводит к изображению одного из этих типов.

DrawingBrush может использоваться для закрашивания пространства векторным рисунком. DrawingBrush имеет Дочерний элемент рисования, использование которого в разметке показано ниже.

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

Градиентные кисти и их атрибуты

Градиенты рисуются посредством задания набора ограничителей градиента в качестве Дочерних элементов XML градиентных кистей. Эти ограничители градиента задают цвета вдоль прогрессии некоторого вида. Существует два типа градиентных кистей, поддерживаемых в данной инфраструктуре: линейные и радиальные.

Градиентные рисуются посредством интерполирования между ограничителями градиента в заданном цветовом пространстве. LinearGradientBrush и GradientBrush совместно используют следующие общие атрибуты:

Атрибут	Описание
SpreadMethod	Это свойство описывает, как кисть должна закрашивать область содержимого вне основной исходной области градиента. Значением по умолчанию является Pad
MappingMode	Это свойство определяет, интерпретируются ли параметры, описывающие градиент, относительно ограничивающего прямоугольника объекта. Значением по умолчанию является относительно-ограничающего-прямоугольника.

Дочерний элемент	Описание
GradientStops	Хранит упорядоченную последовательность элементов GradientStop

В отношении атрибута SpreadMethod рассмотрим следующее.

Варианты SpreadMethod задают, как закрашивается пространство. Значением по умолчанию является Pad.

Варианты атрибута SpreadMethod	Воздействие на Gradient
Pad	Первый цвет и последний цвет используются для закрашивания остального пространства в начале и конце соответственно.
Reflect	Ограничители градиента циклически воспроизводятся в обратном порядке для закрашивания пространства.
Repeat	Ограничители градиента повторяются по порядку до тех пор, пока пространство не будет закрашено.

Атрибут MappingMode

В отношении LinearGradientBrush рассмотрим следующее. LinearGradientBrush задает кисть линейного градиента вдоль вектора.

Атрибут	Описание
EndPoint	Конечная точка линейного градиента. LinearGradientBrush интерполирует цвета с StartPoint до EndPoint, где StartPoint представляет смещение 0, и EndPoint представляет смещение 1. По умолчанию - 1,1
StartPoint	Начальная точка линейного градиента

Следующий пример разметки показывает использовние LinearGradientBrush.

Страница с прямоугольной траекторией закрашивается линейным градиентом.

```
<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>
```

Данный пример показывает страницу с прямоугольной траекторией, которая закрашивается линейным градиентом. Path также имеет свойство отсечения в форме восьмиугольника, который отсекает его.

```
<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 аналогичен в модели программирования линейному градиенту. Однако тогда как линейный градиент имеет начальную и конечную точку для определения вектора-градиента, радиальный градиент имеет круг с фокальной точкой для определения характера изменения градиента. Круг определяет конечную точку градиента, другими словами, ограничитель градиента в 1,0 определяет цвет на окружности круга. Фокальная точка определяет центр градиента. Ограничитель градиента в 0,0 определяет цвет в фокальной точке.

Атрибут	Описание
Center	Центральная точка этого радиального градиента. RadialGradientBrush интерполирует цвета от Focus до окружности эллипса. Окружность определяется посредством Center и радиусами. По умолчанию - 0,5, 0,5
Focus	Фокус радиального градиента
RadiusX	Радиус по направлению X эллипса, который определяет радиальный градиент. По умолчанию - 0,5
RadiusY	Радиус по направлению Y эллипса, который определяет радиальный градиент. По умолчанию - 0,5
FillGradient	Pad, Reflect, Repeat

Альфа-фактор и прозрачность

Согласно изображенному и описанному варианту выполнения каждый пиксел каждого элемента имеет альфа-фактор, находящийся в диапазоне от 0,0 (полностью прозрачный) до 1,0 (полностью непрозрачный). Альфа-фактор используется при сопряжении элементов для достижения визуального эффекта прозрачности.

Каждый элемент может иметь атрибут Opacity, на который будет равномерно умножаться альфа-фактор каждого пиксела.

Кроме того, OpacityMask дает возможность получить спецификацию попиксельной непрозрачности, которая управляет тем, как визуализируемое содержимое будет плавно переходить в его состояние назначения. Непрозрачность, задаваемая OpacityMask, комбинируется мультипликативно с любой непрозрачностью, которая уже может оказаться присутствующей в альфа-канале содержимого. Попиксельная Opacity, заданная при помощи OpacityMask, определяется с учетом альфа-канала для каждого пиксела в маске - данные цвета игнорируются.

Типом OpacityMask является Brush. Это дает возможность получить спецификацию

того, как содержимое Brush отображается на экстент содержимого множеством различных путей. Как и в случае использования для закрашивания геометрической формы, Brush по умолчанию закрашивают все пространство содержимого, растягивая или дублируя его содержимое соответствующим образом. Это означает, что ImageBrush будет растягивать свой ImageSource до полного закрытия содержимого, GradientBrush будет распространяться от кромки до кромки.

Необходимые вычисления для альфа-сопряжения описываются выше в разделе “Атрибут Opacity”.

Следующий пример иллюстрирует, как OpacityMask используется для создания “эффекта вытеснения затемнением” на элементе Glyphs. OpacityMask в примере представляет собой линейный градиент, который вытесняется затемнением из непрозрачного черного в прозрачный черный.

```
//    /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>
```

Изображения в Расширенных документах

На *FixedPage* изображения закрашивают охватываемые регионы. Чтобы поместить изображение на *FixedPage*, регион сначала должен быть задан на странице. Регион определяется геометрической формой элемента Path.

Свойство Fill элемента Path задает содержимое закрашивания для описываемого региона. Изображения представляют собой один тип закрашивания, рисуемого в регионе при помощи ImageBrush. Все кисти имеют поведение по умолчанию, согласно которому закрашивается весь регион или растягиванием, или повторением (мозаичным размещением) соответствующим образом содержимого кисти. В случае ImageBrush содержимое, задаваемое свойством ImageSource, растягивается так, чтобы полностью покрыть регион.

Разметка ниже демонстрирует, как разместить изображение на Canvas.

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

Так как многие изображения являются прямоугольными, включая прямоугольный элемент Path в Словаре ресурсов, может быть полезным упростить разметку. Path

тогда может располагаться с использованием атрибута RenderTransform (см. выше).

```
<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>
```

Цвет

Цвета могут задаваться в изображенной и описанной разметке с использованием обозначения scRGB или sRGB. Спецификация scRGB известна как "IEC 61966-2-2 scRGB" и может быть получена с www.iec.ch.

Параметры ARGB описаны в таблице ниже.

Имя	Описание
R	Красный компонент scRGB текущего цвета
G	Зеленый компонент scRGB текущего цвета
B	Синий компонент scRGB текущего цвета
A	Альфа компонент scRGB текущего цвета

Отображение цветов

В данном случае рассматривается тегирование (сопровождение тегами) цветных элементов метаданными, задающими цветное содержимое. Такие метаданные могут содержать цветовой профиль ICC (Международный консорциум по средствам обработки цветных изображений) или другие данные определения цвета.

Элемент <Glyphs>

Текст представляется в Фиксированных полезных нагрузках с использованием элемента Glyphs. Этот элемент разработан так, чтобы удовлетворять требованиям на печать и на расширенные документы.

Элементы Glyphs могут иметь комбинации следующих свойств.

Свойство	Назначение	Представление разметки (элемент Glyphs)
Origin	Начало координат первого глифа в серии. Глиф размещается так, что передняя кромка его вектора продвижения и его базовая линия пересекают эту точку	Задается при помощи свойств OriginX и OriginY
FontRenderingEmSize	Размер шрифта в единицах поверхности рисования (по умолчанию 1/96 часть дюйма)	Измеряется в единицах Length
FontHintingEmSize	Размер для установки хинтов в пунктах. Шрифты могут включать в себя установку хинтов для получения незначительных различий при разных размерах, например более толстые основные штрихи и более открытый полуовал при меньших размерах, чтобы получить в результате то, что они выглядят более похожими на один и тот же стиль, чем может обеспечить простое масштабирование. Это не одно и то же, что установка хинтов для пиксельного разрешения устройства, которое обрабатывается автоматически. К настоящему времени (март 2003 г.) известные шрифты не включают в себя установку хинтов в зависимости от размера. Значение по умолчанию - 12 пунктов	Измеряется в числах с двойной точностью, представляющих размер шрифта в пунктах

GlyphIndices	Массив из 16-разрядных номеров глифов, которые представляют эту серию	Часть свойства Indices. См. ниже для представления
AdvanceWidths	Массив полной ширины, один для каждого глифа в GlyphIndices. Номинальное начало координат n-го глифа в серии (n>0) представляет собой номинальное начало координат n-1 глифа плюс n-1 полная ширина, добавленная вдоль вектора продвижения серий. Базовые глифы, в основном, имеют ненулевую полную ширину, комбинирующие глифы, в основном, имеют нулевую полную ширину	Часть свойства Indices. См. ниже для представления
GlyphOffsets	Массив смещений глифов. Добавляется к номинальному началу координат глифа, вычисленному выше, для генерирования окончательного начала координат для глифа. Базовые глифы, в основном. имеют смещение глифа (0,0), комбинирующие глифы, в основном, имеют смещение, которое размещает их правильно на верхней части ближайшего предшествующего базового глифа	Часть свойства Indices. См. ниже для представления
GlyphTypeface	Физический шрифт, из которого рисуются все глифы в этой серии	Свойства FontUri, FontFaceIndex и StyleSimulations
UnicodeString	Необязательный* Массив символов, представленный этой серией глифов. * Заметьте, что для GlyphRun, генерируемых из драйверов принтеров Win32, текст, который изначально печатался при помощи вызовов Win32 ExtTextOut(ETO_GLYPHINDEX), передается на драйвер с индексами глифов и без кодовых значений уникода. В данном случае сгенерированная разметка Glyphs и, таким образом, созданный объект GlyphRun не будут включать в себя кодовые значения. Без кодовых значений недоступна функциональная возможность, такая как вырезание и вставка или поиск в программе просмотра фиксированных форматов, однако отображение текста остается возможным	
ClusterMap	Один ввод на символ в UnicodeString. Каждое значение дает смещение первого глифа в GlyphIndices, который представляет соответствующий символ в UnicodeString. Где многочисленные символы отображаются на один глиф, или где один символ отображается на многочисленные глифы, или где многочисленные символы отображаются на многочисленные глифы неразделимо, символ или символ(ы) и глиф или глиф(ы) называются кластером. Все вводы в ClusterMap для многосимвольного кластера отображаются на смещение в массиве GlyphIndices первого глифа кластера	
Sideways	Раскладка глифов происходит по их боковой стороне. По умолчанию глифы визуализируются, как если бы они были в виде горизонтального текста, причем начало координат соответствует началу координат базовой линии для западных языков. При установленном флаге “боком” глиф поворачивается на свою боковую сторону, причем началом координат становится верхний центр неповернутого глифа	Да
BidiLevel	Уровень двунаправленного вложения алгоритма уникода Численно четные значения означают раскладку слева направо, численно нечетные значения означают раскладку справа налево. Раскладка справа налево размещает начало координат серии на правой стороне первого глифа, причем положительные значения вектора продвижения размещают последующие глифы влево от предыдущего глифа	Да
Brush	Кисть переднего плана, используемая для рисования глифов	Выбирается из свойства Shape Fill.
Language	Язык серии, обычно поступает из свойства xml:lang разметки	Задается свойством xml:lang

Обзор разметки текста

Метрики глифа

Каждый глиф определяет метрики, которые задают, как он выравнивается с другими глифами. Примерные метрики согласно одному варианту выполнения показаны на фиг.12.

Полная ширина и комбинирующие знаки

Вообще говоря, глифы в шрифте являются или базовыми глифами, или комбинирующими знаками, которые могут быть присоединены к базовым глифам. Базовые глифы обычно имеют полную ширину, которая не равна нулю, и вектор смещения 0,0. Комбинирующие знаки обычно имеют нулевую полную ширину. Вектор

смеения может использоваться для коррекции положения комбинирующего знака и поэтому может иметь значение не 0,0 для комбинирующих знаков.

Каждый глиф в серии глифов имеет три значения, управляющие его положением. Значения указывают начало координат, полную ширину и смещение глифа, каждое из которых описывается ниже.

Начало координат: Как предполагается, каждому глифу предоставляется номинальное начало координат, для первого глифа в серии это начало координат серии.

Полная ширина: Полная ширина для каждого глифа обеспечивает начало координат следующего глифа относительно этого начала координат глифов. Вектор продвижения всегда проводится в направлении прогрессии серии.

Смещение глифа (базового или знака): Вектор смещения глифа корректирует это положение глифа относительно его номинального начала координат.

Символы, глифы и отображение кластера

Отображения кластера содержат один ввод на кодовое значение уникода. Значение в вводе представляет собой смещение первого глифа в массиве GlyphIndices, которое представляет это кодовое значение. Альтернативно, где кодовое значение представляет собой часть группы кодовых значений, представляющих неразделимый кластер символов, первый глиф в массиве GlyphIndices представляет смещение первого глифа, который представляет этот кластер.

Отображения кластера

Отображение кластера может представлять отображения кодового значения в глиф, которые могут быть “один в один”, “многие в один”, “один в многие” или “многие в многие”. Отображения “один в один” имеют место, когда каждое кодовое значение представляется точно одним глифом, вводами отображения кластера на фиг.13 являются 0, 1, 2 ….

Отображения “многие в один” имеют место, когда два или более кодовых значения отображаются в единственный глиф. Вводы для этих кодовых значений задают смещение этого глифа в буфере индексов глифа. В примере по фиг.14 символы 'f' и 'i' были заменены лигатурой, что является общей практикой набора текста во многих шрифтах с засечками.

В отношении отображений “один в многие” рассмотрим следующее в отношении фиг.15. 'Sara Am' содержит часть, которая располагается поверх предыдущего базового символа (кольцо), и часть, которая располагается справа от базового символа (крючок). Когда текст на тайском языке выравнивается раздвижкой символов, крючок относится от базового символа, тогда как кольцо остается поверх базового символа, поэтому многие шрифты кодируют кольцо и крючок как отдельные глифы. Когда одно кодовое значение отображается в два или более глифов, значение в *ClusterMap* для этого кодового значения ссылается на первый глиф в массиве *GlyphIndeces*, который представляет это кодовое значение.

В отношении отображений “многие в многие” рассмотрим следующее в связи с фиг.16. В некоторых шрифтах неразделимая группа кодовых значений для кластера символов отображается в более чем один глиф. Например, это распространено в шрифтах, поддерживающих рукописные шрифты, относящиеся к индийской группе индоевропейских языков. Когда неразделимая группа кодовых значений отображается в один или несколько глифов, значение в *ClusterMap* для каждого из кодовых значений ссылается на первый глиф в массиве *GlyphIndeces*, который представляет это кодовое значение.

Следующий пример показывает уникод и представление глифа тайского слова □□□□. Первые два кодовые значения комбинируются для генерирования трех глифов.

Задание кластеров

Спецификации кластера предшествуют спецификации глифа для первого глифа кластера не 1:1 (отображения более сложные, чем один символ в один глиф).

Каждая спецификация кластера имеет следующий вид:

(ClusterCodepointCount [:ClusterGlyphCount])

Часть спецификации кластера	Тип	Назначение	Значение по умолчанию
ClusterCodepointCount	Положительное целое число	Номера 16-разрядных кодовых значений уникода комбинируются для образования этого кластера	1
ClusterGlyphCount	Положительное целое число	Номера 16-разрядных индексов глифов комбинируются для образования этого кластера	1

Разметка <Glyphs>

Элемент Glyphs задает шрифт как URI, индекс начертания и набор других атрибутов, описанных выше. Например:

```
<Glyphs
    FontUri             = "file://c:/windows/fonts/times.ttf"
    FontFaceIndex       = "0"              <!-- по умолчанию 0 -->
    FontRenderingEmSize = "20"             <!-- нет значений по умолчанию -->
    FontHintingEmSize   = "12"             <!-- по умолчанию 12 -->
    StyleSimulations    = "BoldSimulation" <!-- по умолчанию None -->
    Sideways            = "false"          <!-- по умолчанию false -->
    BidiLevel           = "0"              <!-- по умолчанию 0 -->
    Unicode             = " ... "          <!-- представление в уникоде -->
    Indices             = " ... "          <!-- см. ниже -->
    остальные атрибуты    ...
/>
```

Каждая спецификация глифа имеет следующий вид:

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

Каждая часть спецификации глифа является необязательной:

Часть спецификации глифа	Назначение	Значение по умолчанию
GlyphIndex	Индекс глифа при визуализации физического шрифта	Определяется таблицей отображения символов шрифтов для соответствующего кодового значения уникода во внутреннем тексте
Advance	Размещение следующего глифа относительно начала координат данного глифа. Измеряется в направлении продвижения, определяемого боковыми атрибутами и атрибутами BidiLevel. Измеряется в сотых долях от размера в em (буква m, напечатанная шрифтом долях от размера в em (буква m, напечатанная шрифтом цицеро) шрифта. Продвижение должно вычисляться, так чтобы ошибки округления не накапливались. См. примечание ниже, как выполнить это требование	Определяется таблицами метрик шрифта HMTX или VMTX шрифтов
uOffset, vOffset	Смещение относительно начала координат глифа для перемещения данного глифа. Обычно используется для присоединения знаков к базовым символам. Измеряется в 1/100 от размера в em шрифта	0,0
Flags	Отличает базовые глифы и комбинирующие знаки	0 (базовый глиф)

В отношении вычисления продвижения без накопления ошибок округления рассмотрим следующее. Каждое значение продвижения должно вычисляться как точное неокругленное начало координат последующего глифа минус сумма вычисленной (т.е. округленной) полной ширины предыдущих глифов. Таким образом, каждый глиф располагается в пределах 0,5% em от его точного положения.

Примеры разметки глифов

```
<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' без кернинга  -->

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


<!-- 'Hello Windows' с кернингом    -->

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


<!-- 'Open file' без лигатуры 'fi'  -->

<Glyphs
   OriginX             = "200"
   OriginY             = "250"
   UnicodeString       = "Open file"
   FontUri             = "file://C:/Windows/Fonts/Times.TTF"
```

```
    Fill                = "#00FF00"
    FontRenderingEmSize = "20"
/>

<!-- 'Open file' с лигатурой 'fi' -->

<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', используя предварительно сформированную 'е' -->

<Glyphs
    OriginX             = "200"
    OriginY             = "450"
    xml:lang            = "ru-RU"
    UnicodeString       = "ежик в тумане"
    FontUri             = "file://C:/Windows/Fonts/Times.TTF"
    Fill                = "#00FF00"
    FontRenderingEmSize = "20"
/>

<!-- 'ежик в тумане', используя формирование 'е' и тремы -->

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

<!-- 'ежик в тумане' принудительная визуализация справа налево, изображающая
        комбинирующий знак в логическом порядке -->

<Glyphs
    OriginX             = "200"
    OriginY             = "550"
    BidiLevel           = "1"
    xml:lang            = "ru-RU"
    UnicodeString       = "ежик в тумане"
    Indices             = "(1:2)72;142,0,-45"
    FontUri             = "file://C:/Windows/Fonts/Times.TTF"
    Fill                = "#00FF00"
    FontRenderingEmSize = "20"
/>

</Canvas>
```

Оптимизация размера разметки глифов

Подробности разметки, такие как индексы глифов и полная ширина, могут быть исключены из разметки, если целевой клиент может надежно их восстанавливать. Следующие варианты дают возможность получить существенную оптимизацию обычно используемых простых рукописных шрифтов.

Оптимизация разметки индексов глифов

Индексы глифов могут быть исключены из разметки, где существует отображение “один в один” между положеиями символов в строке уникода и положениями глифов в строке глифов, и индекс глифа представляет собой значение в таблице CMAP (отображения символов) шрифта, и символ уникода имеет однозначную семантику.

Индексы глифов должны быть представлены в разметке, где отображение символов в глифы:

- не является “один в один”, так что где два или более кодовых значения образуют единственный глиф (лигатуру), или
- одно кодовое значение генерирует многочисленные глифы, или
- где произошел любой другой вид замены глифов, такой как в результате применения особенности OpenType.

Индексы глифов должны быть представлены в разметке, где процессор визуализации может заменять глифами, которые отличны от тех, которые в таблице CMAP (отображения символов) в шрифте. Индексы глифов должны быть представлены, где требуемое представление глифа не является тем, которое в таблице CMAP шрифта.

Оптимизация разметки положений глифа

Полная ширина глифа может быть исключена из разметки, где требуемая полная ширина точно равна полной ширине для глифа в таблицах HMTX (горизонтальных метрик) или VMTX (вертикальных метрик) шрифта.

Вертикальное смещение глифа может быть исключено из разметки, где оно равно нулю. Это почти всегда верно для базовых символов и обычно верно для комбинирующих знаков в более простых рукописных шрифтах. Однако это часто не верно для комбинирующих знаков в более сложных рукописных шрифтах, например арабского языка и индийской группы индоевропейских языков.

Оптимизирование разметки флагов глифа

Флаги глифа могут быть исключены для базовых глифов с нормальным приоритетом выравнивания.

Интерфейс прикладного программирования

Нижеследующее описывает примерный вариант выполнения платформно-независимого интерфейса прикладного программирования (ИПП) пакетирования. Данный уровень ИПП состоит из абстрактных классов и других базовых классов, которые включены как часть уровня пакетирования. ИПП включает в себя классы, описанные ниже.

Контейнер

Контейнер представляет собой логическую сущность, которая хранит вместе коллекцию частей.

```
namespace System.IO.MMCF
{
  // класс Container представляет логическую сущность, которая хранит вместе коллекцию Part

  public class abstract Container : IDisposable
  {
    // Открытые свойства

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

    // Открытые статические методы

    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);
    // Открытые методы

    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 ();

    //  Защищенные методы - для специальной реализации

    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 ();

    // Защищенный конструктор
    protected Container (FileInfo fileInfo, FileAccess access);

  }
}
```

Конструкторы

protected *Container* ***(FileInfo fileInfo, FileAccess access)***

Защищенный конструктор для базового класса. Явно определяющий в этом классе, так что легче документировать и поддерживать этот конструктор. Компилятор все равно добавит его, если он не задан. Также это текущий контракт между абстрактным классом и подкассами. Когда объект fileInfo равен null, он определяет, что контейнер был открыт или создан на потоке.

Свойства

public virtual DateTime *CreationTime* ***{get; set;}***

Получает или устанавливает время создания этого контейнера. Когда это значение установлено, LastAccessTime и LastWriteTime также должны быть обновлены до одинакового значения. Объект System.IO.FileInfo используется для манипулирования этим значением.

Исключения

- InvalidArgumentException - Если CreationTime устанавливается на значение, которое больше, чем LastAccessTime или LastWriteTime.

- InvalidOperationException - Если Container был открыт на потоке, нет возможности

получить это свойство.

public virtual DateTime LastAccessTime **{get; set;}**

Получает или устанавливает последний момент времени, когда этот контейнер был открыт. Объект System.IO.FileInfo используется для манипулирования этим значением.

Исключения

- InvalidArgumentException - Если LastAccessTime устанавливается на значение, которое меньше, чем CreationTime или LastWriteTime.

- InvalidOperationException - Если Container был открыт на потоке, нет возможности получить это свойство.

public virtual DateTime LastWriteTime **{get; set;}**

Получает или устанавливает, когда в последний раз этот контейнер был модифицирован. Также когда обновляется LastWriteTime, LastAccessTime должен быть обновлен до такого же значения. Объект System.IO.FileInfo используется для манипулирования этим значением.

Исключения

- InvalidArgumentException - Если LastWriteTime устанавливается на значение, которое меньше, чем CreationTime.

- InvalidOperationException - Если Container был открыт на потоке, нет возможности получить это свойство.

public FileAccess FileOpenAccess **{get:}**

Получает FileAccess, с которым был открыт контейнер. Это свойство только для чтения. Это свойство устанавливается, когда открывается контейнер.

public abstract Part StartingPart **{get; set;}**

Получает или устанавливает StartingPart контейнера.

Методы

public static Container OpenOnFile **(string path)**

Эта переопределенная версия метода OpenOnFile возвращает Container, заданный на данном пути. Этот метод вызывает переопределение, которое принимает все параметры со следующими установками по умолчанию -

FileMode - FileMode.OpenOrCreate

FileAccess - FileAccess.ReadWrite

FileShare - FileShare.None

public static Container OpenOnFile **(string path, FileMode mode)**

Эта переопределенная версия метода OpenOnFile возвращает Container, заданный по данному пути в заданном режимном коде файла. Этот метод вызывает переопределение, которое принимает все параметры со следующими установками по умолчанию -

FileAccess - FileAccess.ReadWrite

FileShare - FileShare.None

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

Эта переопределенная версия метода OpenOnFile возвращает Container, заданный по данному пути в заданном режимном коде файла и доступе к файлу. Этот метод вызывает переопределение, которое принимает все параметры со следующими установками по умолчанию -

FileShare - FileShare.None

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

Эта переопределенная версия метода OpenOnFile открывает контейнер по данному

пути с режимным кодом, доступом и совместным использованием, установленными в представленные значения.

Исключения

- InvalidArgumentException - Если комбинация параметров FileMode, FileAccess и FileShare не является значащей.

public static Container** OpenOnUri **(Uri uri)

Эта переопределенная версия метода OpenOnUri возвращает Container, заданный по данному uri. Этот метод вызывает переопределение, которое принимает все параметры со следующими установками по умолчанию -

FileMode - FileMode.Open

FileAccess - FileAccess.Read

public static Container** OpenOnUri **(Uri uri, FileMode mode)

Эта переопределенная версия метода OpenOnUri возвращает Container, заданный по данному uri в заданном режимном коде файла. Этот метод вызывает переопределение, которое принимает все параметры со следующими по умолчанию -

FileAccess - FileAccess.Read

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

Эта переопределенная версия метода OpenOnUri открывает контейнер по данному uri с режимным кодом файла и доступом, установленными в представленные значения. Механизм WebRequest/WebResponse используется для получения контейнера. Параметры FileMode и FileAccess применяются к контейнеру, который будет открыт. Этот метод вызывает метод OpenOnStream с правильным типом содержимого.

Исключения

- InvalidArgumentException - Если комбинация параметров FileMode, FileAccess и FileShare не является значащей.

public static Container** OpenOnStream **(Stream stream, string contentType)

Эта переопределенная версия метода OpenOnStream возвращает Container на предусмотренном потоке. Этот метод вызывает переопределение, которое принимает все параметры со следующими по умолчанию -

FileMode - FileMode.Open

FileAccess - FileAccess.Read

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

Эта переопредленная версия метода OpenOnStream возвращает Container на предусмотренном потоке в заданном режимном коде файла. Этот метод вызывает переопределение, которое принимает все параметры со следующими по умолчанию -

FileAccess - FileAccess.Read

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

Эта переопределенная версия метода OpenOnStream открывает контейнер на предусмотренном потоке с режимным кодом и доступом, установленными в представленные значения. Параметры FileMode и FileAccess применяются к контейнеру, который будет открыт. Параметр contentType используется для создания экземпляра объекта соответствующего подкласса.

Исключения

- InvalidArgumentException - Если комбинация параметров FileMode, FileAccess и FileShare не является значащей.

public Part** AddPart **(MMCFUri uri, string contentType)

Part для данного Uri добавляется к контейнеру. Этот метод добавляет Part с пустым

потоком, если не выполняется явный вызов на считывание или запись в поток. Этот метод вызывает AddPartCore, который выполняет фактическую работу, относящуюся к физической реализации.

Исключения

- InvalidArgumentException - Если Part, соответствующая этому Uri, уже существует в контейнере.

public Part** GetPart **(MMCFUri uri)

Возвращается Part для данного Uri. Uri относительно корня Container. Этот метод вызывает GetPartCore, который фактически извлекает часть.

Исключения

- InvalidArgumentException - Если Part, соответствующая этому Uri, не существует в контейнере.

public virtual bool** Exists **(MMCFUri uri)

Так как можно иметь точку отношений к целям, которые все еще не существуют, этот метод обеспечивает удобный путь обнаружения того, существует ли фактически Part в нижележащем Container. Этот uri должен быть относительно корня контейнера.

public void** DeletePart **(MMCFUri uri)

Этот метод удаляет часть контейнера из текущего контейнера. Также будут удалены все отношения, для которых эта часть была SourcePart. Этот метод удаляет нижележащий поток, и объект будет ликвидирован. Также если существуют открытыми многочисленные экземпляры этой Part, то тогда Dispose все другие открытые экземпляры этой части. Этот метод выполняет необходимую очистку, чтобы ввести в действие этот характер изменения, однако фактическое удаление потока является характерным для нижележащей физической реализации и поэтому вызывает метод DeletePartCore, который удалит фактический поток. Ожидающие обработки перечислители Part будут сделаны недействительными.

public PartCollection** GetParts **()

Он возвращает коллекцию всех Part внутри контейнера. Отношения не возвращаются.

public void** Flush **()

Этот метод вызывает сброс для отдельных частей, которые были открыты, таким образом осуществляя то, что все части и отношения сбрасываются на нижележащий контейнер. По существу, этот класс поддерживает массив всех Part, которые он выдает, и затем вызывает Flush для всех Part. Он затем вызывает FlushCore (), который выполняет работу, характерную для контейнера в целом.

public virtual void** Dispose **()

Все открытые Part и Relationship сбрасываются на нижележащий контейнер. Так как этот класс поддерживает массив всех выданных Part, этот метод вызывает Dispose () на все выданные Part. Если необходимо очистить любые другие ресурсы, тогда подклассы должны переопределить этот метод для выполнения дополнительной очистки.

public void** Close **()

Метод Close аналогичен Dispose, поэтому внутренне он выполняет вызов метода Dispose.

public Relationship** AddRelationship **(Uri uri)

Этот метод добавляет отношение между Container и Part, заданные при помощи URI. Он возвращает объект Relationship. Это изменение сбрасывается на нижележащий контейнер только после выполнения вызова метода Flush (). Может быть множество

отношений между одинаковыми источником и целью. Будут сделаны недействительными все ожидающие выполнения перечислители отношений.

public void** DeleteRelationship **(Relationship relationship)

Этот метод удаляет целевое отношение, заданное объектом Relationship. Это изменение сбрасывается на нижележащий контейнер только после выполнения вызова метода Flush (). Операция удаления выполняется на основе "имени" объекта, и как таковой каждый объект однозначно идентифицируется. Будут сделаны недействительными все ожидающие выполнения перечислители отношений.

Исключения

- InvalidArgumentException - Если источник отношения не аналогичен текущей Part.

public RelationshipCollection** GetRelationships **()

Он возвращает коллекцию всех целевых отношений из Container. Так как целевые отношения для контейнера расположены по известным uri, можно обеспечить реализацию по умолчанию, которая будет открывать Part отношения и затем считывать xml из потока и создавать объект коллекции (Исключения - Если XML, который считывался из нижележащего контейнера, был неправильно образован.)

protected abstract Part** AddPartCore **(MMCFUri uri, string contentType)

Этот метод предназначен для специальной реализации для лежащего в основе формата файла. Он будет вызываться из метода AddPart. Он фактически будет создавать часть в нижележащем контейнере. Пустая часть должна создаваться как результат этого вызова.

protected abstract Part** GetPartCore **(MMCFUri uri)

Этот метод предназначен для специальной реализации для лежащего в основе формата файла. Он вызывается из метода GetPart. Этот метод выбирает фактическую часть из нижележащего контейнера. Если часть не существует, он должен возвратить "null".

protected abstract void** DeletePartCore **(MMCFUri uri)

Этот метод предназначен для специальной реализации для лежащего в основе формата файла. Он должен фактически удалять поток, соответствующий этой части. Также, если не существует часть, соответствующая данному URI, она не должна отбрасываться.

protected abstract Part []** GetPartsCore **()

Этот метод предназначен для специальной реализации для лежащего в основе формата файла. Он должен возвращать массив всех Part в контейнере. Так как получение всех частей в контейнере характерно для фактического физического формата, этот метод является полезным. Он предусмотрен так, что фактический вызов GetParts просто передает этот массив в PartCollection и можно обеспечить перечислитель этого. Таким образом, класс PartCollection может быть конкретным классом. Также, если нет частей в контейнере, GetPartsCore должен возвращать пустой массив.

protected abstract void** FlushCore **()

Этот метод предназначен для специальной реализации для лежащего в основе формата файла. Этот метод сбрасывает на диск все содержимое.

protected abstract void** DisposeCore **()

Этот метод предназначен для специальной реализации для лежащего в основе формата файла. Этот метод должен освобождать ресурсы, соответствующие фактическому физическому формату.

Часть

Часть состоит из трех позиций:

URI - относительный корня контейнера.

ContentType - это тип mime потока, представленного этой частью

Stream - фактический поток, соответствующий этой части.

Кроме того, части могут быть связаны с другими частями при помощи Relationship. SourcePart в отношении, обладает отношением.

```
namespace System.IO.MMCF
{
  // Этот класс представляет Part, которая состоит из Uri, ContentType и нижележащего потока
  // Часть может быть связана с другой частью с использованием Relationship

  {
    // Открытые свойства

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

    // Открытые методы

    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 ();

    // Защищенный конструктор
    protected Part (Container container, MMCFUri uri, string contentType);

   }
}
```

Конструкторы

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

Защищенный конструктор для базового класса. Явно определяющий в этом классе, так что легко документировать и поддерживать этот конструктор. Компилятор все равно добавит его, если он не задан. Также это текущий контракт между абстрактным классом и подклассами.

Свойства

public MMCFUri** Uri **{get;}

Это свойство возвращает MMCFUri для Part. Это свойство только для чтения

public string** ContentType **{get;}

*Это свойство возвращает тип содержимого потока, представленного при помощи Part. Это свойство только для чтения. public Container **Container** {get;}*

Это свойство возвращает родительский контейнер для Part. Это свойство только для чтения.

Методы

public Stream** GetStream **()

Этот метод возвращает поток, соответствующий этой части. Он вызывает переопределение, которое принимает все параметры со следующими по умолчанию -

FileMode - Open

FileAccess - ReadWrite

public Stream** GetStream **(FileMode mode)

Этот метод возвращает поток, соответствующий этой части в заданном режиме. Он вызывает переопределение, которое принимает все параметры со следующими по умолчанию -

FileAccess - ReadWrite

public Stream *GetStream* ***(FileMode mode, FileAccess access)***

Этот метод возвращает поток, соответствующий этой части. Он вызывает метод GetStreamCore, который возвращает фактический поток. Этот метод выполняет действия по обслуживанию, необходимые для отслеживания всех открытых потоков.

public abstract Stream *GetStreamCore* ***(FileMode mode, FileAccess access)***

Этот метод возвращает поток, соответствующий этой части. Этот метод предназначен для специальной реализации.

public Relationship *AddRelationship* ***(Uri uri)***

Этот метод добавляет отношение между Part по заданному URI и текущей Part. Он возвращает объект Relationship. Это изменение сбрасывается на нижележащий контейнер только после выполнения вызова метода Flush (). Может быть множество отношений между одинаковыми источником и целью. Будут сделаны недействительными все ожидающие выполнения перечислители отношений.

Исключения

- InvalidOperationException - Если текущая Part является отношением.

public void *DeleteRelationship* ***(Relationship relationship)***

Этот метод удаляет целевое отношение, заданное объектом Relationship. Это изменение сбрасывается в нижележащий контейнер только после выполнения вызова метода Flush (). Операция удаления выполняется, основываясь на "ссылке" объекта, и как таковой каждый объект идентифицируется однозначно. Будут сделаны недействительными все ожидающие выполнения перечислители отношений.

Исключения

- InvalidArgumentException - Если источник отношения не такой же, как текущая Part.

public RelationshipCollection *GetRelationships ()*

Он возвращает коллекцию всех целевых отношений этой Part. Так как целевые отношения для этой Part расположены по известному uri, можно обеспечить реализацию по умолчанию, которая будет открывать Part отношения и затем считывать xml из потока и создавать объект коллекции (Исключения - Если XML, который считывается из нижележащего контейнера, был неправильно образован.)

Отношение

Этот класс используется для выражения отношения между частью источника и частью цели. Единственным путем создания Relationship является вызов Part.AddRelationship (Uri uri). Отношение принадлежит части источника, и, если удаляется часть источника, также удаляются все отношения, которыми она владеет. Цель отношения необязательно должна присутствовать.

```
namespace System.IO.MMCF
{
  // Этот класс представляет отношение между частью источника и частью цели. Единственным путем
     создания Relationship является вызов Part.AddTargetRelationship (Uri uri). Отношение принадлежит
     части источника. Поэтому, если удаляется часть источника, также удаляются все отношения,
     которыми она владеет


  public class Relationship
  {
    // Открытые свойства
    public Part Source { get;  }
    public Uri TargetUri { get; }
    public string Name { get;}

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

   }
}
```

Конструкторы

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

Возвращает объект Relationship.

Свойства

***public Part** Source {get;}*

Получает часть Source дляRelationship. Это свойство только для чтения. Оно устанавливается, когда создается Relationship.

***public Uri** TargetUri {get;}*

Получает TargetUri дляRelationship. Этот uri должен рассматриваться как являющийся относительным для uri источника.

***public string** Name {get;}*

Получает имя, соответствующее этому Relationship.

PartCollection

Это коллекция частей в контейнере.

```
namespace System.IO.MMCF
{
  // Эта коллекция Part со строгим контролем типов
  public class PartCollection : IEnumerable
  {
    // Член класса IEnumerable
    public IEnumerator GetEnumerator ();
    // Внутренние конструкторы
    internal PartCollection (Dictionary<MMCFUri, Part> partDictionary);
   }
}
```

Конструкторы

***internal** PartCollection (Dictionary<MMCFUri, Part>partDictionary)*

Создает PartCollection, основанную на обобщенном словаре объектов Part.

Методы

public IEnumerator GetEnumerator()

Член интерфейса IEnumerable. Он возвращает перечислитель по коллекции частей.

RelationshipCollection

Это коллекция отношений, ассоциированных с частью в контейнере. Может быть более одного отношения между данной частью источника и частью цели.

```
namespace System.IO.MMCF
{
  // Это коллекция Relatioship со строгим контролем типов
  public class RelationshipCollection : IEnumerable
  {
    // Член класса IEnumerable
    public IEnumerator GetEnumerator();
    // Внутренние конструкторы
    internal RelationshipCollection (Relationship [] relationships);
   }
}
```

Конструкторы

***internal** RelationshipCollection (Relationship [] relationship)*

Создает RelationshipCollection, основанную на массиве объектов Relationship.

Методы

public IEnumerator GetEnumerator ()

MMCFUri

Этот класс наследуется из класса URI. Основной функцией этого класса Uri является обеспечение того, что заданный URI начинается с “/”. Мотивацией для этого класса

является:

1. Убедиться в том, что URI, используемый для каждой отдельной части, начинается с “/”. Это обеспечивает то, что все имена частей относительны корня контейнера.

2. Так как класс System.Uri не позволяет разрешать два относительных URI, они должны быть разрешены различным образом, и поэтому было бы хорошо иметь эту логику в одном месте.

3. Привести в исполнение то, что Container представляет собой авторитетный источник. Таким образом, любые относительные ссылки не должны разрешаться на размещение вне контейнера.

```
namespace System.IO.MMCF
{
  // Этот класс используется для создания идентификаторов URI, которые всегда
     начинаются с "/"
  public class MMCFUri : Uri
  {
    // Открытые конструкторы

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

  }
}
```

public** MMCFUri **(string uri)

Создает объект MMCFUri из предоставленной строки uri. Гарантирует, что Uri является относительным и начинается с “/”.

Исключения - InvalidArgumentException - Если URI состоит из имени хоста и протокола, т.е. он является абсолютным URI.

public** MMCFUri **(MMCFUri baseUri, string relativeUri)

Создает объект MMCFUri из предоставленного объекта Uri и строки относительного Uri. Авторы разрешают относительный uri в отношении базового Uri. Гарантирует, что Uri является относительным и начинается с “/”.

Исключения - InvalidArgumentException - Если URI состоит из имени хоста и протокола, т.е. он является абсолютным URI.

Образец кода

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

    // Создает контейнер и добавляет в него часть

    Container c = new Container.OpenOnFile(“myFilePath”);
    Part p1 = c.AddPart(“mypart1”, “contentType”); // Создает пустой поток
    c.Close();

    // Открывает этот же контейнер и добавляет другую часть и отношение

    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();

  }
}
```

№	Доступ к текущему потоку		доступ к требуемому контейнеру		
	CanRead	CANWRITE	Read	Write	ReadWrite
1	true	true	Ограничение до ReadOnly	Ограничение до WriteOnly	Все в порядке

2	false	true	Отбрасывается	Все в порядке	Отбрасывается
3	true	false	Все в порядке	Отбрасывается	Отбрасывается
4	false	false	Отбрасывается	Отбрасывается	Отбрасывается

Другие подробности ИПП

Методы OpenOnFile, OpenOnUri и OpenOnStream

Эти методы имеют жестко запрограммированную логику, и единственным физическим форматом контейнера, который знает эти методы, является реализация Составного файла. Так как авторы владеют этими классами, существует некоторое предположение о конструкторах подклассов, которые вызываются из этих Статических методов. Также эти статические методы создают экземпляры правильного объекта подкласса, основанные на расширении файла или типе содержимого потока, существующего в настоящее время.

Метод OpenOnStream и включение заданного доступа для Container

При создании Container на потоке важно гарантировать, что FileAccess, заданный для контейнера, совместим с потоком, который предусмотрен. Следующая таблица перечисляет различные возможности и пример того, как их рассматривать.

В первом ряду, где поток имеет больший доступ, и авторы хотят, чтобы создаваемый контейнер был более ограниченным, они заключают в оболочку входящий поток при помощи частного (закрытого) потока, называемого RestrictedStream, который имеет соответствующие значения CanRead и CanWrite.

Кэширование в памяти объектов Part и Relationship

Словарь поддерживает все части, к которым был осуществлен доступ, и, если часть запрашивается второй раз, возвращается ссылка на эту часть из словаря. Это более эффективно, и, так как объект Part неизменяемый, это делается без каких-либо проблем. Это же применимо к объектам Relationship. Однако, если нижележащий контейнер был открыт в совместно используемом режиме записи и второй пользователь добавляет или удаляет части из нижележащего контейнера, эти изменения могут не отражаться в этом кэше.

Заключение

Вышеописанные способы и системы модульной инфраструктуры содержимого и формата документов обеспечивают набор стандартных компоновочных блоков для составления, пакетирования, распространения и визуализации документно-ориентированного содержимого. Эти стандартные компоновочные блоки определяют платформно-независимую инфраструктуру для форматов документов, которые дают воможность программным и аппаратным системам генерировать, обмениваться и отображать документы надежным и единообразным образом. Изображенный и описанный формат расширенного пакета обеспечивает формат для хранения разбиваемых на страницы или предварительно разбитых на страницы документов так, что содержимое расширенного пакета может отображаться или печататься с высокой точностью воспроизведения на устройствах и в приложениях в широком спектре сред и в широком спектре сценариев. Хотя изобретение было описано на языке, характерном для конструктивных признаков и/или методологических этапов, необходимо понять, что изобретение, определенное в прилагаемой формуле изобретения, не ограничивается непременно конкретными описанными признаками или этапами. Скорее, конкретные признаки и этапы описаны в виде предпочтительной формы реализации заявленного изобретения.

Формула изобретения

1. Способ определения документа, содержащий этапы, на которых

создают пакет, который определяет документ, причем данный пакет включает в себя множество частей, которые составляют этот документ, при этом каждая из упомянутого множества частей имеет ассоциированное имя и ассоциированный тип содержимого, при этом пакет приспособлен содержать более одной полезной нагрузки, причем каждая полезная нагрузка выступает в роли отличающегося от других представления документа и содержит части, необходимые для обработки этого представления документа, причем по меньшей мере некоторые из упомянутых отличающихся представлений включают в себя одно и то же содержимое, при этом пакет дополнительно содержит множество элементов отношения, причем каждый элемент отношения ассоциирован с одной из упомянутого множества частей, задает отношение между упомянутой одной из упомянутого множества частей и одной или более других из упомянутого множества частей и обеспечивает возможность выявлять это отношение независимо от ассоциированной части; и

обеспечивают множество драйверов, ассоциированных с пакетом, причем упомянутое множество драйверов также ассоциировано с различными форматами документа, и упомянутое множество драйверов обеспечивает множеству приложений возможность доступа к пакету независимо от формата документа, ассоциированного с каждым из упомянутого множества приложений.

2. Способ по п.1, в котором имя, ассоциированное с каждой частью, является уникальным.

3. Способ по п.1, в котором имя, ассоциированное с каждой частью, включает в себя иерархический адрес.

4. Способ по п.1, в котором имя, ассоциированное с каждой частью, включает в себя локальную долю универсального идентификатора информационного ресурса.

5. Способ по п.1, в котором тип содержимого, ассоциированный с каждой частью, включает в себя описание типа информации, содержащейся в части.

6. Способ по п.1, в котором каждая из упомянутого множества частей отображена в конкретный файл.

7. Способ по п.1, в котором упомянутое множество частей хранится в единственном файле.

8. Способ по п.1, в котором по меньшей мере один из драйверов дополнительно ассоциирован с различными протоколами передачи данных.

9. Способ по п.8, в котором упомянутое множество драйверов обеспечивает упомянутому множеству приложений возможность доступа к пакету независимо от формата передачи данных, ассоциированного с каждым из упомянутого множества приложений.

10. Способ по п.8, в котором упомянутое множество драйверов обеспечивает упомянутому множеству приложений возможность доступа к пакету независимо от формата файла, ассоциированного с каждым из упомянутого множества приложений.

11. Способ по п.1, в котором каждая из упомянутого множества частей дополнительно имеет ассоциированный набор отношений.

12. Способ по п.1, в котором упомянутое множество частей, которые составляют документ, включает в себя первую часть, имеющую первый ассоциированный тип содержимого, и вторую часть, имеющую второй ассоциированный тип содержимого.

13. Способ по п.1, в котором данные типа содержимого хранятся в файле, ассоциированном с пакетом.

14. Способ по п.1, дополнительно содержащий этап, на котором передают пакет на носитель, с которого пакет может потребляться потребителем.

15. Машиночитаемый носитель, имеющий машиночитаемые инструкции, которые при их исполнении реализуют способ по п.1.

16. Вычислительная система, содержащая машиночитаемый носитель по п.15.

17. Способ обработки пакета, содержащий этапы, на которых

принимают пакет, включающий в себя множество частей, при этом каждая из упомянутого множества частей уникально идентифицируется и адресуется при помощи ассоциированного имени, причем упомянутое множество частей составляет фиксированную полезную нагрузку, которая является фиксированным представлением документа и содержит корневую часть, которая ссылается на одну или более страничных частей, каждая из которых содержит разметку, описывающую визуализацию отдельной страницы содержимого, при этом пакет дополнительно содержит множество элементов отношения, причем каждый элемент отношения ассоциирован с одной из упомянутого множества частей, задает отношение между упомянутой одной из упомянутого множества частей и одной или более других из упомянутого множества частей и обеспечивает возможность выявлять это отношение независимо от ассоциированной части;

идентифицируют драйвер, ассоциированный с принятым пакетом; и

обрабатывают принятый пакет посредством программы приложения с использованием драйвера, ассоциированного с принятым пакетом.

18. Способ по п.17, в котором каждую из упомянутого множества частей идентифицируют посредством локальной доли универсального идентификатора информационного ресурса.

19. Способ по п.17, в котором каждая часть из упомянутого множества частей имеет ассоциированный тип содержимого, который включает в себя описание типа данных, содержащихся в этой части.

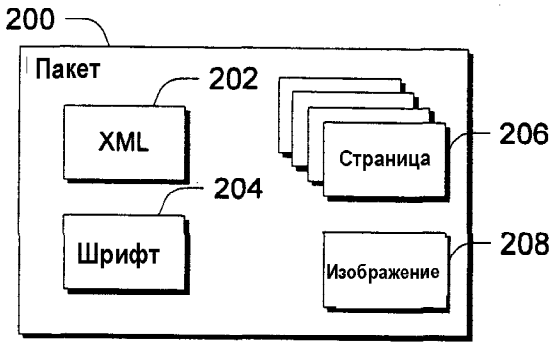
20. Способ по п.17, в котором драйвер, ассоциированный с принятым пакетом, также ассоциирован с конкретным протоколом передачи данных.

21. Способ по п.17, в котором драйвер, ассоциированный с принятым пакетом, также ассоциирован с конкретным форматом файла.

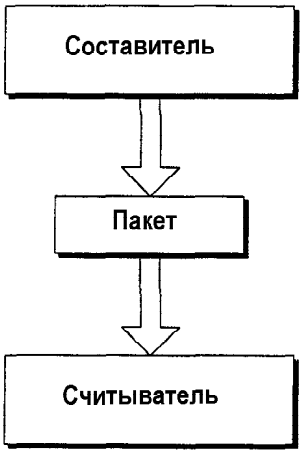
22. Способ по п.17, дополнительно содержащий этап, на котором идентифицируют набор отношений, ассоциированных с принятым пакетом.

23. Машиночитаемый носитель, имеющий машиночитаемые инструкции, которые при их исполнении реализуют способ по п.17.

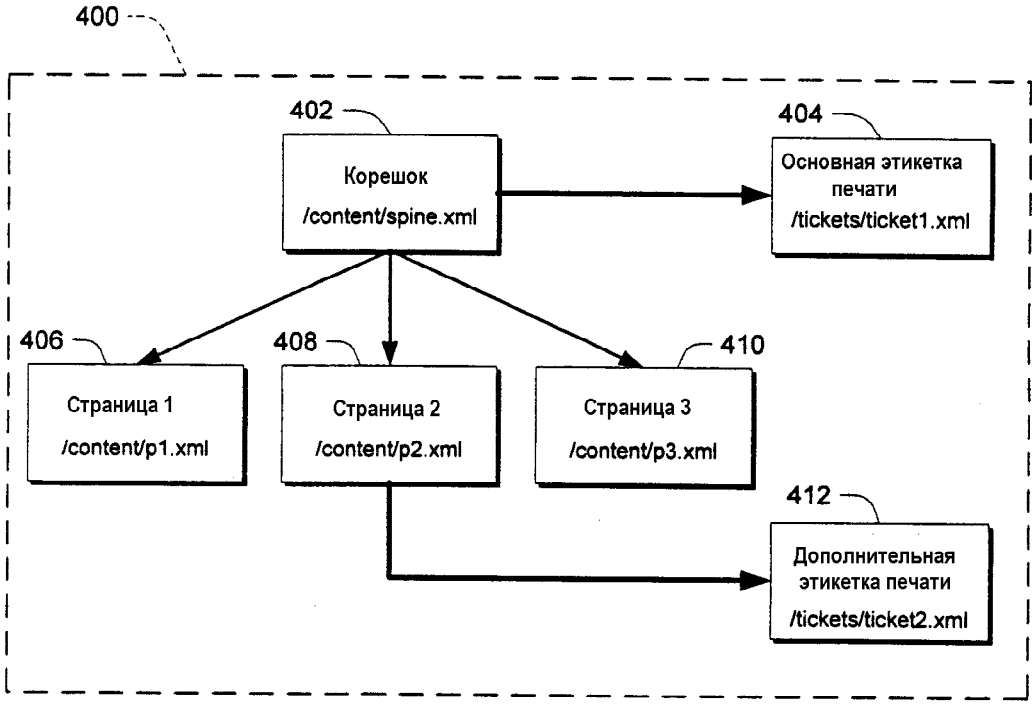
24. Вычислительная система, содержащая машиночитаемый носитель по п.23.



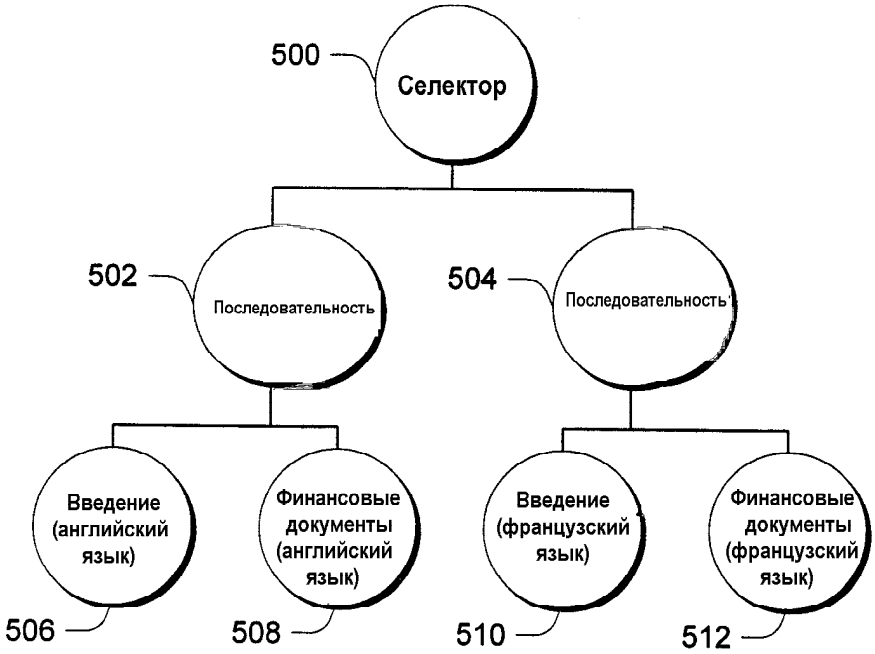
ФИГ.2



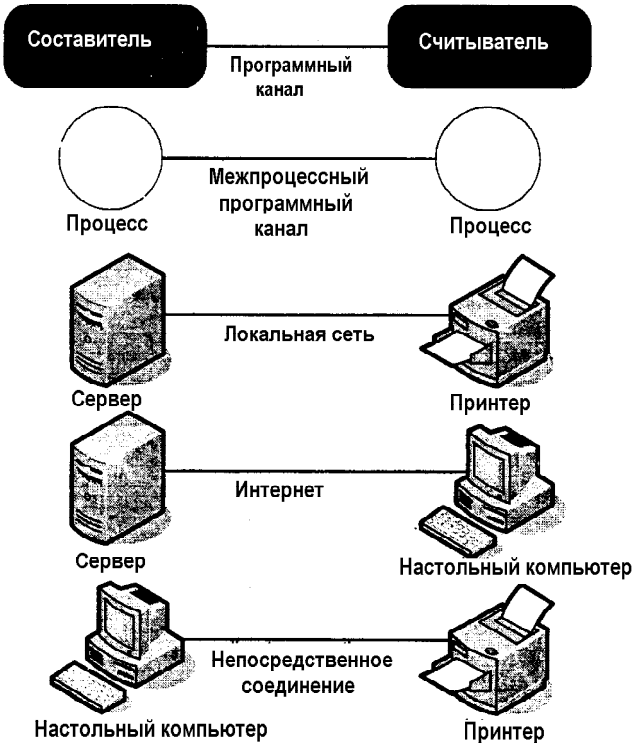
ФИГ.3



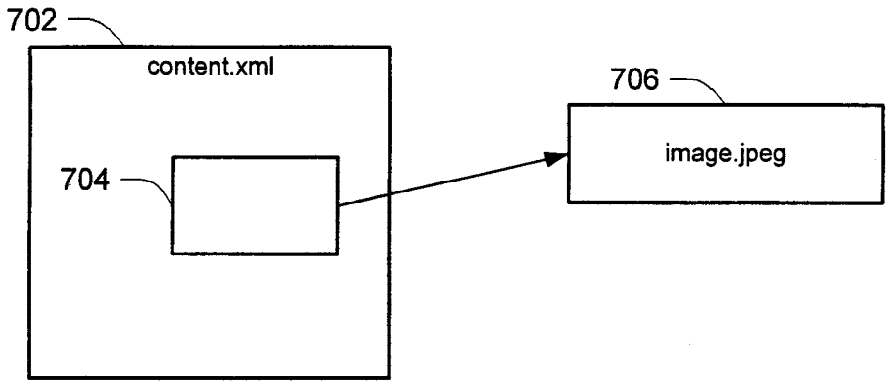
ФИГ.4


500
Селектор
502
Последовательность
504
Последовательность
Введение (английский язык)
Финансовые документы (английский язык)
Введение (французский язык)
Финансовые документы (французский язык)
506
508
510
512

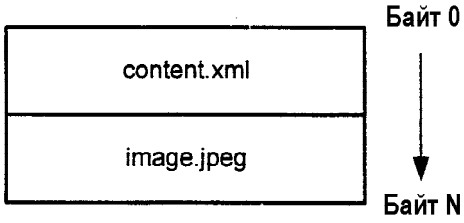
ФИГ.5


Составитель
Программный канал
Считыватель
Процесс
Межпроцессный программный канал
Процесс
Локальная сеть
Сервер
Принтер
Интернет
Сервер
Настольный компьютер
Непосредственное соединение
Настольный компьютер
Принтер

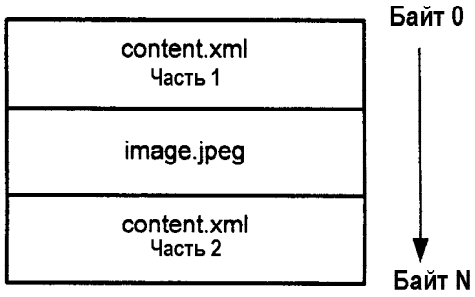
ФИГ.6



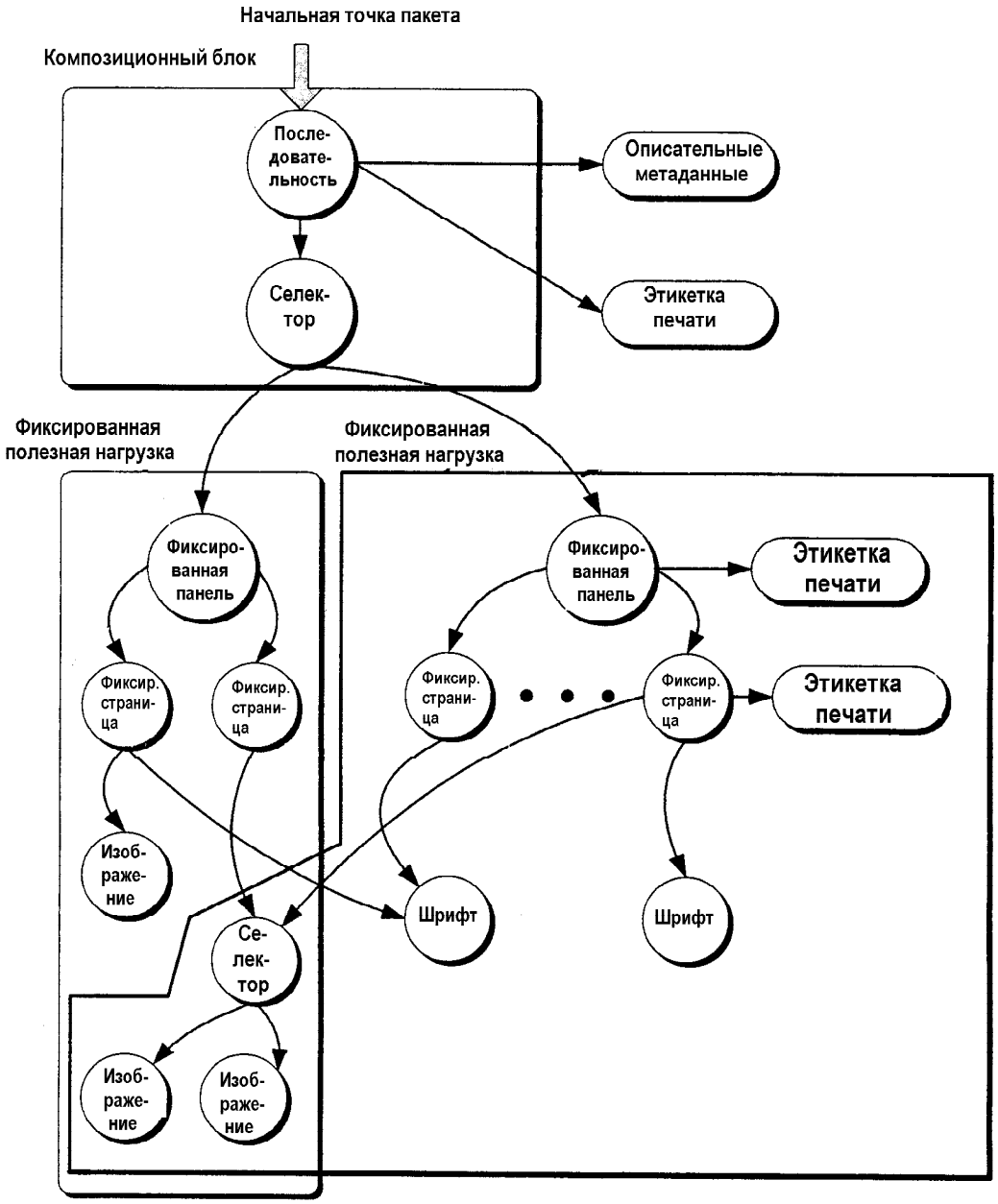
ФИГ.7



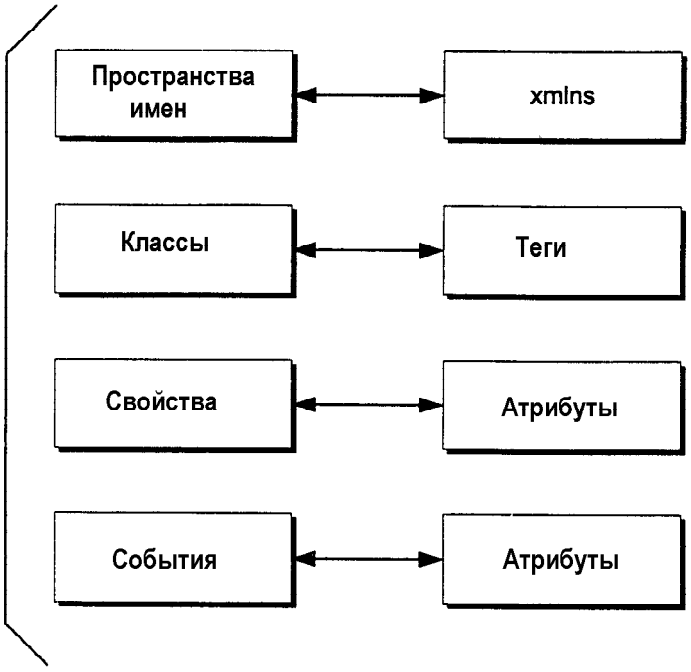
ФИГ.8



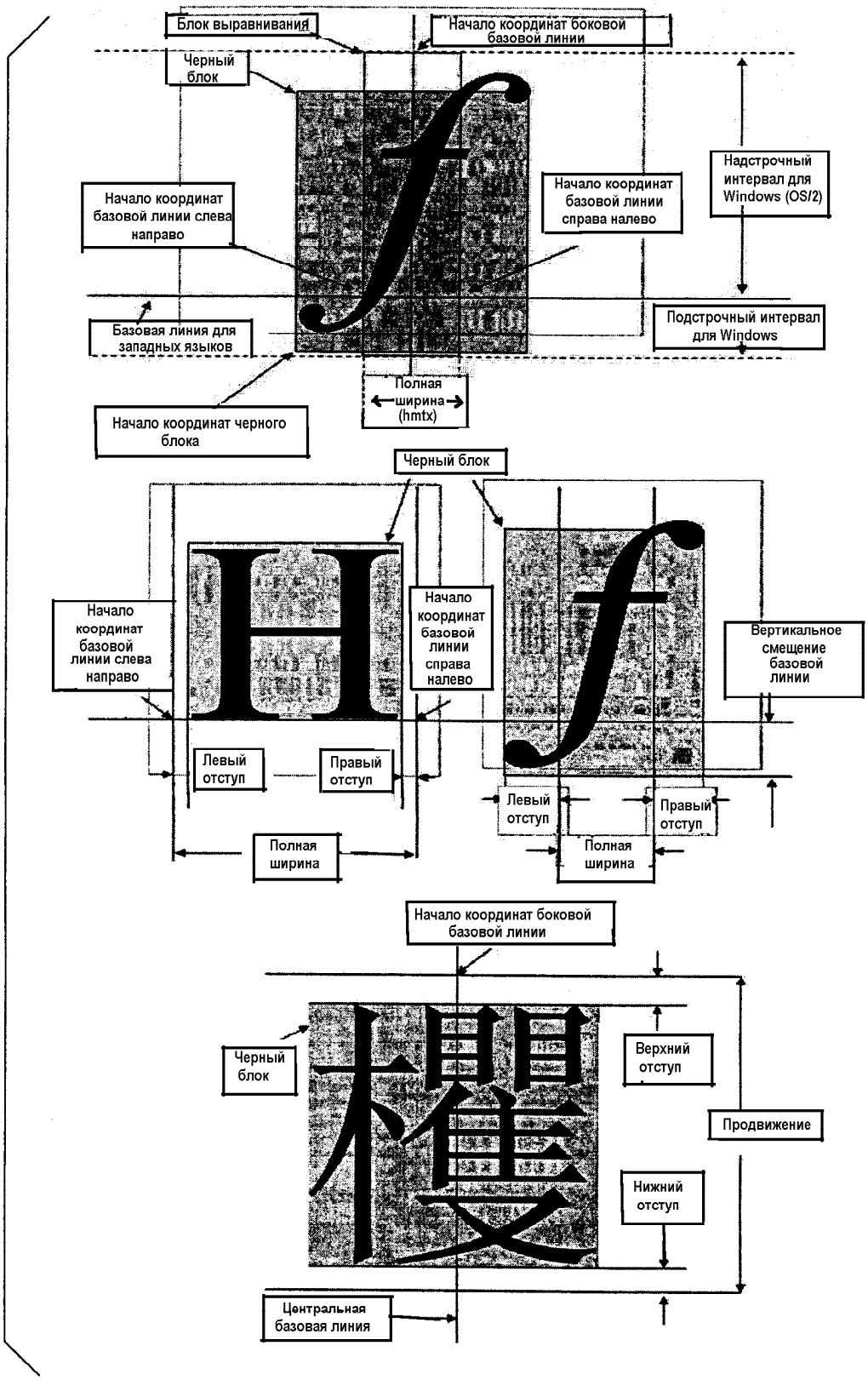
ФИГ.9


Начальная точка пакета
Композиционный блок
Последовательность
Описательные метаданные
Селектор
Этикетка печати
Фиксированная полезная нагрузка
Фиксированная полезная нагрузка
Фиксированная панель
Фиксированная панель
Этикетка печати
Фиксир. страница
Фиксир. страница
Фиксир. страница
Фиксир. страница
Этикетка печати
Изображение
Шрифт
Шрифт
Селектор
Изображение
Изображение

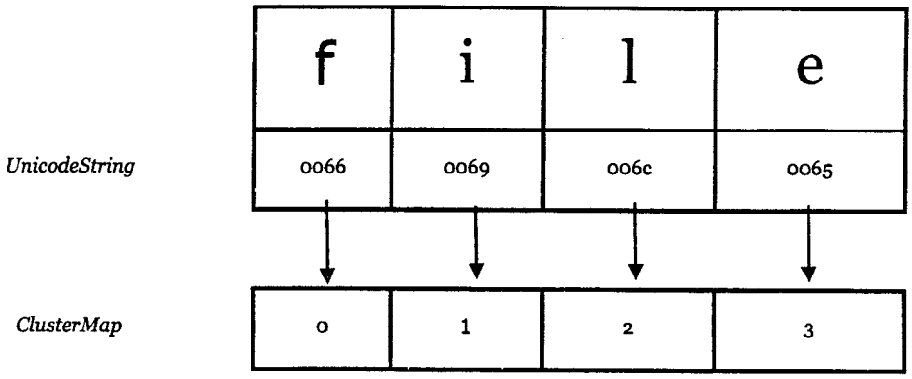
ФИГ.10


Пространства имен
xmlns
Классы
Теги
Свойства
Атрибуты
События
Атрибуты

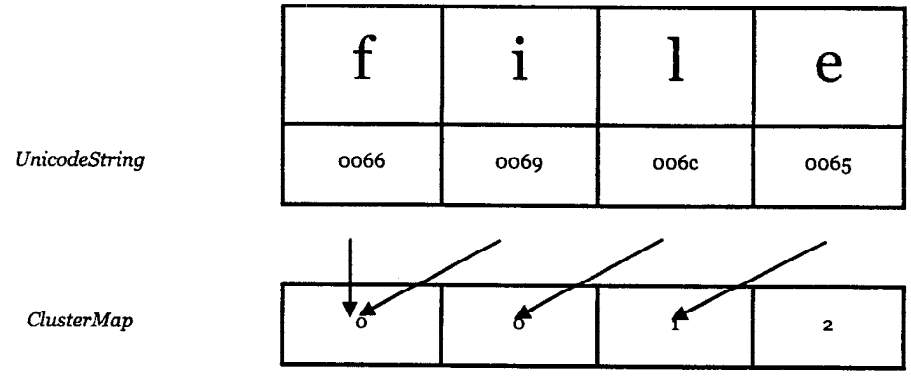
ФИГ.11


Блок выравнивания
Начало координат боковой базовой линии
Черный блок
Надстрочный интервал для Windows (OS/2)
Начало координат базовой линии слева направо
Начало координат базовой линии справа налево
Базовая линия для западных языков
Подстрочный интервал для Windows
Полная ширина (hmtx)
Начало координат черного блока
Черный блок
Начало координат базовой линии слева направо
Начало координат базовой линии справа налево
Вертикальное смещение базовой линии
Левый отступ
Правый отступ
Полная ширина
Левый отступ
Правый отступ
Полная ширина
Начало координат боковой базовой линии
Черный блок
Верхний отступ
Продвижение
Нижний отступ
Центральная базовая линия

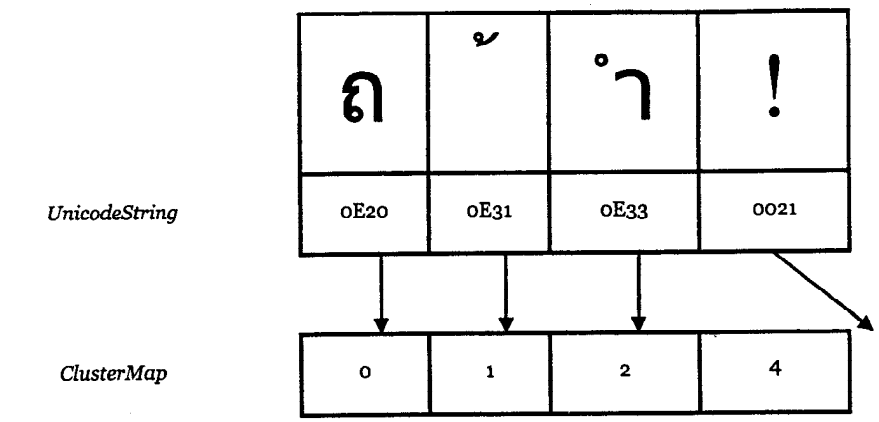
ФИГ.12



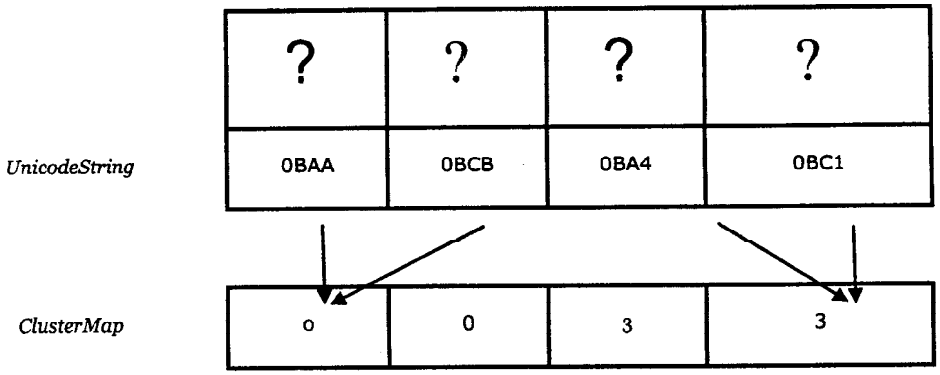
ФИГ.13



ФИГ.14



ФИГ.15



ФИГ.16