(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2012년01월31일
(11) 등록번호 10-1109280
(24) 등록일자 2012년01월17일

(51) Int. Cl.
G06F 17/21 (2006.01) *G06F 17/00* (2006.01)
(21) 출원번호 10-2005-7010458
(22) 출원일자(국제출원일자) 2004년07월22일
심사청구일자 2009년07월16일
(85) 번역문제출일자 2005년06월09일
(65) 공개번호 10-2007-0055303
(43) 공개일자 2007년05월30일
(86) 국제출원번호 PCT/US2004/023369
(87) 국제공개번호 WO 2005/111847
국제공개일자 2005년11월24일
(30) 우선권주장
10/836,608 2004년04월30일 미국(US)
(56) 선행기술조사문헌
KR1020030044907 A
전체 청구항 수 : 총 27 항

(73) 특허권자
마이크로소프트 코포레이션
미국 워싱턴주 (우편번호 : 98052) 레드몬드 원 마이크로소프트 웨이
(72) 발명자
슈, 안드레이
미국 98052 워싱턴주 레드몬드 노스이스트 40번 스트리트 큐36115606
듀니에츠, 제리
미국 98103 워싱턴주 시애틀 맥길브라 블루버드. 이. 439
(뒷면에 계속)
(74) 대리인
제일특허법인

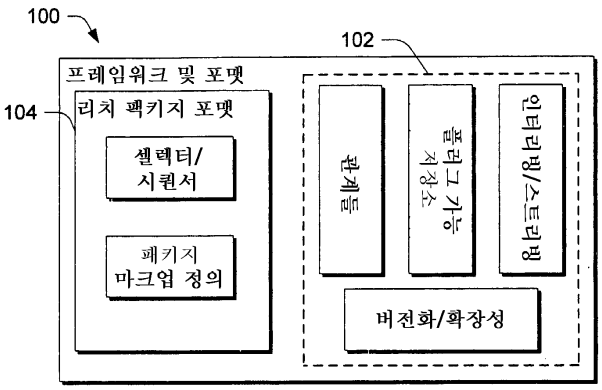
심사관 : 이종익

(54) 선택 가능 및/또는 시퀀스 가능 파트들을 갖는다큐먼트들을 정의하기 위한 방법 및 시스템

(57) 요 약

모듈러 콘텐츠 프레임워크 및 다큐먼트 포맷 방법들 및 시스템들이 기술된다. 프레임워크 및 포맷은 다큐먼트 중심 콘텐츠를 구성, 패키지화, 분산 및 렌더링하기 위한 빌딩 블록 집합을 정의한다. 빌딩 블록은 소프트웨어 및 하드웨어 시스템들이 다큐먼트들을 신뢰성 있고 일관되게 생성, 교환 및 디스플레이할 수 있게 해주는 다큐먼트 포맷에 대한 플랫폼 독립 프레임워크를 정의한다. 프레임워크 및 포맷은 유연하고 확장 가능한 방식으로 설계되어 왔다. 일반 프레임워크 및 포맷 외에, *리치 패키지* 포맷(reach package format)이라고 알려진 특정 포맷이 일반 프레임워크를 사용해서 정의된다. *리치 패키지* 포맷은 페이지가 매겨진 다큐먼트들을 저장하기 위한 포맷이다. 리치 패키지의 콘텐츠는 광범위한 환경들에서 광범위한 시나리오들에 걸쳐 디바이스들 및 어플리케이션들 간의 최대 충실도로 표시 또는 인쇄될 수 있다.

대 표 도 - 도1



(72) 발명자

킹, 조

미국 98103 워싱턴주 시애틀 휘트먼 애비뉴 엔. 4117

폴로크, 조쉬

미국 98033 워싱턴주 커클랜드 커클랜드 크레센트, 아파트 비203720

주, 웨이

미국 98155 워싱턴주 쇼어라인 8번 애비뉴 노스이스트 15812

포엘, 올리버 에이치.

미국 98040 워싱턴주 머서 아일랜드 더블유. 머서 웨이 5413

쉬쓰, 사르자나

미국 98052 워싱턴주 레드몬드 156번 노스이스트, 아파트 34 4850

온스테인, 데이비드

미국 98115 워싱턴주 시애틀 21번 애비뉴 노스이스트 8106

에머슨, 다니엘 에프.

미국 98052 워싱턴주 레드몬드 올드 레드몬드 로드 아파트 씨1107001

특허청구의 범위

청구항 1

문서를 정의하는 패키지를 구축하는 단계 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타내는 마크업(markup)으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 복수개의 파트들 각각은 텍스트 타입 및 이미지 타입을 포함하는 그룹 중 적어도 하나를 포함하는 관련 컨텐츠 타입(associated content type)을 갖고, 상기 서로 다른 표현들 중 적어도 일부는 동일한 컨텐츠를 포함하고, 상기 패키지는 복수개의 관계 요소들(relationship elements)을 더 포함하고, 각 관계 요소는, 상기 복수개의 파트들 중 하나와 관련되고, 상기 복수개의 파트들 중 하나와 상기 복수개의 파트들 중 하나 이상의 다른 파트들과의 관계(relationship)를 정의하며, 상기 관계가 그 관련 파트의 컨텐츠에 독립적으로 발견 가능하도록(discoverable) 함 -,

상기 패키지에 선택기 타입(selector type) 및 시퀀스 타입(sequence type)을 포함하는 파트들의 그룹 중 적어도 하나를 포함하는 하나 이상의 구성 파트들(composition parts)을 포함시키는 단계 - 상기 선택기 타입의 구성 파트는 복수개의 다른 파트들 중에서의 선택을 상기 복수개의 다른 파트들과 관련된 컨텐츠 타입에 기초하여 실행하고, 상기 시퀀스 타입의 구성 파트는 상기 복수개의 다른 파트들의 특정 컨텐츠 타입을 인식하는 소프트웨어가 이용가능한지에 적어도 부분적으로 기초하여 상기 복수개의 다른 파트들을 시퀀싱(sequence)함 -,

상기 하나 이상의 구성 파트들을 식별하기 위해 상기 패키지를 처리하는 단계, 및

상기 구성 파트들 중 하나 이상과 관련된 액션을 수행하는 단계

를 포함하는 방법.

청구항 2

제1항에 있어서,

상기 하나 이상의 구성 파트들을 포함시키는 단계는, 상기 하나 이상의 구성 파트들을 XML 요소들(XML elements)로 표현하는 단계를 포함하는, 방법.

청구항 3

컴퓨터에 제1항의 방법을 실행시키기 위한 컴퓨터 판독 가능 명령들을 기록한 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 4

문서를 정의하는 패키지를 구축하기 위한 수단 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타내는 마크업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 복수개의 파트들 각각은 텍스트 타입 및 이미지 타입을 포함하는 그룹 중 적어도 하나를 포함하는 관련 컨텐츠 타입을 갖고, 상기 서로 다른 표현들 중 적어도 일부는 동일한 컨텐츠를 포함하고, 상기 패키지는 복수개의 관계 요소들을 더 포함하고, 각 관계 요소는, 상기 복수개의 파트들 중 하나와 관련되고, 상기 복수개의 파트들 중 하나와 상기 복수개의 파트들 중 하나 이상의 다른 파트들과의 관계를 정의하며, 상기 관계가 그 관련 파트의 컨텐츠에 독립적으로 발견 가능하도록 함 -,

상기 패키지에 선택기 타입 및 시퀀스 타입을 포함하는 파트들의 그룹 중 적어도 하나를 포함하는 하나 이상의 구성 파트들을 포함시키기 위한 수단 - 상기 선택기 타입의 구성 파트는 복수개의 다른 파트들 중에서의 선택을 상기 복수개의 다른 파트들과 관련된 컨텐츠 타입에 기초하여 실행하고, 상기 시퀀스 타입의 구성 파트는 상기 복수개의 다른 파트들의 특정 컨텐츠 타입을 인식하는 소프트웨어가 이용가능한지에 적어도 부분적으로 기초하여 상기 복수개의 다른 파트들을 시퀀싱함 -,

상기 하나 이상의 구성 파트들을 식별하기 위해 상기 패키지를 처리하기 위한 수단, 및

상기 구성 파트들 중 하나 이상과 관련된 액션을 수행하기 위한 수단을 포함하는 컴퓨팅 시스템.

청구항 5

문서를 정의한 패키지를 수신하는 단계 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타낼 수 있는 마크

업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 서로 다른 표현들 중 적어도 일부는 동일한 컨텐츠를 포함하고, 상기 패키지는 하나 이상의 구성 파트들을 포함하고, 상기 구성 파트들은 문서 파트 선택 또는 문서 파트 시퀀싱 중 적어도 하나를 실행할 수 있으며, 상기 패키지는 복수개의 관계 요소들을 더 포함하고, 각 관계 요소는, 상기 복수개의 파트들 중 하나와 관련되고, 소스 파트와 타겟 파트 간의 관계를 정의하고, 상기 관계가 관련 파트에 독립적으로 발견가능하도록 하고, 각 관계 요소는 상기 소스 파트가 관련된 상기 타겟 파트를 나타내는 타겟 속성(target attribute) 및 상기 관계의 타입 또는 성격(nature)을 나타내는 이름 속성(name attribute)을 포함함 -;

상기 하나 이상의 구성 파트들을 식별하기 위해 상기 패키지를 처리하는 단

계; 및

상기 하나 이상의 구성 파트들에 관련된 액션을 수행하는 단계

를 포함하는 컴퓨터에서 실행되는 방법.

청구항 6

제5항에 있어서,

상기 하나 이상의 구성 파트들은 XML 요소들로 표현되는, 방법.

청구항 7

제5항에 있어서,

상기 복수개의 파트들의 모든 파트는 각각 관련 타입(associated type)을 갖고, 상기 하나 이상의 구성 파트들은 파트 타입에 기초하여 수행하도록 구성된, 방법.

청구항 8

제7항에 있어서,

상기 구성 파트들 중 적어도 일부는 문서 파트 선택을 수행하도록 구성되고 선택이 이루어지는 파트들과 관련된 컨텐츠 타입들에 기초하여 선택을 수행할 수 있는, 방법.

청구항 9

제8항에 있어서,

컨텐츠 타입 선택은, 특정 컨텐츠 타입을 인식하는 소프트웨어가 이용가능한지에 적어도 부분적으로 기초하는, 방법.

청구항 10

컴퓨터에 제9항의 방법을 실행시키기 위한 컴퓨터 판독 가능 명령들을 기록한 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 11

문서를 정의한 패키지를 수신하기 위한 수단 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타낼 수 있는 마크업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 서로 다른 표현들 중 적어도 일부는 동일한 컨텐츠를 포함하고, 상기 패키지는 하나 이상의 구성 파트들을 포함하고, 상기 구성 파트들은 문서 파트 선택 또는 문서 파트 시퀀싱 중 적어도 하나를 실행할 수 있으며, 상기 패키지는 복수개의 관계 요소들을 더 포함하고, 각 관계 요소는, 상기 복수개의 파트들 중 하나와 관련되고, 소스 파트와 타겟 파트 간의 관계를 정의하고, 상기 관계가 관련 파트에 독립적으로 발견가능하도록 하고, 각 관계 요소는 상기 소스 파트가 관련된 상기 타겟 파트를 나타내는 타겟 속성(target attribute) 및 상기 관계의 타입 또는 성격을 나타내는 이름 속성(name attribute)을 포함함 -;

상기 하나 이상의 구성 파트들을 식별하기 위해 상기 패키지를 처리하기 위한 수단; 및

상기 하나 이상의 구성 파트들에 관련된 액션을 수행하기 위한 수단을 포함하고,

상기 복수개의 파트들의 모든 파트는 각각 관련 타입(associated type)을 갖고, 상기 하나 이상의 구성 파트들은 파트 타입에 기초하여 수행하도록 구성되고,

상기 구성 파트들 중 적어도 일부는 문서 파트 선택을 수행하도록 구성되고 선택이 이루어지는 파트들과 관련된 컨텐츠 타입들에 기초하여 선택을 수행할 수 있고,

컨텐츠 타입 선택은, 특정 컨텐츠 타입을 인식하는 소프트웨어가 이용가능한지에 적어도 부분적으로 기초하는 컴퓨팅 시스템.

청구항 12

컴퓨터에 제5항의 방법을 실행시키기 위한 컴퓨터 판독 가능 명령들을 기록한 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 13

문서를 정의한 패키지를 수신하기 위한 수단 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타낼 수 있는 마크업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 서로 다른 표현들 중 적어도 일부는 동일한 컨텐츠를 포함하고, 상기 패키지는 하나 이상의 구성 파트들을 포함하고, 상기 구성 파트들은 문서 파트 선택 또는 문서 파트 시퀀싱 중 적어도 하나를 실행할 수 있으며, 상기 패키지는 복수개의 관계 요소들을 더 포함하고, 각 관계 요소는, 상기 복수개의 파트들 중 하나와 관련되고, 소스 파트와 타겟 파트 간의 관계를 정의하고, 상기 관계가 관련 파트에 독립적으로 발견가능하도록 하고, 각 관계 요소는 상기 소스 파트가 관련된 상기 타겟 파트를 나타내는 타겟 속성 및 상기 관계의 타입 또는 성격을 나타내는 이름 속성을 포함함 -;

상기 하나 이상의 구성 파트들을 식별하기 위해 상기 패키지를 처리하기 위한 수단; 및

상기 하나 이상의 구성 파트들에 관련된 액션을 수행하기 위한 수단을 포함하는 컴퓨팅 시스템.

청구항 14

문서를 정의하는 패키지를 구축하는 단계 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타낼 수 있는 마크업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 서로 다른 표현들 중 적어도 일부는 동일한 컨텐츠를 포함하고, 각 파트는 관련 타입을 갖고, 상기 패키지는 복수개의 관계 요소들을 더 포함하고, 각 관계 요소는, 상기 복수개의 파트들 중 하나와 관련되고, 상기 복수개의 파트들 중 하나와 상기 복수개의 파트들 중 하나 이상의 다른 파트들과의 관계를 정의하고, 상기 관계가 관련 파트에 독립적으로 발견될 수 있도록 하며, 상기 관계 요소들 중 적어도 하나는 소스 파트와 타겟 파트과의 관계를 상기 소스 파트나 상기 타겟 파트를 수정하지 않고 생성함 -,

상기 패키지에 하나 이상의 구성 파트들을 포함시키는 단계 - 상기 구성 파트들 중 일부는 문서 파트 선택 또는 문서 파트 시퀀싱 중 적어도 하나를 실행할 수 있고, 문서 파트 선택을 실행하는 구성 파트들은 언어 타입, 컬러 타입, 페이지 크기 타입 또는 컨텐츠 타입 중 적어도 하나에 기초하여 상기 문서 파트 선택을 실행함 -,

상기 구성 파트들 중 하나 이상을 식별하기 위해 상기 패키지를 처리하는 단계, 및

상기 구성 파트들 중 하나 이상과 관련된 액션을 수행하는 단계

를 포함하는 컴퓨터에서 실행되는 방법.

청구항 15

제14항에 있어서,

상기 하나 이상의 구성 파트들을 포함시키는 단계는, 상기 하나 이상의 구성 파트들을 XML 요소들로서 표현하는 단계를 포함하는, 방법.

청구항 16

제14항에 있어서,

컨텐츠 타입 선택은 특정 컨텐츠 타입을 인식하는 소프트웨어가 이용가능한지에 적어도 부분적으로 기초하는, 방법.

청구항 17

컴퓨터에 제16항의 방법을 실행시키기 위한 컴퓨터 판독 가능 명령들을 기록한 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 18

문서를 정의하는 패키지를 구축하기 위한 수단 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타낼 수 있는 마크업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 서로 다른 표현들 중 적어도 일부는 동일한 컨텐츠를 포함하고, 각 파트는 관련 타입을 갖고, 상기 패키지는 복수개의 관계 요소들을 더 포함하고, 각 관계 요소는, 상기 복수개의 파트들 중 하나와 관련되고, 상기 복수개의 파트들 중 하나와 상기 복수개의 파트들 중 하나 이상의 다른 파트들과의 관계를 정의하고, 상기 관계가 관련 파트에 독립적으로 발견될 수 있도록 하며, 상기 관계 요소들 중 적어도 하나는 소스 파트와 타겟 파트과의 관계를 상기 소스 파트나 상기 타겟 파트를 수정하지 않고 생성함 -,

상기 패키지에 하나 이상의 구성 파트들을 포함시키기 위한 수단 - 상기 구성 파트들 중 일부는 문서 파트 선택 또는 문서 파트 시퀀싱 중 적어도 하나를 실행할 수 있고, 문서 파트 선택을 실행하는 구성 파트들은 언어 타입, 컬러 타입, 페이지 크기 타입 또는 컨텐츠 타입 중 적어도 하나에 기초하여 상기 문서 파트 선택을 실행함 -,

상기 구성 파트들 중 하나 이상을 식별하기 위해 상기 패키지를 처리하기 위한 수단, 및

상기 구성 파트들 중 하나 이상과 관련된 액션을 수행하기 위한 수단을 포함하고,

컨텐츠 타입 선택은 특정 컨텐츠 타입을 인식하는 소프트웨어가 이용가능한지에 적어도 부분적으로 기초하는 컴퓨팅 시스템.

청구항 19

컴퓨터에 제14항의 방법을 실행시키기 위한 컴퓨터 판독 가능 명령들을 기록한 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 20

문서를 정의하는 패키지를 구축하기 위한 수단 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타낼 수 있는 마크업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 서로 다른 표현들 중 적어도 일부는 동일한 컨텐츠를 포함하고, 각 파트는 관련 타입을 갖고, 상기 패키지는 복수개의 관계 요소들을 더 포함하고, 각 관계 요소는, 상기 복수개의 파트들 중 하나와 관련되고, 상기 복수개의 파트들 중 하나와 상기 복수개의 파트들 중 하나 이상의 다른 파트들과의 관계를 정의하고, 상기 관계가 관련 파트에 독립적으로 발견될 수 있도록 하며, 상기 관계 요소들 중 적어도 하나는 소스 파트와 타겟 파트과의 관계를 상기 소스 파트나 상기 타겟 파트를 수정하지 않고 생성함 -,

상기 패키지에 하나 이상의 구성 파트들을 포함시키기 위한 수단 - 상기 구성 파트들 중 일부는 문서 파트 선택 또는 문서 파트 시퀀싱 중 적어도 하나를 실행할 수 있고, 문서 파트 선택을 실행하는 구성 파트들은 언어 타입, 컬러 타입, 페이지 크기 타입 또는 컨텐츠 타입 중 적어도 하나에 기초하여 상기 문서 파트 선택을 실행함 -,

상기 구성 파트들 중 하나 이상을 식별하기 위해 상기 패키지를 처리하기 위한 수단, 및

상기 구성 파트들 중 하나 이상과 관련된 액션을 수행하기 위한 수단을 포함하는 컴퓨팅 시스템.

청구항 21

문서를 정의하는 패키지를 수신하는 단계 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타낼 수 있는 마크업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 서로 다른 표현들의 적어도 일부는 동일한 컨텐츠를 포함하고, 각 파트는 관련 타입을 갖고, 상기 패키지는 하나 이상의 구성 파트들을 포함하며, 상기 구성 파트들은 문서 파트 선택 또는 문서 파트 시퀀싱 중 적어도 하나를 실행할 수 있으며, 문서 파트 선택을 실행하는 구성 파트들은 언어 타입, 컬러 타입, 페이지 크기 타입 또는 컨텐츠 타입 중 적어도 하나에 기초하여 상기 문서 파트 선택을 실행할 수 있으며, 상기 패키지는 파트 이름 확장자들(part name extensions)로

부터 컨텐츠 타입들로의 디폴트 맵핑들을 정의하는 하나 이상의 디폴트 요소, 및 상기 디폴트 맵핑들과 일치하지 않는 파트들에 대한 컨텐츠 타입들을 지정하는 하나 이상의 오버라이드(override) 요소를 포함하는 타입 스트림(types stream)을 더 포함함 -,

상기 하나 이상의 구성 파트들을 식별하기 위해 상기 패키지를 처리하는 단계,

상기 하나 이상의 구성 파트들과 관련된 액션을 실행하는 단계, 및

상기 문서를 구성하는 상기 복수개의 파트들 중 두 개 이상의 파트들 간의 하나 이상의 관계를, 상기 문서를 구성하는 상기 복수개의 파트들 중 상기 두 개 이상의 파트들 내의 컨텐츠를 조사하지 않고, 발견하는 단계

를 포함하는 컴퓨터에서 실행되는 방법.

청구항 22

제21항에 있어서,

상기 하나 이상의 구성 파트들은 XML 요소들로 표현되는, 방법.

청구항 23

제21항에 있어서,

상기 구성 파트들 중 하나 이상은 문서 파트 선택을 실행할 수 있고, 컨텐츠 타입 선택은 특정 컨텐츠 타입을 인식하는 소프트웨어가 이용가능한지에 적어도 부분적으로 기초하는, 방법.

청구항 24

컴퓨터에 제23항의 방법을 실행시키기 위한 컴퓨터 판독 가능 명령들을 기록한 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 25

문서를 정의하는 패키지를 수신하기 위한 수단 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타낼 수 있는 마크업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 서로 다른 표현들의 적어도 일부는 동일한 컨텐츠를 포함하고, 각 파트는 관련 타입을 갖고, 상기 패키지는 하나 이상의 구성 파트들을 포함하며, 상기 구성 파트들은 문서 파트 선택 또는 문서 파트 시퀀싱 중 적어도 하나를 실행할 수 있으며, 문서 파트 선택을 실행하는 구성 파트들은 언어 타입, 컬러 타입, 페이지 크기 타입 또는 컨텐츠 타입 중 적어도 하나에 기초하여 상기 문서 파트 선택을 실행할 수 있으며, 상기 패키지는 파트 이름 확장자들(part name extensions)로부터 컨텐츠 타입들로의 디폴트 맵핑들을 정의하는 하나 이상의 디폴트 요소, 및 상기 디폴트 맵핑들과 일치하지 않는 파트들에 대한 컨텐츠 타입들을 지정하는 하나 이상의 오버라이드(override) 요소를 포함하는 타입 스트림(types stream)을 더 포함함 -,

상기 하나 이상의 구성 파트들을 식별하기 위해 상기 패키지를 처리하기 위한 수단,

상기 하나 이상의 구성 파트들과 관련된 액션을 실행하기 위한 수단, 및

상기 문서를 구성하는 상기 복수개의 파트들 중 두 개 이상의 파트들 간의 하나 이상의 관계를, 상기 문서를 구성하는 상기 복수개의 파트들 중 상기 두 개 이상의 파트들 내의 컨텐츠를 조사하지 않고, 발견하기 위한 수단을 포함하고,

상기 구성 파트들 중 하나 이상은 문서 파트 선택을 실행할 수 있고, 컨텐츠 타입 선택은 특정 컨텐츠 타입을 인식하는 소프트웨어가 이용가능한지에 적어도 부분적으로 기초하는 컴퓨팅 시스템.

청구항 26

컴퓨터에 제21항의 방법을 실행시키기 위한 컴퓨터 판독 가능 명령들을 기록한 하나 이상의 컴퓨터 판독가능 기록매체.

청구항 27

문서를 정의하는 패키지를 수신하기 위한 수단 - 상기 패키지는, 상기 문서의 서로 다른 표현들을 나타낼 수 있

는 마크업으로 기술되고 상기 문서를 구성하는 복수개의 파트들을 포함하고, 상기 서로 다른 표현들의 적어도 일부는 동일한 컨텐츠를 포함하고, 각 파트는 관련 타입을 갖고, 상기 패키지는 하나 이상의 구성 파트들을 포함하며, 상기 구성 파트들은 문서 파트 선택 또는 문서 파트 시퀀싱 중 적어도 하나를 실행할 수 있으며, 문서 파트 선택을 실행하는 구성 파트들은 언어 타입, 컬러 타입, 페이지 크기 타입 또는 컨텐츠 타입 중 적어도 하나에 기초하여 상기 문서 파트 선택을 실행할 수 있으며, 상기 패키지는 파트 이름 확장자들로부터 컨텐츠 타입들로의 디폴트 맵핑들을 정의하는 하나 이상의 디폴트 요소, 및 상기 디폴트 맵핑들과 일치하지 않는 파트들에 대한 컨텐츠 타입들을 지정하는 하나 이상의 오버라이드 요소를 포함하는 타입 스트림을 더 포함함 -,

상기 하나 이상의 구성 파트들을 식별하기 위해 상기 패키지를 처리하기 위한 수단,

상기 하나 이상의 구성 파트들과 관련된 액션을 실행하기 위한 수단, 및

상기 문서를 구성하는 상기 복수개의 파트들 중 두 개 이상의 파트들 간의 하나 이상의 관계를, 상기 문서를 구성하는 상기 복수개의 파트들 중 상기 두 개 이상의 파트들 내의 컨텐츠를 조사하지 않고, 발견하기 위한 수단을 포함하는 컴퓨팅 시스템.

청구항 28

삭제

청구항 29

삭제

청구항 30

삭제

청구항 31

삭제

청구항 32

삭제

청구항 33

삭제

명 세 서

기 술 분 야

[0001] 본 발명은 콘텐츠 프레임워크(content framework) 및 다큐먼트 포맷(document format)과, 이를 모두 활용할 수 있는 관련 방법 및 시스템에 관한 것이다.

배 경 기 술

[0002] 오늘날, 일반적으로 콘텐츠를 표현하는 다수의 상이한 타입들의 콘텐츠 프레임워크들이 있으며, 다양한 타입들의 다큐먼트들을 포맷하는 다수의 상이한 타입들의 다큐먼트 포맷들이 있다. 상기 프레임워크들 및 포맷들은 각각 관련 다큐먼트를 작성, 생성, 처리 또는 소비하기 위해 자신의 관련 소프트웨어를 필요로 한다. 적합한 디바이스에 설치된 특정 관련 소프트웨어를 가진 자들의 경우, 관련 다큐먼트들을 작성, 생성, 처리 또는 소비하는데는 별다른 문제가 없다. 적합한 소프트웨어를 갖지 못한 자들의 경우, 관련 다큐먼트들을 작성, 생성, 처리 또는 소비하는 것이 통상 불가능하다.

[0003] 이러한 단점에 대하여, 다큐먼트의 생성 및 소비가 있는 한, 어디에서나 사용할 수 있을 것(ubiquity)이 요구된다.

발명의 상세한 설명

[0004] 모듈러 콘텐츠 프레임워크 및 다큐먼트 포맷 방법 및 시스템이 기술된다. 프레임워크 및 포맷은 다큐먼트 중심 콘텐츠를 구성, 패키지, 분산 및 렌더링하기 위한 빌딩 블록 집합을 정의한다. 빌딩 블록은 소프트웨어 및 하드웨어 시스템들이 다큐먼트들을 신뢰성 있고 일관되게 생성, 교환 및 표시 할 수 있게 해주는 다큐먼트 포맷에 대한 플랫폼에 의존하지 않는 프레임워크(Platform-independent Framework)를 정의한다. 프레임워크 및 포맷은 유연하고 확장 가능한 방식으로 설계되어 왔다.

[0005] 이러한 일반적인 프레임워크 및 포맷에 더하여, *리치 패키지(Reach Package)* 포맷이라고 하는 특정 포맷이 일반 프레임워크를 사용해서 정의된다. *리치 패키지* 포맷은 페이지가 매겨진 다큐먼트들을 저장하기 위한 포맷이다. 리치 패키지의 콘텐츠는 광범위한 환경들에서 광범위한 시나리오들에 걸쳐 디바이스들 및 어플리케이션들 사이에서 최대 충실도로 디스플레이 또는 인쇄될 수 있다.

실 시 예

[0027] **개요**

[0028] 본 명세서는 모듈러 콘텐츠 프레임워크 및 다큐먼트 포맷을 기술한다. 프레임워크 및 포맷은 다큐먼트 중심의 콘텐츠를 구성(composition), 패키지, 배포(distribution) 및 렌더링하기 위한 빌딩 블록 집합을 정의한다. 이러한 빌딩 블록은 소프트웨어 및 하드웨어 시스템들이 다큐먼트들을 신뢰성 있고 일관되게 생성, 교환 및 디스플레이할 수 있게 해주는 다큐먼트 포맷에 대한 플랫폼에 의존하지 않는 프레임워크(platform-independent framework)를 정의한다. 프레임워크 및 포맷은 융통성 있고 확장 가능한 방식으로 설계되어 왔다. 다양일 실시예들에서, 포함될 수 있는 콘텐츠 타입, 콘텐츠를 표현하는 방법, 또는 콘텐츠를 처리하기 위한 클라이언트들을 생성하는 플랫폼에 대한 제한은 없다.

[0029] 이러한 일반적인 프레임워크에 더하여, 일반적인 프레임워크를 사용하여 특정의 포맷이 정의된다. 이러한 포맷은 본 명세서에서 *리치 패키지(Reach Package)* 포맷이라고 칭하며, 페이지가 매겨지거나 미리 페이지가 매겨진 다큐먼트들을 저장하기 위한 포맷이다. 리치 패키지의 콘텐츠는 광범위한 환경들에서 광범위한 시나리오들에 걸쳐 디바이스들 및 어플리케이션들 사이에서 최대 충실도(full fidelity)로 표시 또는 인쇄될 수 있다.

[0030] 후술되는 프레임워크의 목적들 중 하나로서, 후술된 프레임워크 및 포맷에 따라 생성된 콘텐츠를 판독 또는 기록하는, 독립적으로 기록된 소프트웨어 및 하드웨어 시스템들의 상호 운용성(interoperability)을 보장하는 것이다. 상호 운용성을 달성하기 위해, 포맷은 콘텐츠를 판독 또는 기록하는 시스템들이 만족시켜야만 하는 형식적 요구 사항들을 정의한다.

[0031] 이하의 설명에서, "프레임워크"라는 제목 및 "리치 패키지 포맷"이라는 제목의 두 개의 주요 부분으로 구성하여 후술한다.

[0032] "프레임워크" 부분은 일례의 패키징 모델을 나타내며, 프레임워크 패키지들을 구성하는 각종 파트들 및 그 관계를 기술한다. 프레임워크 패키지의 기술적 메타데이터(descriptive metadata)의 사용에 대한 정보를 설명하며, 또한, 물리적 저장소(Physical Container)로 매핑하고, 프레임워크 마크업을 확장시키고, 프레임워크 버젼 메카니즘을 사용하는 절차들을 설명한다.

[0033] "리치 패키지 포맷" 부분은 *리치 패키지(Reach Package)*라고 하는 일 특정 타입의 프레임워크가 구축된 패키지(framework-built package)의 구조를 설명한다. 본 장은 또한 고정된 페이로드에 지정된 패키지 파트들을 기술하고, 리치 패키지 마크업 모델 및 드로잉 모델을 정의한다. 본 장은 일실시예의 샘플들과 함께 일실시예의 리치 마크업 요소들 및 속성들로 끝을 맺는다.

[0034] 이하의 설명의 하이 레벨 개요로서, 본 발명의 프레임워크 및 포맷의 양상들(100)을 도시한 도 1을 참조한다. 프레임워크의 특정 성분들(components, 102)의 예들이 도시되어 있으며, 리치 패키지 포맷의 특정 성분들(104)이 도시되어 있다.

[0035] 프레임워크(102)는, 이에 한하지 않지만, 관계 컴포넌트, 플러그 가능 저장소(pluggble container) 컴포넌트, 인터리브/스트리밍(interleaving/streaming) 컴포넌트 및 버젼화/확장 가능(versioning/extensibility) 컴포넌트를 포함하는 일실시예의 성분들을 구비하며, 상기 컴포넌트들 각각을 보다 상세히 후술한다. 리치 패키지 포맷(104)은 셀렉터/시퀀서(selector/sequencer) 컴포넌트 및 패키지 마크업 정의(package markup definition) 컴포넌트를 포함하는 성분들을 구비한다.

[0036] 이하의 설명에서, 상기 컴포넌트들이 프레임워크 및 패키지 포맷의 어디에 해당하는지에 대한 견지를 유지할 수

있도록, 도 1을 자주 참조하기로 한다.

[0037] **프레임워크(THE FRAMEWORK)**

[0038] 이하의 설명에서, 일반적인 프레임워크가 설명된다. 별도의 소제목로, "패키지 모델(Package Model)", "구성(Composition) 파트: 셀렉터 및 시퀀스", "기술적 메타데이터", "물리적 모델", "물리적 매핑" 및 "버젼화 및 확장 가능성(Versioning and Extensibility)"을 포함한다. 각각의 소제목는 하나 이상의 관련 부제(sub-heading)를 갖는다.

[0039] **패키지 모델(The Package Model)**

[0040] 본 장은 패키지 모델을 기술하며, 패키지 및 파트들, 드라이버들, 관계들, 패키지 관계들 및 시작 파트를 기술하는 소제목들을 포함한다.

[0041] **패키지 및 파트(Packages and Parts)**

[0042] 본 모델에서, 콘텐츠는 패키지 내에 유지된다. *패키지(package)*는 관련 *파트(part)*의 집합을 유지하는 논리적 개체(logical entity)이다. 패키지의 목적은 다큐먼트(또는 다른 타입의 콘텐츠)의 모든 조각(piece)들을 프로그래머 및 엔드유저가 작업하기 쉬운 하나의 객체(object)로 통합하는데 있다. 예를 들어, 도 2를 참조하면, 다큐먼트를 표현하는 XML 마크업 파트(202), 다큐먼트에서 사용되는 폰트를 기술하는 폰트 파트(204), 다큐먼트의 페이지들을 기술하는 다수의 페이지 파트들(206), 및 다큐먼트 내의 그림을 나타내는 그림 파트를 포함하는 다수의 파트들을 포함하는 다큐먼트를 유지하는 일례의 패키지(200)를 나타낸다. 다큐먼트를 표현하는 XML 마크업 파트(202)는 패키지의 전체 콘텐츠가 파싱(parse)될 필요 없이 쉽게 탐색 및 참조될 수 있다는 점에서 유익하다. 이는 보다 명백하게 후술된다.

[0043] 본 명세(specification)를 통해, 리더(reader, 또는 소비자라고도 함) 및 라이터(writer, 또는 생산자)의 개념이 소개되고 설명된다. 본 명세서에서 사용되는 리더는 모듈러 콘텐츠 포맷 기반의 파일들 또는 패키지들을 판독하는 개체(entity)를 말한다. 본 명세서에서 사용되는 라이터는 모듈러 콘텐츠 포맷 기반의 파일들 또는 패키지들을 기록하는 개체를 말한다. 일례로서, 도 3을 참조하면, 패키지를 생성하는 라이터 및 패키지를 판독하는 리더를 나타내고 있다. 통상, 라이터 및 리더는 소프트웨어로 구체화 된다. 적어도 하나 이상의 실시예에서, 패키지 생성 및 포맷과 관련된 프로세싱 오버헤드 및 복잡성의 대부분은 라이터에서 발생한다. 이는, 당업자들이 아는 바와 같이, 리더의 프로세싱 복잡성 및 오버헤드의 상당 부분을 일소시키는 것으로, 현재의 다수의 모델들의 출발점인 것이다. 이러한 실시형태는 이하에서 후술된다.

[0044] 하나 이상의 실시예에 따르면, 단일 패키지는 패키지에 유지되는 콘텐츠의 하나 이상의 표현들을 포함한다. 패키지는 본 명세서에서 저장소(container)라고 하는 단일 파일일 수 있다. 이는, 예를 들어, 다큐먼트의 모든 컴포넌트 피스(component pieces: 이미지, 폰트, 데이터 등)를 포함하는 다큐먼트들을 배포하는 편리한 방법을 엔드유저에게 제공한다. 패키지들은 종종 단일의 파일에 직접 대응하지만, 항상 그럴 필요는 없다. 패키지는 다양한 방법들로(예를 들어, 이에 한하지 않지만, 단일 파일에서, 루스 파일(loose file)의 집합에서, 데이터베이스에서, 네트워크 커넥션을 통한 순간적인 전송으로 등) 물리적으로 표현될 수도 있는 논리적 개체이다. 따라서, 저장소들이 패키지들을 유지하지만, 모든 패키지들이 저장소에 저장되는 것은 아니다.

[0045] 추상 모델(abstract model)은 임의의 물리적 스토리지 메카니즘과 무관하게 패키지들을 기술한다. 예를 들어, 추상 모델은 패키지가 위치한 물리 세계와 관련된 "파일들", "문자열(string)들", 또는 다른 물리적 용어들을 말하는 것이 아니다. 후술하는 바와 같이, 추상 모델은 사용자들이 다양한 물리적 포맷, 통신 프로토콜 등을 위한 드라이버들을 생성할 수 있게 해준다. 유추하여, 어플리케이션이 화상을 인쇄하기 원할 때, 프린터의 추상적 개념(특정 종류의 프린터를 인식하는 드라이버에 의해 표현됨)을 사용한다. 따라서, 어플리케이션은 세부적인 인쇄 장치 또는 인쇄 장치와 통신하는 방법을 알 필요가 없다.

[0046] 저장소는, 그렇지는 않다면 루스(loose)하고 접속분리된 파일들의 집합일수도 있는 것들에 대하여 다수의 장점들을 제공한다. 예를 들어, 유사한 컴포넌트들이 수집되고, 콘텐츠가 인덱싱 및 압축될 수도 있다. 또한, 컴포넌트들 간의 관계가 식별될 수도 있으며, 권한 관리(right management), 디지털 서명(digital signature), 암호화 및 메타데이터가 컴포넌트들에 적용될 수도 있다. 물론, 저장소들은 다른 특징들을 위해 사용될 수도 있으며 명백하게 상술되지 않은 다른 특징들을 구현할 수 있다.

[0047] 공통 파트 특성(Common Part Properties)

[0048] 도시된 실시예에서, 파트는 공통 특성들(예를 들어, 네임(name)) 및 바이트단위의 문자열(a stream of bytes)을

포함한다. 이는 파일 시스템의 파일이나 HTTP 서버의 리소스와 유사하다. 콘텐츠에 더하여, 각각의 파트는 몇몇 *공통 파트 속성(common part properties)*들을 갖는다. 이는 *네임* - 파트의 네임, 및 *콘텐츠 타입* - 파트에 저장된 콘텐츠의 타입을 포함한다. 파트들은 또한 후술되는 바와 같이, 하나 이상의 관련된 관계들을 가질 수도 있다.

[0049] 어떤 방식으로 파트를 언급할 필요가 있을 때마다 *파트 네임(part name)*들이 사용된다. 도시된 실시예에서, 네임들은 파일 시스템의 경로들 또는 URI의 경로들과 유사한 계층구조(hierarchy)로 구성된다. 파트 네임의 예들은 다음과 같다:

/document.xml

/tickets/ticket.xml

/images/march/summer.jpeg

/pages/page4.xml

[0050]

[0051] 전술한 바와 같이, 본 실시예에서, 파트 네임들은 이하의 특징들을 갖는다:

[0052] • 파트 네임들은 전형적인 파일 시스템의 파일 네임들과 유사하다.

[0053] • 파트 네임들은 포워드 슬래시('/')로 시작한다.

[0054] • 파일 시스템의 경로들 또는 URI의 경로들처럼, 파트 네임들은 일련의 디렉토리와 같은 네임들(상기 일례에서, 티켓(ticket), 이미지/마치(image/march) 및 페이지들(pages))에 의한 계층구조로 구성될 수 있다.

[0055] • 이러한 계층구조는 슬래시로 구분되는 세그먼트들로 구성된다.

[0056] • 네임의 최종 세그먼트는 전형적인 파일 시스템의 파일명과 유사하다.

[0057] 파트들, 특히, 파트 네임에 사용될 수 있는 유효한 문자들로 네임을 명명하는 규칙은 본 명세서에 기술된 프레임워크에 특정적임을 주목해야 한다. 이러한 파트 네임 규칙들은 인터넷-표준 URI 네이밍 규칙들에 기초한다. 본 실시예에 따르면, 본 실시예에서 파트 네임을 지정하는데 사용되는 문법은 RFC2396 규정(URI(Uniform Resource Identifiers: 일반 구문)의 Section 3.3 (경로 컴포넌트) 및 5(상대적 URI 참조)에 정의된 **abs_path**와 정확하게 일치한다.

[0058] 이하의 추가 제한 사항들이 유효한 파트 네임으로서 **abs_path**에 적용된다:

[0059] • Section 3(URI Syntactic Components) 및 3.4(Query Component)에 정의된 바와 같이, 질의 컴포넌트(Query Component)는 파트 네임에 적용될 수 없다.

[0060] • Segment 4.1(fragment identifier(프래그먼트 식별자))에 정의된 바와 같이, 프래그먼트 식별자는 파트 네임에 적용될 수 없다.

[0061] • 기존 파트의 파트 네임에 *("/" 세그먼트)를 첨부함으로써 생성된 네임을 갖는 임의의 파트를 갖는 것은 위법이다.

[0062] 파트 네임에 대한 문법을 설명한다:

```
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
```

[0063]

[0064] 패키지의 모든 파트들의 네임들의 세그먼트들은 트리를 형성하는 것으로 볼 수 있다. 이는 파일 시스템에서 발생하는 바와 유사한데, 트리의 리프(leaf)가 아닌 모든 노드들은 폴더들이며, 리프 노드(leaf node)들은 콘텐츠를 포함하는 실제 파일들이다. 이러한 네임 트리의 폴더와 유사한 노드들(즉, 리프가 아닌 노드들)은 패키지에서 파트들을 구성하는 것과 유사한 기능을 한다. 그러나, 상기 "폴더들"이 네이밍 계층구조(naming heirarchy)의 개념으로서만 존재함(지속적인 포맷(persistence format)에서 다른 표현(manifestation)을 갖지 않음)을 주목한다.

[0065] 파트 네임들은 "폴더" 레벨에 존재할 수 없다. 명백하게, 파트 네이밍 계층구조의 리프가 아닌 노드("폴더")들은 동일한 네임을 가진 파트 및 서브폴더를 포함할 수 없다.

[0066] 본 실시예에서, 모든 파트는 어떤 타입의 콘텐츠가 파트에 저장되는지를 식별하는 *콘텐츠 타입(content type)*을 갖는다. 콘텐츠 타입의 일례들은 이하를 포함한다:

image/jpeg

text/xml

[0067] text/plain; charset="us-ascii"

[0068] 콘텐츠 타입은 RFC2045(MIME: Multipurpose Internet Mail Extension)에 정의된 바와 같이, 도시된 프레임워크에서 사용된다. 자세하게는, 각각의 콘텐츠 타입은 미디어 타입(예를 들어, 텍스트), 서브타입(예를 들어, 플레인(plain)) 및 key=value 형태(예를 들어, charset="us-ascii")의 파라미터 옵션 집합을 포함하며; 다수의 파라미터들은 세미콜론에 의해 분리된다.

[0069] 파트 어드레싱(Part Addressing)

[0070] 종종 파트들은 다른 파트들에 대한 참조자를 포함한다. 간단한 일례로서, 두 파트(마크업 파일 및 이미지)를 포함하는 저장소를 가정해본다. 마크업 파일이 처리될 때, 관련 이미지가 식별되고 위치가 정해질 수 있도록, 마크업 파일은 이미지에 대한 참조자를 유지하고자 할 것이다. 콘텐츠 타입 및 XML 스키마의 설계자들은 상기 참조자들을 나타내는데 URI를 사용할 수도 있다. 이를 가능하게 하기 위해, 파트 네임 세계 및 URI 세계 간의 매핑이 정의될 필요가 있다.

[0071] 패키지의 URI 사용을 허용하기 위해, 특별한 URI 해석 규칙이 패키지 기반의 콘텐츠를 평가할 때 사용돼야만 한다: 패키지 자체는 URI 참조를 위한 "권한(authority)"으로서 취급되어야 하며, URI의 경로 컴포넌트는 패키지 내의 파트 네임 계층구조를 탐색(navigation) 하는데 사용된다.

[0072] 예를 들어, http://www.example.com/foo/something.package라는 패키지 URI가 제공되면, /abc/bar.xml에 대한 참조는 http://www.example.com/abc/bar.xml 라는 URI가 아니라, /abc/bar.xml 이라는 파트를 의미한다고 해석된다.

[0073] 저장소에서 하나의 파트로부터 다른 파트로의 참조자를 가질 필요가 있을 때 상대적 URI들이 사용돼야만 한다.

상대적 참조자들을 사용하는 것은, 크로스-파트 참조자들을 변경하지 않고, 저장소의 콘텐츠가 상이한 저장소(또는 예를 들어, 파일 시스템으로부터의 저장소로)에 함께 이동될 수 있도록 한다.

[0074] 파트로부터의 상대적 참조자들은 참조자를 포함하는 파트의 "베이스(base) URI"에 상대적으로 해석된다. 디폴트에 의해, 파트의 베이스 URI가 파트 네임이다.

[0075] 이하의 네임들을 갖는 파트들을 포함하는 저장소를 고려해본다:

/markup/page.xml
/images/picture.jpeg
/images/other_picture.jpeg

[0076]

[0077] "/markup/page.xml" 파트가 "../images/picture.jpeg"에 대한 URI 참조자를 포함하면, 본 참조자는 상기 규칙들에 따라 파트 네임 "/images/picture.jpeg"을 참조하는 것으로 해석되어야 한다.

[0078] 몇몇 콘텐츠 타입들은 콘텐츠에서 상이한 베이스를 지정함으로써 디폴트 베이스 URI를 오버라이드하는 방법을 제공한다. 오버라이드(override)가 존재하는 경우, 외부적으로 지정되는 기본 URI가 디폴트 대신 사용되어야 한다.

[0079] 때때로, 파트의 일부분 또는 지정 포인트를 "어드레싱(addressing)"하는 것이 유용하다. URI 세계에서, 프래그먼트 식별자가 사용된다[RFC2396 참조]. 저장소는 동일한 방법으로 메카니즘이 작용한다. 자세하게는, 프래그먼트는 어드레싱된 파트의 콘텐츠 타입의 문맥에서 파악되는 추가 정보를 포함하는 문자열이다. 예를 들어, 비디오 파일에서는, 프래그먼트는 하나의 프레임을 식별할 수 있으며, XML 파일에서는 xpath를 통한 XML 파일의 일부분을 식별할 수도 있다.

[0080] 프래그먼트 식별자는 어드레싱된 파트의 프래그먼트들을 식별하도록 파트를 어드레싱하는 URI와 관련해서 사용된다. 프래그먼트 식별자는 임의선택사항(optional)으로서, "#" 문자에 의해 URI로부터 분리된다. 이와 같이, 그래그먼트 식별자는 URI의 파트가 아니며, 종종 URI와 관련해서 사용된다.

[0081] 이하의 설명은 파트 네이밍에 대한 몇몇 안내를 제공하는데, 패키지 및 파트 네이밍 모델은 유연성이 있는 것이다. 이러한 유연성(flexibility)은 광범위한 프레임워크 패키지 어플리케이션들을 허용한다. 그러나, 다수의 상관없는 소프트웨어 시스템들이 서로 충돌하지 않고 패키지의 "자신들만의" 파트들을 조작할 수 있는 시나리오가 가능하도록 프레임워크가 설계됨을 인식하는 것이 중요하다. 이를 위하여, 가능하다면, 이를 가능케하는 특정한 가이드라인들이 제공된다.

[0082] 본 명세서에 제공된 가이드라인들은 파트 네이밍의 충돌 발생을 최소화하거나 적어도 감소시키고, 충돌이 발생할 때는 충돌을 처리하는 메카니즘을 기술한다. 패키지에서 파트들을 생성하는 라이터(writer)들은 패키지내의 기존 파트들과의 네이밍 충돌을 검출하고 처리하는 단계들을 포함해야 한다. 네임 충돌이 발생하는 이벤트에서는, 라이터들이 기존 파트들을 모르는 상태에서 교체할 수 없을 수도 있다.

[0083] 패키지가 단일의 라이터에 의해 조작되는 것이 보장되는 상황에서, 라이터는 이러한 가이드라인들로부터 벗어날 수도 있다. 그러나, 다수의 독립 라이터들이 패키지를 공유할 가능성이 있으면, 모든 라이터들은 이러한 가이드라인들을 따라야만 한다. 그러나, 어떠한 경우이든지 모든 라이터들이 상기 가이드라인들을 따를 것이 권고된다.

[0084] • 라이터들이 기존 저장소에 파트들을 추가할 때, 파트들을 루트에 직접 배치하거나 이미 존재하는 폴더에 배치하는 것 보다는, 네이밍 계층구조의 새로운 "폴더"에 추가하도록 할 것이 요구된다. 이러한 방법으로, 네임 충돌의 가능성은 파트 네임의 제1 세그먼트로 제한된다. 이러한 새로운 폴더 내에 생성된 파트들은 기존 파트들과 충돌할 위험 없이 네이밍될 수 있다.

[0085] • 폴더에 대한 "바람직한(preferred)" 네임이 이미 기존 파트에 의해 사용되는 경우, 라이터는 대체의 폴더 네임들을 선택하기 위한 몇몇 기법을 채택해야만 한다. (수 회의 실패 반복후, 가능하다면, GUID에 의존하여) 유용한 폴더 네임이 발견될 때까지 라이터들은 바람직한 네임에 디지트(digit)를 첨부하는 기법을 사용해야 한다.

[0086] • 이러한 방법의 결과로서, 리더들이 어디에나 가능한 파트 네임(magic part name) 또는 잘 알려진 파트 네

임을 통해 파트를 위치시키고자 시도해서는 안된다. 대신 라이터들은 생성하는 각각의 폴더내의 적어도 하나의 파트에 대한 패키지 관계를 생성하여야 한다. 리더들은 잘 알려진 네임들에 의존하기 보다는 파트들을 위치시키기 위해 상기 패키지 관계들을 사용하여야 한다.

[0087] • 일단, 리더가 (상술된 패키지 관계들 중 하나를 통해) 폴더내에서 적어도 하나의 파트를 발견하면, 다른 파트들을 찾기 위해 해당 폴더 내의 잘 알려진 파트 네임에 관하여 종래의 것(convention)을 사용할 수도 있다.

[0088] **드라이버(Drivers)**

[0089] 본 명세서에 기술된 파일 포맷은 상이한 어플리케이션들, 상이한 다큐먼트 타입들 등에 의해 사용될 수 있는데, 그 중 다수는 충돌하는 포맷, 충돌하는 용도를 가질 수도 있다. 하나 이상의 *드라이버*들은 파일 포맷의 차이, 통신 프로토콜의 차이 등과 같은 다양한 충돌들을 해결하는데 사용된다. 예를 들어, 상이한 파일 포맷들은 루스 파일들 및 복합 파일들을 포함하고, 상이한 통신 프로토콜들은 http, 네트워크, 및 무선 프로토콜들을 포함한다. 드라이버 그룹은 다양한 파일 포맷들 및 통신 프로토콜들을 단일 모델로 추상화한다. 다수의 드라이버들이 상이한 시나리오들, 상이한 고객 요구 사항들, 상이한 물리 구성들 등을 위해 제공될 수 있다.

[0090] **관계들(Relationships)**

[0091] 패키지내의 파트들은 해당 패키지내의 다른 파트들에 대한 참조자를 포함할 수도 있다. 그러나, 일반적으로, 이러한 참조자들은 파트의 콘텐츠 타입에 특정한 방법들로, 즉, 참조하는 파트들 내부에서 임의적인 마크업(markup)으로 또는 어플리케이션 지정 인코딩(application-specific encoding)으로 표현된다. 이는 상기 참조자를 포함하는 파트들의 콘텐츠 타입들을 인식하지 못하는 리더들로부터 파트들 간의 내부적 연결 관계를 효율적으로 은닉시킨다.

[0092] 공통 콘텐츠 타입들(예를 들어, 리치 패키지 섹션에 기술된 고정된 페이로드 마크업)에 대해서도, 리더는 다른 파트들에 대한 참조자들을 발견하고 해결하기 위해 파트의 모든 콘텐츠를 파싱(parsing)할 필요가 있다. 예를 들어, 한 번에 한 페이지씩 다큐먼트들을 인쇄하는 인쇄 시스템을 구현할 때, 특정 페이지에 포함된 그림 및 폰트들을 식별하는 것이 요구될 수도 있다. 기존 시스템들은 각각의 페이지에 대한 모든 정보를 파싱해야만 하는데, 이는 시간 소모적일 수 있으며, 또한, 각각의 페이지의 언어를 이해해야만 하는데, 이는 특정 디바이스 또는 리더(디바이스에 대하여 프로세서 파이프라인을 통과함에 따라 다큐먼트에 대한 중간 프로세싱을 실행하는 디바이스 또는 리더)에 따라서는 상황이 안 될 수도 있다. 대신, 본 명세서에 기술된 시스템 및 방법은 파트들 간의 관계를 식별하고 그러한 관계들의 자연적 특성을 기술하기 위해 관계들을 사용한다. 관계 언어는 간단하며, 리더들이 다수의 상이한 언어들을 알 필요 없이 관계들을 이해할 수 있도록 정의된다. 일 실시예에서, 관계들은 개별적인 파트들로서 XML로 표현된다. 각각의 파트는 파트가 소스인 관계들을 포함하는 관련된 관계 파트(relationship part)를 갖는다.

[0093] 예를 들어, 스프레드시트 어플리케이션은 상기 포맷을 사용하여 상이한 스프레드시트들을 파트들로서 저장한다. 스프레드시트 언어에 대해 모르는 어플리케이션도 스프레드시트와 관련된 다양한 관계들을 발견할 수 있다. 예를 들어, 어플리케이션은 스프레드시트의 이미지들 및 스프레드시트와 관련된 메타데이터를 발견할 수 있다. 일례의 관계 스키마(relationship schema)는 아래와 같다:

```
<?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>
```

[0094]

[0095] 이러한 스키마는 2개의 XML 요소(하나는 "관계들(relationships)", 하나는 "관계(relationship)"라 함)를 정의한다. 이러한 "관계" 요소는 본 명세서에 기술된 바와 같이 단일 관계를 기술하는데 사용되며, 이하의 속성(attribute)들을 갖는다: (1) "타겟(target)"; 소스 파트가 관련된 파트를 나타냄, (2) "네임(name)"; 관계의 타입 또는 근원성(nature)을 나타냄. "관계들" 요소는 0개 이상의 "관계" 요소들을 보유할 수 있도록 정의되며, 간단하게 "관계" 요소들을 함께 하나의 단위로 집합시키도록 기능한다.

[0096] 본 명세서에 기술된 시스템 및 방법은, "관계들"이라고 하는 상기 문제점들을 해결하기 위해 보다 상위 레벨의 메카니즘을 도입한다. 관계들은 패키지의 소스 파트와 타겟 파트 간의 연결(connection)의 종류를 나타내는 추가 방법을 제공한다. 관계들은 파트들의 콘텐츠를 보지 않고 직접 "발견 가능한" 파트들 간의 연결을 구성하는데 있어서, 콘텐츠-지정 스키마와 무관하며, 해결책에 있어서 보다 신속하다. 또한, 상기 관계들은 프로토콜에 대하여 독립적이다. 각종의 상이한 관계들이 특정 파트와 관련될 수도 있다.

[0097] 관계들은 이들을 변경시키지 않고, 파트들을 연관시키는 제2의 중요한 기능을 제공한다. 때로는, 이러한 정보는 "주석" 형태로서 기능하는데, "주석된" 파트의 콘텐츠 타입은 제공된 정보를 첨부하는 방법을 정의하지 않는다. 가능한 예로서는, 첨부된 기술적 메타데이터, 인쇄 티켓 및 실제의 주석을 포함한다. 결국, 몇몇 시나리오들은 명백하게 해당 파트를 변경하지 않고 기존 파트에 정보가 첨부될 것을 요구한다 - 예를 들어, 파트가 암호화되어 있지만 복호화될 수 없을 때, 또는, 파트가 디지털 서명되어 있으며 바뀌고 있을 때, 이는 서명을 무효화한다. 다른 일례에서, 사용자는 JEPG 이미지 파일에 주석을 첨부하기를 원할 수도 있다. JEPG 이미지 포맷은 현재 주석을 식별하기 위한 지원을 하지 않는다. 사용자 희망사항을 수용하기 위해 JEPG 포맷을 변경하는 것은 실용적이지 않다. 그러나, 본 명세서에 기술된 시스템 및 방법은 사용자가 JEPG 이미지 포맷을 변경하지 않고 JEPG 파일에 주석을 제공할 수 있게 해준다.

[0098] 일 실시예에서, 관계들은 관계 파트들에서 XML을 사용하여 표현된다. 하나 이상의 관계들의 소스인 저장소의 각각의 파트는 연관된 관계 파트를 갖는다. 이러한 관계 파트는 해당 소스 파트에 대한 관계들의 리스트를 보유한다(콘텐츠 타입 *application/PLACEHOLDER*를 사용해서 XML로 표현됨).

[0099] 도 4는 "스파인(spine)" 파트(402)(FixedPanel과 유사함)가 세 페이지들(406, 408 및 410)을 함께 결합하는 환경(400)을 나타낸다. 스파인에 의해 함께 결합된 페이지 집합은 연관된 "인쇄 티켓"(404)을 갖는다. 또한, 페이지 2는 자신의 인쇄 티켓(412)을 갖는다. 스파인 파트(402)로부터 인쇄 티켓(404)으로의 연결(connection) 및 페이지 2로부터 인쇄 티켓(412)으로의 연결은 관계들을 사용해서 표현된다. 도 4의 구성에서, 스파인 파트(402)는 이하의 일례에 도시된 바와 같이, 스파인을 ticket1에 연결하는 관계를 포함한 연관된 관계 파트를 갖는다.

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

[0100]

[0101] 관계들은 단일의 <Relationships> 요소에 내포된 <Relationship> 요소들을 사용해서 표현된다. 이러한 요소들은 http://mmcfrels(PLACEHOLDER) 네임스페이스에서 정의된다. 상기 일례의 스키마 및 관련 설명(예컨대, 관계들)을 참조하기로 한다.

[0102] 관계 요소는 이하의 추가 속성들을 갖는다:

[0103]

속성	필요성	의미
타겟	예	관계의 다른 엔드에 있는 파트를 나타내는 URI. 관련 URI들은 소스 파트와 관련해서 해석돼야만 한다.
네임	예	관계의 역할 또는 목적을 유일하게 정의하는 절대 URI.

[0104] 네임 속성은 실제 어드레스일 필요가 없다. 상이한 타입들의 관계들이 자신의 네임에 의해 식별된다. 이러한 네임들은 네임스페이스들이 XML 네임스페이스들에 대해 정의되는 방법과 동일한 방법으로 정의된다. 자세하게는, XML 네임스페이스와 마찬가지로, 비관계적 부류(non-coordinating parties)는 인터넷 도메인 네임 스페이스 뒤에 패턴화된 네임들을 사용해서 충돌하지 않는(non-conflicting) 관계 네임들을 안전하게 생성할 수 있다.

[0105] 관계들 파트(relationships part)는 다른 관계들에 참여하도록 허용되지 않는다. 그러나, 다른 모든 면에서(예를 들어, URI 어드레싱 가능하고, 오픈, 판독, 삭제 가능하다는 점 등에서) 제1 클래스 파트이다. 관계들은 통상 패키지 외부의 것은 가리키지 않는다. 관계 타겟들을 식별하는데 사용되는 URI들은 일반적으로 URI 스키마를 포함하지 않는다.

[0106] 파트 및 그 관련 관계 파트는 네이밍 관례(convention)에 의해 연결된다. 본 일례에서, 스파인에 대한 관계 파트는 /content/_rels/spine.xml.rels에 저장되고, 페이지 2에 대한 관계들은 /content/_rels/p2.xml.rels에 저장된다. 여기서, 두 개의 특별한 네이밍 관례들이 사용됨을 주목한다. 첫째, 네임 계층구조에서의 소정의 "폴더"의 몇몇 (다른) 파트에 대한 관계 파트는 (관계들을 식별하기 위해) _rels라고 하는 "서브-폴더"에 저장된다. 둘째, 이러한 관계 유지 파트의 네임은 고유 파트의 네임에 .rels 확장자를 부여함으로써 형성된다. 특정 실시예에서, 관계 파트들은 콘텐츠 타입 application/xml+relationshipsPLACEHOLDER의 것이다.

[0107] 관계는 두 파트들 간의 직접적인 연결을 나타낸다. 관계가 표현되는 방법으로 인해, 소스 파트들로부터 관계들을 이동하는 것이 효율적이다(임의의 주어진 파트에 대한 관계들 파트를 찾는 것은 성가신 것이기 때문이다). 그러나, 관계의 타겟으로부터 관계들을 다시 이동하는 것은 효율적이지 않다(하나의 파트에 대한 모든 관계들을 찾는 방법은 저장소의 모든 관계들을 조사하는 것이므로).

[0108] 관계의 백워드 이동을 가능하게 하기 위해, 새로운 관계가 다른 (이동 가능한) 방향을 나타내는데 사용된다. 이는 일 유형의 관계를 설계하는 설계자가 사용할 수 있는 모델링 기술이다. 상술된 일례에 이어서, ticket1이 첨부된 스파인을 찾는 능력이 중요하다면, 티겟에서 스파인에 연결하는 제2 관계가 사용된다:

In *content/_rels/p1.xml.rels*:

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

[0109]

[0110] **패키지 관계들(Package Relationships)**

[0111] "패키지 관계들"은 한 패키지내의 잘 알려진 파트들을 찾는데 사용된다. 이러한 방법은 패키지의 파트들을 검색하기 위해 네이밍 관례들을 의존하는 것을 피하고, 상이한 페이로드들내에서 동일한 파트 네임들 간의 충돌이 없음을 보장한다.

[0112] 패키지 관계들은 그 타겟은 파티이지만, 그 소스는 파트가 아닌 특수한 관계들이다. 소스는 전체로서 패키지이다. "알려진" 파트를 포함하는 것은, "알려진" 관계 네임을 실제로 포함하는 것으로, 해당 파트를 검색하는데 도움이 되는 것이다. 이는, 관계들이 비관계적 부류(non-coordinating parties)로 네이밍 될 수 있게 하는 잘 정의된 메카니즘이 있는 한편, 특정한 실시예들은 이러한 파트네임을 위한 메카니즘을 포함하지 않기 때문에 가능한 것이다 - 이러한 실시예들은 일련의 가이드라인에 의해 제한됨 -. 패키지 관계들은 패키지 관계들 파트

(package relationships part)에서 발견되며, 관계 파트들의 표준 네이밍 관례를 사용하여 네이밍된다. 따라서, "/_rels/.rels"로 네이밍된다.

[0113] 패키지 관계들 파트내의 관계들은 공지된 파트들을 찾는데 유용하다.

[0114] **시작 파트(The Start Part)**

[0115] 패키지-레벨, 공지된 파트의 일례는 패키지 "시작" 파트이다. 이는 패키지가 오픈될 때 통상 처리되는 파트이다. 이는 패키지에 저장된 다큐먼트 콘텐츠의 논리적 루트를 나타낸다. 패키지의 시작 파트는 알려진 패키지 관계에 따라서 위치된다. 일례에서, 이러한 관계는 이하의 네임을 갖는다: http://mmcf-start-part-PLACEHOLDER.

[0116] **구성 파트: 셀렉터 및 시퀀스(Composition Parts: Selector and Sequence)**

[0117] 설명하는 프레임워크는 파트들로부터 보다 고차수의 구조를 구축하는 두 개의 메카니즘(*셀렉터* 및 *시퀀스*)을 정의한다.

[0118] 셀렉터는 다수의 다른 파트들 중에서 "선택"하는 파트이다. 예를 들어, 셀렉터 파트는 다큐먼트의 영어 버전을 나타내는 파트와 다큐먼트의 불어 버전을 나타내는 파트 중에 "선택"할 수도 있다. 시퀀스는 다수의 다른 파트들을 "순차 배열(sequence)"하는 파트이다. 예를 들어, 시퀀스 파트는 두 파트들, 5 페이지 다큐먼트를 나타내는 파트, 및 10 페이지 다큐먼트를 나타내는 파트를 (선형적인 시퀀스로) 결합할 수도 있다.

[0119] 이러한 두 타입의 *구성* 파트들(시퀀스 및 셀렉터) 및 상기 파트들을 조합하기 위한 규칙들은 *구성 모델(composition model)*을 포함한다. 구성 파트들은 다른 구성 파트들을 구성할 수 있어서, 예를 들어, 두 구성들 중에서 선택하는 셀렉터를 가질 수 있다. 일례로서, 영어 표현 및 불어 표현을 둘 다 포함하는 재무 보고서의 일례를 나타낸 도 5를 고려해본다. 상기 표현들 각각은 개요 파트(커버 페이지) 및 재무상태 파트(스프레드시트)로 더 구성된다. 본 일례에서, 셀렉터(500)는 보고서의 영어 표현 및 불어 표현 중에서 선택한다. 영어 표현이 선택되면, 시퀀스(502)는 영어 개요 파트(506)를 영어 재무 파트(508)와 시퀀스한다. 대안으로서, 불어 표현이 선택되면, 시퀀스(504)는 불어 개요 파트(510)를 불어 재무 파트(512)와 시퀀스한다.

[0120] **구성 파트 XML(Composition Part XML)**

[0121] 본 실시예에서, 컴포지션 파트들은 모두 공통적인 구성 네임스페이스로부터 도출되는, 소수의 XML 요소들을 사용하여 기술된다. 일례로서, 다음을 고려해 본다:

Element: <selection>
Attributes: None
Allowed Child Elements: <item>

Element: <sequence>
Attributes: None
Allowed Child Elements: <item>

Element: <item>
Attributes: Target – the part name of a part in the composition

[0122]

[0123] 일례로서, 상술된 도 5의 일례들에 대한 XML은 다음과 같다:

MainDocument.XML

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

EnglishRollup.XML

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

FrenchRollup.XML

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

[0124]

[0125] XML에서, MainDocument.xml은 패키지의 전체 파트를 나타내고, "item" 태그에 의해 묶여진 상이한 아이템들, 즉, "EnglishRollup.xml" 및 "FrenchRollup.xml" 중에서 "selection" 태그를 통해 선택이 이루어짐을 나타낸다.

[0126] EnglishRollup.xml 및 FrenchRollup.xml은 각각의 "item" 태그들에 의해 묶여진 각각의 아이템들과 함께 해당 시퀀스를 "sequence" 태그를 통해 시퀀스한다.

[0127] 따라서, 셀렉터 및 시퀀스를 기술하기 위한 간단한 XML 문법이 제공된다. 구성 블록의 각각의 파트가 생성되고, 하나의 오퍼레이션 - 선택 또는 시퀀스를 실행한다. 파트들의 계층구조를 사용하여, 상이한 견고한 선택 및 시퀀스의 집합들이 생성될 수 있다.

[0128] **구성 블록(Composition Block)**

[0129] 패키지의 *구성 블록*은 패키지의 시작 파트로부터 도달할 수 있는 모든 구성 파트들(셀렉터 또는 시퀀스)의 집합을 포함한다. 패키지의 시작 파트가 셀렉터도 시퀀스도 아니면, 구성 블록은 비어 있는 것으로 간주된다. 시작 파트가 구성 파트(composition part)이면, 구성 파트들의 자식 <item>들은 구성 파트들의 직접적이고 비주기적인 그래프(directed acyclic graph)를 생성하도록 재귀적으로 횡단한다(비구성 파트(non-composition part)와 만날 때 횡단를 정지함). 상기 그래프는 구성 블록이다(본 실시예에 따르면, 패키지가 유효해 지기 위해서는 비주기적(acyclic)이어야 함)

[0130] **구성 시만틱(Composition Semantics)**

[0131] 비교적 직접적인(strait-farward) XML 문법을 설정하였다면, 이하의 설명은 콘텐츠 타입에 기초하여 선택이 이루어지도록 정보를 표현하는 방법을 기술한다. 즉, 전술한 XML은 리더들이 구성에 어셈블되는 파트들을 위치시키기에는 충분한 정보를 제공하지만, 리더가 구성의 자연적 특성에 대해 더 파악하도록 도움을 주기에는 충분한 정보를 제공하지는 않는다. 예를 들어, 두 파트들을 구성하는 선택이 주어지면, 선택을 위한 기초사항(예를 들어, 언어, 종이 크기 등)에 대해 리더가 어떻게 알 것인가? 그 답은 이러한 규칙들이 구성 파트의 *콘텐츠 타입(content type)*과 관련된다는 것이다. 따라서, 언어에 기초하여 표현들을 선택하는데 사용되는 셀렉터 파트는 종이 크기들을 근거로 하여 표현들을 선택하는 셀렉터 파트와 상이한 관련 콘텐츠 타입을 갖는다.

[0132] 일반적인 프레임워크는 이러한 콘텐츠 타입에 대한 일반적인 형태를 정의한다:

[0133] Application/XML+Selector_SOMETHING

[0134] Application/XML+Sequence_SOMETHING

[0135] 이러한 콘텐츠 타입에 있어서, SOMETHING은 선택 또는 시퀀스의 자연적 특성, 예를 들어, 종이 크기, 컬러, 언어, 리더 장치에 상주하는 소프트웨어 등을 나타내는 단어로 대체된다. 이러한 프레임워크에서, 모든 종류의 셀렉터들 및 시퀀스들을 상정할 수 있을 것이며, 각각 매우 상이한 의미(semantic)들을 가질 수 있을 것이다.

[0136] 본 프레임워크는 또한 모든 리더들 또는 리딩 장치들이 이해해야만 하는 셀렉터들 및 시퀀스들의 잘 알려진 콘

텐츠 타입들을 정의한다.

[0137]

콘텐츠 타입	규칙
Application/XML+Selector_SupportedContentType	콘텐츠 타입에 기초하여 아이템을 선택한다. 주어진 콘텐츠 타입을 이해하는 소프트웨어가 유용한 제1 아이템을 선택한다.

[0138] 일례로서, 이하를 고려해 본다. 패키지가 페이지를 갖는 다큐먼트를 포함하며, 페이지 중간에 비디오가 나타나는 영역이 있다고 가정한다. 본 일례에서, 페이지의 비디오 파트는 Quicktime 비디오 형태의 비디오를 포함할 수도 있다. 상기 시나리오와 관련된 한가지 문제점은 Quicktime 비디오가 보편적으로 이해되지 않는다는 점이다. 그러나, 본 프레임워크에 따르면, 특히 후술된 리치 패키지 포맷에 따라 보편적으로 이해되는 이미지 포맷(JPEG)이 있다고 가정한다. 전술한 다큐먼트를 포함하는 패키지를 생성할 때, 패키지의 파트로서 비디오를 정의하는 것 외에, 생산자는 페이지에 대한 JPEG 이미지를 정의하고, SupportContentType 셀렉터를 삽입하여, 사용자의 컴퓨터가 Quicktime 비디오를 이해하는 소프트웨어를 가지면, Quicktime 비디오가 선택되고, 그렇지 않으면, JPEG 이미지가 선택되도록 된다.

[0139] 따라서, 상술된 바와 같이, 프레임워크-레벨의 셀렉터 및 시퀀스 컴포넌트들은 본 일례에서는 XML로 정의되는 견고한 계층구조가 구축되도록 한다. 또한, 콘텐츠 타입을 사용해서 셀렉터 및 시퀀스의 동작들을 식별하는 잘 정의된 방법이 있다. 또한, 일 실시예에 따르면, 일반 프레임워크는 소비자(예를 들어, 리더 또는 리딩 장치)가 이해하거나 이해하지 못하는 바에 기초하여 패키지의 사용 및 프로세싱을 허용하며, 미리 정의된 하나의 특정한 콘텐츠 타입을 포함한다.

[0140] 다른 구성 파트 콘텐츠 타입들이 유사한 규칙들을 사용해서 정의될 수 있는데, 그 일례들은 후술된다.

[0141] **기술적 메타데이터(Descriptive Metadata)**

[0142] 일 실시예에 따르면, 기술적 메타데이터 파트들은 패키지 리더들이 값들을 신뢰성 있게 발견할 수 있게 해주는 특성 값들을 저장하는 방법을 라이터(writer) 또는 생산자(producer)에게 제공한다. 이러한 특성들은 통상 저장소 내의 개별 파트들 뿐만 아니라 패키지에 대한 추가 정보를 전체적으로 기록하는데 사용된다. 예를 들어, 패키지의 기술적 메타데이터 파트는 패키지 작성자, 키워드, 요약서 등과 같은 정보를 보유할 수도 있다.

[0143] 본 실시예에서, 기술적 메타데이터는 XML로 표현되며, 잘 알려진 콘텐츠 타입으로 파트들에 저장되며, 잘 알려진 관계 타입(relationship type)들을 사용해서 발견될 수 있다.

[0144] 기술적 메타데이터(descriptive metadata)는 *메타데이터 특성(metadata properties)*들을 보유한다. 메타데이터 특성들은 특성 네임 및 하나의 또는 다수의 특성 값들에 의해 표현된다. 특성 값들은 간단한 데이터 타입들을 가지며, 각각의 데이터 타입은 단일의 XML qname에 의해 기술된다. 기술적 메타데이터 속성들이 간단한 타입들을 갖는다는 사실은 패키지내에서 복잡한 XML 타입들로 데이터를 저장할 수 없음을 의미한다. 이러한 경우에, 전체의(full) XML 파트로서 정보를 저장해야만 한다. 이렇게 하면, 간단한 타입만을 사용하는 것에 대한 모든 제약 사항들이 제거되며, "평이한(flat)" 기술적 메타데이터 속성 모델의 간편함(simplicity)이 손실된다.

[0145] 특성들의 집합을 정의하기 위한 범용 메카니즘에 더하여, 상기 메카니즘을 사용해서 저장된 특정의 잘 정의된 *다큐먼트 코어 특성(document core properties)* 집합이 있다. 이러한 다큐먼트 코어 특성들은 통상 다큐먼트를 기술하는데 사용되며 제목, 키워드, 작성자 등과 같은 속성들을 포함한다.

[0146] 마지막으로, 이러한 다큐먼트 코어 특성들을 보유하는 메타데이터 파트들은 다큐먼트 코어 속성들에 더하여 주문자 정의의 특성(custom-defined properties)들을 보유할 수 있다.

[0147] **메타데이터 포맷(Metadata Format)**

[0148] 일 실시예에 따르면, 기술적 메타데이터 파트들은 콘텐츠 타입을 가지며, 이하의 규칙들에 따른 관계들에 의해 타겟팅 된다:

[0149]

기술적 메타데이터 발견 규칙	주문자-정의의 특성 사용	다큐먼트 코어 특성 사용
기술적 메타데이터 파트의 콘텐츠 타입은 ~이어야 한다:	application/xml-SimpleTypeProperties-PLACEHOLD	

기술적 메타데이터 파트를 타겟으로 하는 관계를 가질 수 있는 소스 파트의 콘텐츠 타입은 ~일 수도 있다:	임의적	임의적
기술적 메타데이터 파트를 타겟으로 하는 관계의 네임은 ~일 수도 있다:	*custom-defined Uri-namespace*	http://mmcf-DocumentCore-PLACEHOLDER
소스 파트에 첨부될 수 있는 기술적 메타데이터 파트들의 수는 ~일 수도 있다:	무제한적(UNBOUNDED)	0 또는 1
동일한 기술적 메타데이터 파트가 첨부될 수 있는 소스 파트들의 수는 ~여야만 한다:	무제한적(UNBOUNDED)	무제한적(UNBOUNDED)

[0150] 일 실시예에 따르면, 이하의 XML 패턴은 기술적 메타데이터를 표현하는데 사용된다. 마크업의 각각의 컴포넌트에 대한 세부 사항은 샘플링 후 표에 제공된다.

```
<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>
```

[0151]

[0152]

마크업 컴포넌트	설명
xmins:mcs="http://mmcf-common-services/PLACEHOLDER"	MMCF 공통 서비스 네임스페이스를 정의한다.
xmins:xsd="http://www.w3.org/2001/XMLSchema"	XML 스키마 네임스페이스를 정의한다. 다수의 커스텀-정의 특성들 및 대부분의 다큐먼트 코어 특성들은 XSD를 사용해서 정의된 빌트인 데이터 타입들을 갖는다. 각각의 특성이 자신의 네임스페이스를 가질 수 있더라도, XSD 네임는 기술적 메타데이터 XML의 루트에 배치된다.
mcs:properties	기술적 메타데이터 XML의 루트 요소
mcs:property	특성 요소. 특성 요소는 특성 qname 및 값을 유지한다. 무한수의 특성 요소들이 있을 수도 있다. 특성 요소들은 루트 요소의 직계의 자식(immediate child)으로 간주된다.
xmins:prns	특성 네임스페이스: 다큐먼트 코어 특성의 경우, http://mmcf-DocumentCore-PLACEHOLDER이다. 주문자-정의의 속성인 경우, 커스텀 네임스페이스이다.
prns:name	특성 네임: 특성 네임을 유지하는 문자열 특성
mcs:type="datatype"	타입은 특성 데이터타입 정의를 유지하는 스트링 속성이다. 예를 들면, xsd:string
mcs:value	본 컴포넌트는 특성의 값을 지정한다. 값 요소들은 속성 요소들의 직계의 자식이다. mcs:multivalued="true"이면, 무제한수의 값 요소(value element)들이 있을 수도 있다.

[0153] **다큐먼트 코어 특성들(Document Core Properties)**

[0154] 이하는 특성 네임, 특성 타입 및 설명을 포함하는 다큐먼트 코어 특성들의 표이다.

[0155]

네임	**타입**	**기술**
Comments	String, optional, single-valued	작성자가 포함하는 다큐먼트 전체에 대한 코멘트. 이는 다큐먼트의 요약서일 수도 있다.
Copyright	String, optional, single-valued	본 다큐먼트에 대한 저작권문자열
EditingTime	Int64, optional, single-valued	다큐먼트를 편집하는데 소비되는 초단위 시간. 어플리케이션 로직에 의해 설정됨. 이 값은 적합한 타입을 가져야만 한다.

IsCurrentVersion	boolean, optional, single-valued	본 인스턴스가 다큐먼트의 현 버전인지 구 버전인지를 나타낸다. 본 필드는 VersionHistory로부터 유도될 수 있으며, 유도 프로세스는 비쌀 수도 있다.
Language	Keyword(= string256), optional, multi-valued	다큐먼트 언어(영어, 불어 등). 본 필드는 어플리케이션 로직에 의해 설정된다.
RevisionNumber	String, optional, single-valued	다큐먼트 개정판.
Subtitle	String, optional, single-valued	다큐먼트의 2차적인 또는 해설적인 타이틀
TextDataProperties	TextDataProperties, optional, single-valued	다큐먼트가 텍스트를 가지면, 본 특성은 단락 카운트, 라인 카운트 등과 같은 다큐먼트의 텍스트 특성들의 집합을 정의한다.
	CharacterCount	Int64
	LineCount	Int64
	PageCount	Int64
	ParagraphCount	Int64
	WordCount	Int64
TimeLastPrinted	datetime, optional, single-valued	본 다큐먼트가 최종 인쇄된 날짜 및 시간
Title	String, optional, single-valued	다큐먼트를 처리하는 어플리케이션에 의해 이해되는 다큐먼트 타이틀. 이는 패키지를 포함하는 파일의 네임과 상이하다.
TitleSortOrder	String, optional, single-valued	타이틀의 소팅 순서(예를 들어, "The Beatles"는 "The"가 없는 SortOrder "Beatles"를 갖는다.
ContentType	Keyword(= string256), optional, multi-valued	어플리케이션 로직에 의해 설정된 다큐먼트 타입. 저장 타입은 인식된 "mime-type"이어야만 한다. 이러한 특성은 특정 타입의 다큐먼트 카테고리화 또는 탐색에 유용할 수도 있다.

[0156] **물리적 모델(Physical Model)**

[0157] 물리적 모델은 패키지가 라이터 및 리더들에 의해 사용되는 다양한 방법들을 정의한다. 상기 모델은 3개의 컴포넌트들: *라이터(writer)*, *리더(reader)*, 및 라이터와 리더 간의 *파이프(pipe)*를 기초로 한다. 도 6은 패키지에 관한 통신을 위해 함께 작업하는 라이터 및 리더의 몇몇 일례들을 도시한다.

[0158] 파이프는 라이터로부터 리더로 데이터를 전달한다. 다수의 시나리오들에서, 파이프는 리더가 로컬 파일 시스템으로부터 패키지를 판독하기 위해 행하는 API 호출(call)들을 단순하게 포함할 수 있다. 이를 *다이렉트 액세스(direct access)*라고 한다.

[0159] 그러나, 때로는 리더 및 라이터는 몇가지 유형의 프로토콜을 통해 서로 통신해야만 한다. 예를 들어, 이러한 통신은 프로세스 경계에 걸쳐 또는 서버와 데스크탑 컴퓨터 사이에서 발생한다. 이를 *네트워크 액세스(network access)*라고 하는데, 파이프의 통신 특성(구체적으로 말하자면, 속도 및 요청 지연시간) 때문에 중요하다.

[0160] 최대 성능을 위해, 물리적 패키지 디자인은 3개의 중요 영역들: *액세스 스타일*, *레이아웃 스타일* 및 *통신 스타일*에 대한 지원을 고려해야만 한다.

[0161] **액세스 스타일(Access Style)**

[0162] 스트리밍 소비(Streaming Consumption)

[0163] 네트워크 액세스를 사용하는 라이터와 리더 간의 통신이 즉시 일어나는(instantaneously) 것이 아니기 때문에, 패키지를 점진적으로 생성 및 소비하도록 하는 것이 중요하다. 특히, 본 실시예에 따르면, 어떠한 물리적 패키지 포맷이라도, 패키지의 모든 비트들이 파이프를 통해 전달되기 전에, 데이터(예를 들어, 파트들)를 수신하는 리더가 데이터를 해석 및 처리하기 시작할 수 있도록 설계될 것이 권고된다. 이러한 기능을 *스트리밍 소비*

*(streaming consumption)*라고 한다.

[0164] 스트리밍 생성(Streaming Creation)

[0165] 라이터가 패키지를 생성하기 시작할 때, 패키지에 무엇이 입력될지를 항상 아는 것은 아니다. 일례로서, 어플리케이션이 인쇄 스풀 파일 패키지를 생성하기 시작할 때, 얼마나 많은 페이지들이 패키지에 입력될 필요가 있는지는 모를 수도 있다. 또 다른 예로서, 리포트를 동적으로 생성하는 서버상의 프로그램은 리포트를 완전히 생성할 때까지 리포트가 얼마나 긴지 또는 얼마나 많은 화상들을 리포트가 포함하는지를 알지 못할 수도 있다. 이와 같은 라이터들을 허용하기 위해, 물리적 패키지들은 다른 파트들이 이미 추가된 후에도 라이터들이 파트들을 동적으로 추가할 수 있게 해야만 한다(예를 들어, 라이터는 기록을 시작할 때 얼마나 많은 파트들이 생성될 것인지를 미리 언급할 것이 요구되어서는 않된다). 또한, 물리적 패키지들은 라이터가 해당 파트의 최종 길이를 알지 못한 채 파트의 콘텐츠를 기록하기 시작할 수 있도록 해야 한다. 이러한 요구 사항들은 *스트리밍 생성(Streaming Creation)*을 가능케 한다.

[0166] 동시적인 생성 및 소비(Simultaneous Creation and Consumption)

[0167] 하이-파이프라인 아키텍처에서, 스트리밍 생성 및 스트리밍 소비는 특정 패키지에 대해 동시에 발생할 수 있다. 물리적 패키지를 설계할 때, 스트리밍 생성 지원 및 스트리밍 소비 지원은 설계를 상반되는 방향으로 몰고 간다. 그러나, 종종 양측을 지원하는 디자인을 찾을 수 있다. 파이프라인 아키텍처의 장점들 때문에, 물리적 패키지들이 *동시적인 생성 및 소비*를 지원할 것이 권고된다.

[0168] **레이아웃 스타일(Layout Styles)**

[0169] 물리적 패키지들은 파트들의 집합을 갖는다. 이러한 파트들은 두가지 스타일들: 간단하게 순서를 매기는 오더링 스타일 및 인터리브된 스타일 중 레이아웃될 수 있다. 간단한 오더링의 경우, 패키지의 파트들은 정의된 오더링으로 레이아웃된다. 패키지의 제1 바이트로부터 시작해서 최종 바이트까지 패키지가 순전히 선형적인 방식으로 전달되는 경우(pure linear fashion), 제1 파트의 모든 바이트들이 먼저 도착하고, 그 후, 제2 파트의 모든 바이트들이 도착하며, 이런 식으로 계속된다.

[0170] 인터리브된 레이아웃에서, 다수의 파트들의 바이트들이 인터리브되어, 특정 시나리오에서 성능이 향상되도록 한다. 인터리빙으로부터 상당한 이익을 얻는 두 시나리오들은 멀티-미디어 재생(예를 들어, 비디오 및 오디오의 동시 전달) 및 인라인 리소스 참조(예를 들어, 마크업 파일의 중간에서 화상을 참조)이다.

[0171] 인터리빙은 인터리브 파트들로 된 콘텐츠를 구성하기 위한 특수한 관례를 통해 처리된다. 파트들을 피스들로 분할하고, 이 피스들을 인터리브 시킴으로써, 보다 큰 원본의 파트를 쉽게 재구성할 수 있는 한편, 희망하는 인터리브 결과를 달성할 수 있다. 인터리빙 작업 방법을 이해하기 위해, 도 7은 두 파트들: content.xml(702) 및 image.jpeg(704)를 포함하는 간단한 일례를 나타낸다. 제1 파트, content.xml은 페이지의 콘텐츠를 기술하고, 해당 페이지의 중간에는 페이지에 나타나야만 하는 이미지(image.jpeg)에 대한 참조가 있다.

[0172] 인터리빙이 왜 중요한지를 이해하기 위해, 도 8에 도시된 바와 같이, 이러한 파트들이 간단한 오더링(ordering)을 사용해서 패키지에서 어떻게 배열되는지를 생각해 본다. 이러한 패키지를 처리하는(또한 바이트들을 순차적으로 수신하는) 리더는 image.jpeg 파트 뿐만 아니라 content.xml 파트를 모두 수신할 때까지 그림을 디스플레이할 수 없게 될 것이다. 어떠한 경우에는(예를 들어, 소형 또는 간단한 패키지들, 또는 고속 통신 링크)에서, 이는 문제가 되지 않을 수도 있다. 다른 경우에서는(예를 들어, content.xml이 매우 크거나 또는 통신 링크가 매우 느린 경우), 화상에 도달하기 위해 모든 content.xml 파트를 판독할 필요성은, 바람직하지 않은 성능을 야기하거나 리더 시스템에 대해 부당한 메모리 수요를 야기하게 된다.

[0173] 이상적인 성능에 보다 가깝게 달성하기 위해, 그림이 참조되는 바로 직후, 중간에 content.xml 파트를 분할하여 image.jpeg 파트를 삽입할 수 있는 것이 좋다. 이는 리더가 보다 일찍 화상을 처리하기 시작할 수 있게 해준다: 참조자를 만나자마자, 이미지 데이터가 따라온다. 예를 들어, 이는 도 9에 도시된 패키지 레이아웃을 야기한다. 성능상의 장점들 때문에, 종종 물리적 패키지가 인터리빙을 지원하는 것이 바람직하다. 사용되는 물리적 패키지의 종류에 따라, 인터리빙은 지원될 수도 지원되지 않을 수도 있다. 상이한 물리적 패키지들이 인터리빙의 내부 표현을 상이하게 처리할 수도 있다. 물리적 패키지가 인터리빙을 처리하는 방법과 무관하게, 인터리빙은 물리 레벨에서 발생하는 최적화이며, 물리 파일내에서의 다수의 피스들로 분할되는 파트는 하나의 논리적 파트이며; 피스들 자체는 파트가 아님을 기억하는 것이 중요하다.

[0174] **통신 스타일(Communication Styles)**

[0175] 라이터와 리더 간의 통신은 파트들의 *순차적 전달(sequential delivery)*에 기초하거나, 또는 순서 없이 액세스될 수 있는 파트들에 대한 *랜덤-액세스(random access)*에 기초할 수 있다. 어떤 통신 스타일이 사용될 지는 파이프 패키지 포맷 및 물리적 패키지 포맷 모두의 능력에 좌우된다. 일반적으로, 모든 파이프들은 순차적 전달을 지원한다. 물리적 패키지들은 순차적 전달을 지원해야만 한다. 랜덤-액세스 시나리오를 지원하기 위해, 사용중인 파이프 및 물리적 패키지는 모두 랜덤-액세스를 지원해야만 한다. 몇몇 파이프들은 랜덤-액세스를 가능케 할 수 있는 프로토콜들(예를 들어, 바이트 범위의 지원성능을 갖는 HTTP 1.1)에 기초한다. 이러한 파이프들이 사용될 때 최대 성능을 위해, 물리적 패키지가 랜덤-액세스를 지원할 것이 권고된다. 이러한 지원의 부재시, 리더들은 필요한 파트들이 순차적으로 전달될 때까지 단순히 대기한다.

[0176] **물리적 매핑(Physical Mappings)**

[0177] 논리적(logical) 패키지 모델은 패키지 추상화(package abstraction)를 정의한다: 패키지의 실제 인스턴스는 패키지의 몇몇 특정한 물리적 표현에 기초한다. 패키지 모델은 다양한 트랜스포트들(예를 들어, 네트워크 기반의 프로토콜) 뿐만 아니라 물리적인 영구적 포맷들에 매핑될 수도 있다. 물리적 패키지 포맷은 추상적 패키지 모델로부터 특정한 물리적 포맷의 특징들로의 매핑으로서 기술될 수 있다. 패키지 모델은 어떤 물리적 패키지 포맷들이 패키지의 보관, 분배, 또는 스풀링을 위해 사용되어야 하는지를 지정하지 않는다. 일 실시예에서, 논리적 구조만이 지정된다. 패키지는 루스 파일들의 집합, .ZIP 파일 아카이브(archive), 복합 파일, 또는 몇몇 다른 포맷에 의해 "물리적으로" 구현될 수도 있다. 선택된 포맷은 타겟팅 된 소비 장치 또는 디바이스용 드라이버에 의해 지원된다.

[0178] 매핑되는 컴포넌트(Components Being Mapped)

[0179] 각각의 물리적 패키지 포맷은 이하의 컴포넌트들에 대한 매핑을 정의한다. 몇몇 컴포넌트들은 임의선택적(optional)이며, 특정한 물리적 패키지 포맷은 상기 선택적 컴포넌트들을 지원하지 않을 수도 있다.

[0180]

	컴포넌트	**설명**	**요구됨 또는 선택적**
파트	네임	파트 네이밍	요구됨
	콘텐츠 타입	파트에 저장된 식별된 콘텐츠의 종류	요구됨
	파트 콘텐츠	파트의 실제 콘텐츠를 저장한다	요구됨

[0181] 공통 매핑 패턴들(Common Mapping Patterns)

[0182]

	컴포넌트	**설명**	**요구됨 또는 선택적**
액세스 스타일	스트리밍 소비	전체 패키지가 도달하기 전에 리더들이 파트들의 처리를 시작할 수 있게 해준다.	선택적
	스트리밍 생성	기록될 모든 파트들을 미리 알지 않은채 패키지로의 파트 기록을 라이터들이 시작할 수 있게 해준다.	선택적
	동시적인 생성 및 소비	스트리밍 생성 및 스트리밍 소비가 동일한 패키지에서 동시에 발생할 수 있게 해준다.	선택적
레이아웃 스타일	간단한 오더링	파트 N+1에 대한 바이트들 전에 파트 N에 대한 모든 바이트들이 패키지에 나타난다.	선택적
	인터리브	다수의 파트들에 대한 바이트들이 인터리브된다.	선택적
통신 스타일	순차적 전달	파트 N 전부가 파트 N+1 이전에 리더에게 전달된다.	선택적
	랜덤-액세스	순차적 순서 없이 리더가 파트의 전달을 요청할 수 있다.	선택적

[0183] 기능들이 패키지-모델 컴포넌트들과 부분적으로 일치하는 많은 물리적 스토리지 포맷들이 존재한다. 패키지 모델로부터 그러한 스토리지 포맷으로의 매핑을 정의함에 있어서, 물리 스토리지 매체에 본래 존재하지 않는 추가 능력들을 제공하기 위해 매핑 층들을 사용하는 반면, 패키지 모델과 물리 스토리지 매체 간의 능력들의 임의의 유사성들을 이용하는 것이 요구될 수도 있다. 예를 들어, 몇몇 물리적 패키지 포맷들은 파일 시스템에 개별 파일들로서 개별 파트들을 저장할 수도 있다. 이러한 물리 포맷에서는, 다수의 파트 네임들을 동일한 물리적 파일 네임들에 직접적으로 매핑하는 것이 자연스러울 것이다. 유효하지 않은 파일 시스템 파일 네임이 아닌 문자들을 사용하는 파트 네임들은 몇몇 종류의 이스케이프(escape) 메카니즘을 필요로 할 수도 있다.

[0184] 다수의 경우에 있어서, 상이한 물리적 패키지 포맷들의 설계자들은 단일 공통 매핑(single common mapping)의 문제점을 겪을 수도 있다. 공통 매핑의 문제점의 두 개의 예로서, 임의의 콘텐츠 타입(content type)들을 파트들과 관련시킬 때, 및 인터리브된 레이아웃 스타일을 지원할 때 발생한다. 본 명세는 그러한 공통 매핑의 문제점들에 대하여 공통 해결책을 제안한다. 특정의 물리적 패키지 포맷들의 설계자들은, 본 명세서에 정의된 공통 매핑 해결책들을 사용하도록 장려될 수 있으나, 요구되는 것은 아니다.

[0185] 파트의 콘텐츠 타입 식별(Identifying Content Types of Parts)

[0186] 물리적 패키지 포맷의 매핑은 각각의 파트에 대한 콘텐츠 타입을 저장하기 위한 메카니즘을 정의한다. 몇몇 물리적 패키지 포맷들은 콘텐츠 타입들을 나타내기 위한 본래의 메카니즘(예를 들어, MIME의 "콘텐츠 타입" 헤더)을 갖는다. 이러한 물리적 패키지들의 경우, 파트들에 대한 콘텐츠 타입들을 표현하는 원본의 메카니즘을 사용할 것이 권고된다. 다른 물리적 패키지 포맷들의 경우, 맵핑시 몇몇 다른 메카니즘이 콘텐츠 타입들을 표현하는데 사용된다. 이러한 패키지 내의 콘텐츠 타입들을 표현하는데 권고되는 메카니즘은, 타입 스트림(type streams)으로 알려진 특별하게 명명된 XML 스트림을 패키지에 포함하는 것이다. 이러한 스트림은 파트가 아니며, 따라서 자체적으로 URI-어드레싱이 가능하지는 않다. 그러나, 인터리빙 파트들을 위해 사용되는 동일한 메카니즘들을 사용하여 물리적 패키지에서 인터리브될 수 있다.

[0187] 타입(types) 스트림은 최상위 레벨의 "Types" 요소, 및 하나 이상의 "Default" 및 "Override" 부요소들을 갖는 XML을 포함한다. "Default" 요소들은 파트 네임 확장자로부터 콘텐츠 타입으로의 디폴트 매핑을 정의한다. 이는 파일 확장자가 종종 콘텐츠 타입에 대응한다는 사실을 이용한 것이다. "Override" 요소들은 디폴트 매핑으로 커버되지 않거나 디폴트 매핑과 일치되지 않은 파트들에 대한 콘텐츠 타입들을 지정하는데 사용된다. 패키지 라이터들은 "Default" 요소들을 사용해서 파트당 "Override" 요소들의 수를 감소시킬 수도 있는데, 반드시 그럴 필요는 없다.

[0188] "Default" 요소는 이하의 속성들을 갖는다:

[0189]

네임	설명	필요성
확장자	파트 네임 확장자. "Default" 요소는 이러한 속성값에 이어지는 마침표로 그 명칭이 끝을 맺는 임의의 파트와 일치한다.	예
콘텐츠 타입	RFC2045에 정의된 콘텐츠 타입. 임의의 매칭 파트들의 콘텐츠 타입을 나타낸다("Override" 요소에 의해 오버라이드되지 않는 한; 이하를 참조하라).	예

[0190] "Override" 요소는 이하의 속성들을 갖는다:

[0191]

네임	설명	필요성
파트 네임	파트 네임 URI. "Override" 요소는 이러한 속성 값과 동일한 네임을 갖는 파트와 일치한다.	예
콘텐츠 타입	RFC2045에 정의된 콘텐츠 타입. 일치하는 파트의 콘텐츠 타입을 나타낸다.	예

[0192] 다음은 type 스트림에 포함된 XML의 일례이다:

```
<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>
```

[0193]

[0194] 이하의 표는 파트들, 및 상기 type 스트림에 의해 정의되는 대응 콘텐츠 타입들의 샘플 리스트를 나타낸다:

[0195]

파트 네임	콘텐츠 타입
/a/b/sample1.text	plain/text
/a/b/sample2.jpeg	image/jpeg
/a/b/sample3.picture	image/gif
/a/b/sample4.picture	image/jpeg

[0196] 패키지내의 모든 파트에 있어서, types 스트림은 (a) 하나의 매칭하는 "Default" 요소, (b) 하나의 매칭하는 "Override" 요소, 또는 (c) 매칭하는 "Default" 요소 및 매칭하는 "Override" 요소 모두(이러한 경우, "Override" 요소가 우선 순위를 가짐)를 포함한다. 일반적으로, 임의의 주어진 확장자에 있어서, 많아야 하나의 "Default" 요소가 있으며, 임의의 주어진 파트 네임에 있어서 많아야 하나의 "Override" 요소가 있다.

[0197] types 스트림의 "Default" 및 "Override" 요소들의 순서는 중요하지 않다. 그러나, 인터리브된 패키지에서는, "Default" 및 "Override" 요소들은 물리적 패키지에서 대응하는 파트(들) 이전에 나타난다.

[0198] 인터리빙(Interleaving)

[0199] 모든 물리적 패키지들이 파트들의 데이터 스트림들의 인터리빙을 근본적으로 지원하는 것은 아니다. 일 실시예에서, 그러한 임의의 물리적 패키지로의 매핑은 파트들의 인터리빙이 되게 하기 위해 본 장에 기술된 일반적인 메카니즘을 사용한다. 일반적인 메카니즘은 파트의 데이터 스트림을 다른 파트들의 피스들 또는 전체 파트들과 인터리브될 수 있는 다수의 피스들로 분할하도록 작용한다. 파트의 개별 피스들은 물리적 매핑에 존재하며, 논리적 패키지 모델에서 어드레스 불가능하다. 피스들은 제로 사이즈를 가질 수도 있다.

[0200] 이하, 파트 네임에서 파트의 개별 피스들에 대한 네임으로의 맵핑은, 리더가 파트의 데이터 스트림을 형성하기 위해 원래의 순서로 피스들을 연결시키도록(sticth)정의된다.

[0201] 주어진 파트 네임에 대하여 피스 네임들을 유도하기 위한 문법은 다음과 같다:

[0202]
```
piece_name = part_name "/" "[" 1*digit "]" [ ".last" ] ".piece"
```

[0203] 문법에 의해 생성되는 피스 네임(piece_names)에 대한 이하의 유효성 제약 사항(validity constraints)들이 존재한다:

[0204] • 피스 번호들은 0으로 시작하며, 연속적인 양의 정수이다. 피스 번호들은 좌측 여백을 0으로 채울 수 있다.

[0205] • 파트의 피스 집합의 최종 피스는 피스 네임에서 ".piece" 전에 ".last"를 포함한다.

[0206] • 피스 네임은 물리적 패키지에서 네임들에 매핑되기 전에 논리적 파트의 네임으로부터 생성된다.

[0207] 자신의 자연적인 순서로 피스들을 저장할 필요는 없지만, 그렇게 기억하면 최적 효율성을 제공할 수도 있다. 인터리브된 (피스) 파트들을 포함하는 물리적 패키지는 인터리브되지 않은 (한 피스의) 파트들을 또한 포함할 수 있다. 따라서 이하의 일례는 타당하다:

[0208]
```
spine.xaml/[0].piece
pages/page0.xaml
spine.xaml/[1].piece
pages/page1.xaml
spine.xaml/[2].last.piece
pages/page2.xaml
```

[0209] 특정 매핑(Specific Mappings)

[0210] 이하는 이하의 물리 포맷들: 윈도우즈 파일 시스템의 루스 파일들에 대한 특정의 매핑을 정의한다.

[0211] 윈도우즈 파일 시스템의 루스 파일들에 대한 매핑(Mapping to Loose Files in a Windows file system)

[0212] 논리적 모델의 요소들을 물리적 포맷으로 매핑하는 방법을 더 잘 이해하기 위해, 메트로(Metro) 패키지를 윈도우즈 파일 시스템의 루스 파일들의 집합으로서 표현하는 기본적인 경우를 고려해 본다. 논리적 패키지의 각각의 파트는 별도의 파일(stream)에 포함되게 된다. 논리적 모델의 각각의 파트 네임은 파일의 네임에 대응한다.

[0213]

논리적 컴포넌트	물리적 표현
파트	파일(들)
파트 네임	경로를 갖는 파일 네임(URI 처럼 보임, 슬래시를 백슬래시로 변경 등)
파트 콘텐츠 타입	파일 네임 및 그 관련 타입의 간단한 리스트를 표현하는 XML을 포함하는 파일

[0214] 파트 네임은 이하의 표에 도시된 바와 같이 타당한 윈도우즈 파일 네임으로 번역된다.

[0215] 논리적 파트 네임 세그먼트(URI 세그먼트) 및 윈도우즈 파일네임에 타당한 두 개의 문자 집합들이 이하에 제공된다. 본 표는 두가지 중요한 사실을 나타낸다:

[0216] URI를 파일 네임으로 변환할 때 이스케이프할 필요가 있는 두 개의 타당한 URI 심볼들, ":" 및 "*" 가 있다.

[0217] URI에서는 존재할 수 없는 타당한 파일 네임 심볼들 ^ {} [] #이 있다(인터리빙 같은 특별 매핑을 위해 사용될 수 있다).

[0218] "이스케이프"는 파트 네임이 파일 네임에서 사용될 수 없는 문자를 포함하는 경우, 타당한 파일 네임 문자를 생성하는 기술로서 사용된다. 문자를 이스케이프하기 위해, 탈자 부호(^)가 사용되며, 뒤에 문자의 16진수가 따라온다.

[0219] abs_path(파트 네임)로부터 파일 네임으로 매핑하기 위해:

[0220]
```
remove first /
convert all  / to \
escape colon and asterisk characters
```

[0221] 예를 들어, 파트 네임 /a:b/c/d*.xaml은 이하의 파일 네임 a^25b\c\d^2a.xaml 이 된다.

[0222] 역 매핑을 수행하기 위해:

[0223] convert all \ to /

[0224] add / to the beginning of the string

[0225] unescape characters by replacing ^[hexCode] with the corresponding characters

From URI grammar rules (RFC2396)	Characters that are valid for naming files, folders, or shortcuts
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"	Alphanum | ^ Accent circumflex (caret) & Ampersand ' Apostrophe (single quotation mark) @ At sign { Brace left } Brace right [Bracket opening] Bracket closing , Comma $ Dollar sign = Equal sign ! Exclamation point - Hyphen # Number sign (Parenthesis opening) Parenthesis closing % Percent . Period + Plus ~ Tilde _ Underscore

[0226]

[0227] 버전화 및 확장성(Versioning and Extensibility)

[0228] 다른 기술 명세(specification)들처럼, 본 명세서에 기술된 명세는 차후의 보강을 위해서 전개될 수도 있다. 본 명세의 제1 판의 디자인은 제1 판에 기초하여 기록된 소프트웨어 시스템과 차후의 개정판들을 위해 기록된 소프트웨어 시스템 간의 다큐먼트의 차후 상호 교환을 위한 계획들을 포함한다. 마찬가지로, 본 명세는 제삼자들이 명세에 대한 확장판을 생성할 수 있도록 한다. 그러한 확장판은, 예를 들어, 프린터의 존재를 알지 못하는 다른 리더들과의 호환성을 여전히 보유하는 반면, 확장판은 몇몇 특정 프린터의 기능을 이용하는 다큐먼트가 구축되도록 할 수도 있다.

[0229] 고정된 페이로드 마크업의 새로운 버전을 사용하는 다큐먼트들, 또는 마크업에 대한 제삼자 확장판(third-party extensions)들은 행동(예를 들어, 가시적으로 렌더링 하는 방법)에 대하여 적합한 결정을 할 것을 리더들에게 요구한다. 리더들을 안내하기 위해, 다큐먼트 작성 툴(또는 다큐먼트를 생성한 툴)은, 그렇지 않으면 인식되지 않는, 요소들 또는 속성들을 대하게 되는 리더들에 대한 적합한 동작을 지정해야만 한다.

[0230] 새로운 프린터, 브라우저, 및 다른 클라이언트는 차후 기능들을 위한 다양한 지원을 구현할 수도 있다. 새로운 확장판의 버전을 이용하는 다큐먼트 작성 툴들은 확장판의 버전을 알지 못하는 리더들의 해동을 주의 깊게 고려해야만 한다.

[0231] 버전화 네임스페이스(Versioning Namespace)

[0232] XML 마크업 인식은 네임스페이스 URI들에 기초한다. 임의의 XML-네임스페이스의 경우, 리더는 해당 네임스페이스에 정의된 XML-요소들 및 XML-속성들 중 전부를 인식하거나 전혀 인식하지 않는 것으로 예상된다. 리더가 새로운 네임스페이스를 인식하지 않으면, 리더는 다큐먼트 내에 지정되어 있는 바와 같이 폴백 렌더링 오퍼레이션(fallback rendering operation)을 실행할 필요가 있다.

[0233] XML 네임스페이스 URI 'http://PLACEHOLDER/version-control'은 버전에 적응적이고, 확장판에 적응적인 고정된 페이로드 마크업을 구성하는데 사용되는 XML 요소들 및 속성들을 포함한다. 고정된 페이로드가 그 내부에 반드시 버전화 요소들을 가질 필요는 없다. 그러나, 적응적인 콘텐츠를 구축하기 위해, <ver:Compatibility.Rules>

및 <ver:AlternativeContent> XML-요소들 중 적어도 하나를 사용해야만 한다.

[0234] 고정된 페이로드 마크업의 명세는 그와 관련된 xmlns URI: 'http://PLACEHOLDER/pdl'을 갖는다. 고정된 페이로드 내의 이러한 네임스페이스를 사용하는 것은 본 명세에 정의된 요소들만이 사용됨을 리더 어플리케이션에 알려주게 된다. 본 명세의 차후 버전들은 자신의 네임스페이스들을 가지게 될 것이다. 새로운 네임스페이스와 친숙한 리더 어플리케이션들은 이전 버전에 정의된 속성의 요소들의 대집합을 지원하는 방법을 파악할 것이다. 새로운 버전에 익숙하지 않은 리더 어플리케이션들은 PDL에 대하여 몇몇 미지의 확장판의 URI인 것처럼 새로운 버전의 URI를 생각한다. 이러한 어플리케이션들은 한 네임스페이스가 다른 네임스페이스의 대집합인 네임스페이스들 간의 관계가 존재함을 모를 수도 있다.

[0235] 백워드 및 "포워드" 호환성(Backward and "Forward" Compatibility)

[0236] 본 명세서에 기술된 시스템 및 방법을 지원하는 어플리케이션 또는 디바이스의 문맥(context)에서, 호환성(compatibility)은 명세의 이전 버전, 또는 명세의 미지의 확장판 또는 버전을 사용해서 작성된 다큐먼트들을 파싱 및 표시하는 클라이언트의 기능을 말한다. 다양한 버전화 메카니즘들은 후술된 바와 같이 클라이언트가 명세의 하위 레벨의 버전들에 기초하여 다큐먼트들을 지원할 수 있도록 차후 구현이 이루어지도록 하는 "백워드 호환성"을 나타낸다.

[0237] 프린터와 같은 구현된 클라이언트가 마크업 언어의 차후 버전을 사용한 다큐먼트를 수신하는 경우, 클라이언트는 활용가능한 렌더링 옵션들을 파싱하여 이해하게 될 것이다. 구 버전의 명세에 따라 작성된 클라이언트 소프트웨어가 신 버전의 특징을 사용한 다큐먼트를 처리하는 기능은, "포워드 호환성(forward compatibility)"라고 한다. 포워드 호환성이 가능하도록 작성된 다큐먼트를 "버전-적응적(version-adaptive)"이라고 한다.

[0238] 또한, 구현된 클라이언트들은 새로운 요소들 또는 속성들을 나타내는 미지의 확장판을 갖는 다큐먼트들을 지원가능할 필요가 있을 것이므로, 다양한 의미론(semantic)이 "확장판-적응적(extension-adaptable)"인 다큐먼트들의 보다 일반적인 경우를 지원한다.

[0239] 프린터 또는 뷰어가 미지의 확장판을 대하게 되면, 주변 콘텐츠를 적응적으로 렌더링하는 것에 관한 안내를 위해 확장판의 사용과 함께 임베드된 정보를 찾는다. 이는 미지의 요소들 또는 속성들을 알려진 콘텐츠로 대체하는 것을 포함한다. 그러나, 이는 미지의 콘텐츠를 아주 무시하는 것을 포함해서 다른 형태들을 취할 수 있다. 명시적인 안내가 부재한 경우, 리더는 마크업의 미지의 확장판의 존재를 에러-상태로 처리해야만 한다. 안내가 제공되지 않으면, 확장자는 콘텐츠를 이해하는데 기본적인 것으로 여겨진다. 렌더링 실패는 포착되어 사용자에게 보고된다.

[0240] 이러한 모델을 지원하기 위해, 마크업 언어의 새롭고 확장된 버전들이 네임스페이스에서 관련 확장판들을 논리적으로 그룹화하여야 한다. 이러한 방법으로, 다큐먼트 작성자들은 최소의 수의 네임스페이스들을 사용해서 확장된 기능들을 이용할 수 있다.

[0241] 버전화 마크업(Versioning Markup)

[0242] 확장판-적응적인 동작을 지원하기 위한 XML 어휘에는 이하의 요소를 포함한다.

[0243]

버전화 요소 및 계층	설명
<Compatibility.Rules>	파서가 미지의 요소 또는 속성에 반응하는 방법을 제어한다.
<Ignorable>	관련 네임스페이스 URI가 무시될 수 있다고 선언한다.
<ProcessContent>	요소가 무시되면, 요소의 콘텐츠가 무시된 요소의 저장소에 의해 포함된 것처럼 처리된다고 선언한다.
<CarryAlong>	다큐먼트가 변경될 때 무시될 수 있는 콘텐츠가 보존돼야만 하는지를 다큐먼트 구성 툴에 지시한다.
<MustUnderstand>	무시될 수 있다고 선언된 요소의 영향을 역으로 한다.
<AlternateContent>	버전화/확장판 기능을 사용하는 마크업에서, <AlternateContent> 요소는 바람직한 것으로 지정된 마크업을 처리할 수 없는 리더 어플리케이션들에 의해 사용되는 대체의 "폴백" 마크업과 연관시킨다.
<Prefer>	양호한 콘텐츠를 지정한다. 상기 콘텐츠는 버전/익스텐션 기능들을 클라이언트가 안다는 내용이다.
<Fallback>	다운-레벨 클라이언트의 경우, 양호한 콘텐츠를 위해 대체될 '다운-레벨' 콘텐츠를 지정한다.

[0244] <Compatibility.Rules> 요소

[0245] Compatibility.Rules는 Xaml 루트 요소 뿐만 아니라 첨부된 속성을 보유할 수 있는 임의의 요소에 첨부될 수 있다. <Compatibility.Rules> 요소는 파서가 미지의 요소들 또는 속성들에 대응하는 방법을 제어한다. 통상 이러한 아이템들은 에러로 보고된다. Ignorable 요소를 Compatibility.Rules 속성에 추가하는 것은 특정 네임스페이스로부터의 아이템들이 무시될 수 있음을 컴파일러에게 알려 준다.

[0246] Compatibility.Rules는 요소들 Ignorable 및 MustUnderstand를 포함할 수 있다. 디폴트에 의해, 모든 요소들 및 속성들은 MustUnderstand로 추정된다. 요소들 및 속성들은 Ignorable 요소를 저장소의 Compatibility.Rules 속성에 추가함으로써 Ignorable이 될 수 있다. 요소 또는 속성은 MustUnderstand 요소를 내포된 저장소들 중 하나에 추가함으로써 MustUnderstand가 될 수 있다. Ignorable 또는 MustUnderstand는 동일한 Compatibility.Rules 요소 내의 특정 네임스페이스 URI를 일컫는다.

[0247] <Compatibility.Rules> 요소는 저장소 자신의 태그 또는 속성들이 아닌 저장소의 콘텐츠에 영향을 준다. 저장소의 태그 또는 속성들에 영향을 주기 위해, 그 저장소는 호환성 규칙(compatibility rule)들을 포함해야만 한다. Xaml 루트 요소는 Canvas와 같이 다른 루트 요소들이 될 수도 있는 요소들에 대한 호환성 규칙들을 지정하는데 사용될 수 있다. Compatibility.Rules 복합 속성은 저장소의 제1 요소이다.

[0248] <Ignorable> 요소

[0249] <Ignorable> 요소는 포함된 네임스페이스 URI가 무시될 수 있음을 선언한다. <Ignorable> 태그가 현 블록 또는 저장소 블록내의 항목 앞에 선언되고, 네임스페이스 URI가 파서에게 인식되지 않으면 항목은 무시될 수 있다고 간주될 수 있다. URI가 인식되면, Ignorable 태그는 무시되고 모든 항목들은 인식된다. 일 실시예에서, Ignorable로 명백하게 선언되지 않은 모든 아이템들은 인식돼야만 한다. Ignorable 요소는 <ProcessContent> 및 <CarryAlong> 요소들을 포함할 수 있는데, 상기 요소들은 구성된 다큐먼트들 내에 콘텐츠가 어떻게 보존돼야만 하는지에 대한 안내를 다큐먼트 구성 툴에 제공할 뿐만 아니라 요소가 무시되는 방법을 변경하는데 사용된다.

[0250] <ProcessContent> 요소

[0251] <ProcessContent> 요소는 요소가 무시되는 경우, 요소의 콘텐츠가 무시된 요소의 저장소에 포함되는 것처럼 처리됨을 선언한다.

[0252] <ProcessContent> 속성들

[0253]

속성	내용
Elements	콘텐츠를 처리하는 요소 네임들의 스페이스 범위가 정해진 리스트, 또는 모든 요소들의 콘텐츠가 처리돼야만 함을 나타내는 "*". Elements 속성은 지정되지 않으면 "*"로 디폴트된다.

[0254] <CarryAlong> 요소

[0255] 선택사항인 <CarryAlong> 요소는 무시 가능 콘텐츠가 다큐먼트가 변경될 때 보존돼야만 하는지를 다큐먼트 구성 툴에 나타낸다. 구성 툴이 무시 가능한 콘텐츠를 보존하거나 폐기하는 방법은 구성 툴의 영역 내이다. 다중의 <CarryAlong> 요소들이 네임스페이스에서 동일한 요소 또는 속성을 지칭한다면, 지정된 최종 <CarryAlong>가 우선 순위를 갖는다.

[0256] <CarryAlong> 속성들

[0257]

속성	기술
Elements	다큐먼트가 구성될 때 함께 전달되도록 요청된 요소 네임들의 스페이스 범위가 정해진 리스트, 또는 네임스페이스의 모든 요소들의 콘텐츠가 함께 전달돼야만 함을 나타내는 "*". Elements 속성은 지정되지 않으면 "*"로 디폴트된다.
Attributes	함께 전달된 요소들 내의 속성 네임들의 스페이스 범위가 정해진 리스트, 또는 요소들의 모든 속성들이 함께 전달돼야만 함을 나타내는 "*". 요소가 무시되고 함께 전달될 때, 모든 속성들은 상기 속성의 콘텐츠와 무관하게 함께 전달된다. 이하의 일례에서와 같이, 이러한 속성은 지정된 속성이 무시되지 않는 요소에서 사용되는 경우에만 효과가 있다. 디폴트에 의해, 속성은 "*"이다.

[0258] <MustUnderstand> 요소

[0259] <MustUnderstand>는 Ignorable 요소의 효과를 역으로 바꾸는 요소이다. 예를 들어, 다른 콘텐츠와 결합될 때, 이러한 기술이 유용하다. <MustUnderstand> 요소에 의해 정의된 범위 밖에서, 요소는 Ignorable로 유지된다.

[0260] <MustUnderstand> 속성들

[0261]

속성	기술
NamespaceUri	아이템이 이해돼야만 하는 네임스페이스의 URI.

[0262] <AlternateContent> 요소

[0263] <AlternateContent> 요소는 지정된 콘텐츠의 임의의 부분이 인식되지 않은 경우 다른 콘텐츠가 제공되도록 한다. AlternateContent 블록은 <Prefer> 및 <Fallback> 블록 모두를 사용한다. <Prefer> 블록의 어느 것도 인식되지 않으면, <Fallback> 블록의 콘텐츠가 사용된다. 네임스페이스는 폴백이 사용됨을 나타내기 위해 <MustUnderstand>로 선언된다. 네임스페이스가 무시 가능하다고 선언되고 네임스페이스가 <Prefer> 블록 내에서 사용되면, <Fallback> 블록의 콘텐츠는 사용되지 않는다.

[0264] 버전화 마크업(Versioning Markup)의 예들

[0265] <Ignorable>의 사용

[0266] 본 예는, 의제(fictitous) 마크업 네임스페이스, http://PLACEHOLDER/Circle을 사용하는 것으로, 초기 버전으로 요소 Circle을 정의하고, 마크업의 차후 버전(버전 2)에 소개된 Circle의 Opacity 속성 및 마크업의 또 다른 버전(버전 3)에 소개된 Luminance 속성을 사용한다. 이러한 마크업은 버전 1, 2, 3 등에서 여전히 로딩가능하다. 또한, <CarryAlong> 요소는 편집기가 v3:Luminance를 인식하지 못하더라도, 편집시 v3:Luminance MUST가 보존되는 것으로 지정한다.

[0267] 버전 1 리더에 대하여, Opacity 및 Luminance 가 무시된다.

[0268] 버전 2 리더에 대하여, Luminance 만이 무시된다.

[0269] 버전 3 리더 및 그 이상에 대하여, 모든 속성들이 사용된다.

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

[0270]

[0271] <MustUnderstand>의 사용

[0272] 이하의 일례는 <MustUnderstand> 요소의 사용을 설명한다.

```
<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>
```

[0273]

[0274] 루트 요소에서 Ignorable로 선언되었더라도, <MustUnderstand> 요소의 사용은 v3:Luminance에 대한 참조들이 에러가 되게 한다. 예를 들어, 대신 버전 2에 추가된 Canvas의 Luminance 속성을 사용하는 대체예의 콘텐츠와 결합되는 경우 본 기술은 유용하다(이하 참조). Canvas 요소의 범위 밖에서, Circle의 Luminance 속성이 다시 무시될 수 있다.

```
<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>
```

[0275]

[0276] <AlternateContent>의 사용

[0277] 임의의 요소 또는 속성이 <MustUnderstand>로 선언되지만 <AlternateContent> 블록의 <Prefer> 블록에서 인식되지 않으면, <Prefer> 블록은 전체적으로 스킵되고, <Fallback> 블록이 통상적으로 처리된다(즉, 대하게 되는 어떠한 MustUnderstand 항목이라도 에러로 보고된다).

```
<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>
```

[0278]

[0279] **리치 패키지 포맷(THE REACH PACKAGE FORMAT)**

[0280] 이하의 설명에서, 특정 파일 포맷에 대한 설명이 제공된다. 본 장의 별도의 주요 부제들은 "리치 패키지 포맷 서론", "리치 패키지 구조", "고정된 페이로드 파트", "FixedPage 마크업 기본 사항", "고정된 페이로드 요소들 및 속성들"과 "FixedPage 마크업"을 포함한다. 각각의 주요 부제는 하나 이상의 관련 소제목들을 갖는다.

[0281] **리치 패키지 포맷 서론(Introduction to the Reach Package Format)**

[0282] 일례의 프레임워크가 상술되었는데, 이하의 설명은 전술한 툴들을 활용하여 제공되는 지정된 포맷 중 하나이다. 이하의 설명은 단지 하나의 일례의 포맷을 구성하는 것으로 청구된 주제의 어플리케이션을 제한하려는 의도가 아님을 알 것이다.

[0283] 본 실시예에 따르면, 단일 패키지는 다중의의 페이로드들을 포함할 수도 있는데, 각각은 다큐먼트의 상이한 표현으로서 작용한다. *페이로드(payload)*는 식별 가능한 "루트" 파트 및 루트 파트의 타당한 처리를 위해 요구되는 모든 파트들을 포함하는 파트들의 집합이다. 예를 들어, 페이로드는 다큐먼트의 고정 표현, 리플로 가능한 표현, 또는 임의의 표현일 수 있다.

[0284] 이하의 설명은 *고정된 페이로드(fixed payload)*라고 하는 특정 표현을 정의한다. 고정된 페이로드는 순서대로 FixedPage 파트들을 참조하는 FixedPanel 마크업을 포함하는 루트 파트를 갖는다. 이는 함께 멀티-페이지 다큐먼트의 정확한 렌더링을 기술한다.

[0285] 적어도 하나의 고정된 페이로드를 유지하고, 후술된 다른 규칙들을 따르는 패키지를 *리치 패키지(reach package)*라고 하는 것으로 알려져 있다. 리치 패키지의 리더 및 라이터는 리치 패키지 포맷의 명세에 기초하여 자신의 파서 및 렌더링 엔진을 구현할 수 있다.

[0286] **리치 패키지 특징(Features of Reach Packages)**

[0287] 본 실시예에 따르면, 리치 패키지는 정보 작업자들이 다큐먼트들을 배포, 보관(archiving) 및 렌더링하기 위해 가지는 요구 사항들을 나타낸다. 알려진 렌더링 규칙을 사용해서, 리치 패키지들은 클라이언트 디바이스 또는 어플리케이션을 특정 운영 체제 또는 서비스 라이브러리에 결합시키려 하지 않고, 저장된 포맷으로부터 불명료하지 않으며 정확하게 재생 또는 인쇄될 수 있다. 또한, 리치 페이로드가 중립적이고 어플리케이션에 의존하지 않는 방법으로 표현되기 때문에, 다큐먼트는 패키지를 생성하는데 사용되는 어플리케이션 없이도 일반적으로 뷰(view) 및 인쇄될 수 있다. 이러한 기능을 제공하기 위해, *고정된 페이로드(fixed payload)*라는 개념이 도입되고, 리치 패키지에 포함된다.

[0288] 본 실시예에 따르면, 고정된 페이로드는 고정된 수의 페이지들을 가지며, 페이지 분할은 항상 동일하다. 고정된 페이로드로 페이지상의 모든 요소들의 레이아웃이 선정된다. 각각의 페이지는 고정된 크기 및 방위를 갖는다. 소비 측에서는 아무런 레이아웃 계산도 실행되지 않으며, 콘텐츠가 간단하게 렌더링될 수 있다. 이는 단지 그래픽 뿐만 아니라 텍스트에도 적용이 되며, 정확한 인쇄상 위치로 고정된 페이로드에서 표현된다. 페이지의 콘텐츠(텍스트, 그래픽, 이미지)는 강력하지만 간단한 비쥬얼 프리미티브(visual primitives) 집합을 사용해서 기술된다.

[0289] 리치 패키지들은 페이지들을 구성하기 위한 다양한 메카니즘들을 지원한다. 일군의 페이지들은 차례대로 함께 "FixedPanel"로 "합쳐진다(glued)". 이러한 그룹의 페이지들은 대략 전통적인 멀티-페이지 다큐먼트와 동일하다. FixedPanel은 구성(composition)("복합" 다큐먼트를 조합하도록 시퀀스 및 선택을 구축하는 프로세스)에 더 참여할 수 있다.

[0290] 본 실시예에서, 리치 패키지들은 예를 들어, FixedPanel 집합을 함께 보다 큰 단일의 "다큐먼트"로 합치는데 사용될 수 있는 *FixedPanel 시퀀스*라고 하는 특정한 종류의 시퀀스를 지원한다. 예를 들어, 상이한 소스들로부터 온 두 다큐먼트들: 2 페이지 커버 메모(FixedPanel) 및 20 페이지 리포트(FixedPanel)를 함께 합친다고 상상해

보라.

[0291] 리치 패키지는 "동일한" 콘텐츠의 다른 표현들을 포함하는 다큐먼트 패키지들을 생성할 때 사용될 수 있는 다수의 특정 셀렉터들을 지원한다. 특히, 리치 패키지는 *언어(language)*, *컬러 기능(calor capability)*, 및 *페이지 크기(page size)*에 기초하여 선택이 될 수 있도록 한다. 따라서, 예를 들어, 셀렉터를 사용해서 다큐먼트의 영어 표현 및 불어 표현 중 하나를 선택하는 이중 언어(bi-lingual) 다큐먼트를 가질 수 있다.

[0292] 리치 패키지내의 구성에 대하여 이러한 셀렉터 및 시퀀스의 간단한 사용에 더하여, 셀렉터 및 시퀀스는 또한 그 이상의 셀렉터 및 시퀀스를 참조함으로써, 강력한 집합 계층구조들이 구축될 수 있도록 함을 주지한다. 본 실시예에 따르면, 실행될 수 있고 실행될 수 없는 것에 대한 정확한 규칙들이 "리치 패키지 구조"라는 제목의 장에서 후술된다.

[0293] 또한, 리치 패키지는 고정된 페이로드가 아니라, 다큐먼트의 보다 풍부하고 편집 가능한 표현들 일 수 있는 추가의 페이로드들을 포함할 수 있다. 이는 패키지로 하여금 시각적으로 정확하며 편집 어플리케이션 없이 뷰될 수 있는 표현 뿐만 아니라 에디터 어플리케이션에서 잘 동작하는 풍부하고 편집가능한 다큐먼트를 포함할 수 있게 해준다.

[0294] 마지막으로, 본 실시예에 따르면, 리치 패키지는 *인쇄 티켓(print ticket)*이라고 하는 것을 지원한다. 인쇄 티켓은 패키지가 인쇄될 때 사용돼야만 하는 설정치들을 제공한다. 이러한 인쇄 티켓들은 실질적인 융통성을 얻기 위한 다양한 방법들에 첨부될 수 있다. 예를 들어, 인쇄 티켓은 전체 패키지에 "첨부"될 수 있으며, 그 설정치들은 전체 패키지에 영향을 준다. 인쇄 티켓은 구조의 보다 하위 레벨에서(예를 들어, 개별 페이지들에) 더 부착될 수 있으며, 이러한 인쇄 티켓들은 첨부되는 파트들을 인쇄하는 경우 사용되어야 하는 오버라이드 설정치들을 제공하게 된다.

[0295] **리치 패키지 구조(The Reach Package Structure)**

[0296] 전술한 바와 같이, 리치 패키지는 "고정된" 페이지, FixedPanel, 구성(composition), 인쇄 티켓 등을 포함하는 일련의 특징을 지원한다. 이러한 특징은 패키지 모델의 핵심 컴포넌트들(파트 및 관계)을 사용하여 패키지 내에 표현된다. 본 장 및 관련 소제목에서, "리치 패키지"의 완전한 정의가 제공되는데, 모든 파트들 및 관계들이 조합되고, 관련되는 방법에 대한 설명을 포함한다.

[0297] **리치 패키지 구조 개요(Reach Package Structure Overview)**

[0298] 도 10은 일례의 리치 패키지를 나타낸 것으로, 본 실시예에서는, 패키지 내에서 구성하거나 발견될 수 있는 타당한 타입의 파트들의 각각을 나타낸다. 이하의 표는 각각의 타당한 파트 타입을 열거한 것이며 각각에 대한 설명을 제공한다:

[0299]

FixedPage application/xml+FixedPage-PLACEHOLDER	각각의 FixedPage 파트는 페이지의 콘텐츠를 표현한다.
FixedPanel application/xml+FixedPage-PLACEHOLDER	각각의 FixedPanel은 FixedPage 집합을 순서대로 함께 합친다.
Font	폰트들은 다큐먼트 도안(glyph)의 신뢰성 있는 재생을 보장하기 위해 패키지에 내장될 수 있다.
Image image/jpeg image/png	이미지 파트들이 포함될 수 있다.
Composition Parts application/xml+Selector+[XXX] Application/xml+Sequence+[XXX]	셀렉터 및 시퀀스가 보다 상위 레벨의 구조를 패키지에 도입하면서 "구성" 블록을 생성하는데 사용될 수 있다.
Descriptive Metadata application/xml+SimpleTypeProperties-PLACEHOLDER	기술적 메타데이터(예를 들어, 제목, 키워드)가 다큐먼트에 포함될 수 있다.
Print Ticket application/xml+PINTTICKET-PLACEHOLDER	인쇄 티켓이 패키지를 인쇄할 때 사용되는 설정치를 제공하도록 포함될 수 있다.

[0300] 리치 패키지는 "어느 곳에서나 뷰 및 인쇄"되는 다큐먼트가 되도록 디자인되기 때문에, 리치 패키지의 리더 및 라이터는 "타당한" 리치 패키지를 구성하는 것의 공통적이고 불명료하지 않게 정의된 기대치들을 공유해야만 한

다. "타당한" 리치 패키지의 정의를 제공하기 위해, 소수의 개념들이 먼저 정의된다.

[0301] 리치 구성 파트(Reach Composition Parts)

[0302] 리치 패키지는 패키지의 시작 파트로부터 구성 블록을 이동함으로써 "발견될 수 있는" 적어도 하나의 FixedPanel을 포함해야만 한다. 본 실시예에 따르면, 발견 프로세스(discovery process)는 이하의 알고리즘을 따른다:

[0303] • 패키지 시작 파트에서 시작해서 구성 파트들의 그래프를 재귀적으로 횡단한다.

[0304] • 상기 이동를 실행할 때, *리치 구성 파트들(reach composition parts)*(후술됨)인 구성 파트들로만 횡단한다.

[0305] • 그래프 에지에는 모든 터미널 노드들(밖으로 나가는 호(arc)들이 없는 노드들)의 위치를 정한다.

[0306] 터미널 노드는 (그 <item> 요소들을 통해) *리치 페이로드 루트(reach payload roots)*라고 하는 파트들의 집합을 일컫는다.

[0307] 고정된 페이로드(Fixed Payload)

[0308] 고정된 페이로드는 루트 파트가 FixedPanel 파트인 페이로드이다. 예를 들어, 도 10의 고정된 페이로드들 각각은 관련 FixedPanel 파트를 루트 파트로서 갖는다. 페이로드는 FixedPanel의 타당한 처리를 위해 요구되는 모든 파트들의 전체 종료(full closure)를 포함한다. 이들은 다음을 포함한다:

[0309] • FixedPanel 자체;

[0310] • FixedPanel 내에서 참조된 모든 FixedPage;

[0311] • 페이로드의 임의의 FixedPage에 의해 (직접, 또는 셀렉터를 통해 간접적으로) 참조되는 모든 이미지 파트들;

[0312] • 페이로드 내의 임의의 FixedPage 내에서 사용되는 이미지 브러시들로부터 직접 또는 간접적으로 참조되는 (후술된 바와 같은) 모든 *리치 셀렉터들(reach selectors)*;

[0313] • 페이로드의 임의의 FixedPage에 의해 참조되는 모든 폰트 파트들;

[0314] • 고정된 페이로드의 임의의 파트에 첨부되는 모든 기술적 메타데이터 파트들; 및

[0315] • 고정된 페이로드의 임의의 파트에 첨부된 임의의 인쇄 티켓들.

[0316] 리치 패키지에 대한 타당성 규칙(Validity Rules for Reach Package)

[0317] 전술한 정의들에 있어서, 본 실시예에 따라 "타당한" 리치 패키지를 기술하는 순응 규칙(conformance rule)들을 이하 설명한다:

[0318] • 리치 패키지는 전술한 바와 같이 패키지 관계의 표준 메카니즘을 사용하여 정의된 시작 파트를 포함하여야 한다;

[0319] • 리치 패키지의 시작 파트는 셀렉터 또는 시퀀스이어야 한다;

[0320] • 리치 패키지는 FixedPanel인 적어도 하나의 페이로드의 루트를 가져야만 한다;

[0321] • PrintTicket 파트들은 구성 파트들, FixedPanel 파트들 또는 FixedPanel(들)에서 식별된 임의의 FixedPage 파트들 중 임의의 파트에 첨부될 수도 있다. 본 예에서는, http://PLACEHOLDER/HasPrintTicketRel 관계를 통해 이루어진다;

[0322] ◦ PrintTicket은 상기 파트들 중 임의의 파트 또는 모든 파트들에 첨부될 수도 있다;

[0323] ◦ 임의의 주어진 파트가 하나 이상의 첨부된 PrintTicket을 가져서는 안된다;

[0324] • 기술적 메타데이터 파트는 패키지의 임의의 파트에 첨부될 수도 있다;

[0325] • FixedPayload의 모든 폰트 객체는 "폰트 파트"의 장에서 정의되는 폰트 포맷 규칙들을 만족시켜야만 한다.

[0326] • 고정된 페이로드의 임의의 FixedPage 내에서 화상에 대한 참조자들은 렌더링 대상의 실제 이미지 파트를 찾기 위해 (다른 셀렉터들을 통해 잠정적으로 재귀적으로) 선택할 수도 있는 셀렉터를 가리킬 수도 있다;

[0327] • 고정된 페이로드에서 사용되는 모든 이미지 객체는 "이미지 파트"의 장에서 정의되는 폰트 포맷 규칙들을 만족시켜야만 한다;

[0328] • (직접, 또는 셀렉터를 통해 간적접으로) FixedPage로부터 참조되는 임의의 폰트, 이미지 또는 셀렉터 파트의 경우, 참조하는 FixedPage로부터 참조되는 파트로의 "요구되는 파트" 관계가 있어야만 한다(관계 네임 = http://mmcf-fixed-RequiredResource-PLACEHOLDER).

[0329] **리치 구성 파트(Reach Composition Parts)**

[0330] 리치 패키지가 다수의 타입들의 구성 파트를 포함할 수도 있지만, 잘 정의된 구성 파트의 타입들의 집합만이 본 다큐먼트에 따라서 잘 정의된 동작을 갖는다. 잘 정의된 동작을 갖는 구성 파트들을 *리치 구성 파트(reach composition parts)*들이라고 한다. 그 외의 파트들은 리치 패키지의 타당성을 결정하는 경우에는 적절하지 않다.

[0331] 이하의 타입들의 구성 파트들은 리치 구성 파트들로서 정의된다:

[0332]

언어 셀렉터 application/xml+selector+language	자연 언어에 기초하여 표현들 중에서 선택한다
컬러 셀렉터 application/xml+selector+color	단색인지 컬러인지에 기초하여 표현들 중에서 선택한다
페이지 크기 셀렉터 application/xml+selector+pagesize	페이지 크기에 기초하여 표현들 중에서 선택한다
콘텐츠 타입 셀렉터 application/xml+selector+contenttype	콘텐츠 타입들이 시스템에 의해 이해될 수 있는지에 기초하여 표현들 중에서 선택한다
고정 시퀀스 application/xml+sequence+fixed	고정 콘텐츠인 자식을 시퀀스로 결합한다

[0333] 리치 셀렉터(Reach Selectors)

[0334] 리치 구성 파트로 정의된 셀렉터 구성 파트는 *리치 셀렉터(reach selectors)*라고 한다. 전술한 바와 같이, *언어 셀렉터(language selector)*는 영어 또는 불어와 같은 자연 언어에 기초하여 표현들 중에서 선택한다. 상기 언어를 발견하기 위해, 셀렉터는 각각의 아이템들을 검사한다. XML인 것만 고려된다. 각각의 파트의 루트 요소가 그 언어를 결정하기 위해 검사된다. xml:lang 속성이 존재하지 않으면, 파트는 무시된다. 셀렉터는 상기 파트들 각각을 차례로 고려하는데, 언어가 시퀀스의 디폴트 언어와 매치하는 첫번째 파트를 선택한다.

[0335] *컬러 셀렉터(color selector)*는 단색인지 컬러인지에 기초하여 표현들 중에서 선택한다. *페이즈 크기 셀렉터(page size selector)*는 페이지 크기에 기초하여 표현들 중에서 선택한다. *콘텐츠 타입 셀렉터(content type selector)*는 그 콘텐츠 타입이 시스템에 의해 인식될 수 있는지에 기초하여 표현들 중에서 선택한다.

[0336] 리치 시퀀스(Reach Sequences)

[0337] 리치 구성 파트로서 정의된 그러한 시퀀스 구성 파트는 *리치 시퀀스(reach sequence)*라고 한다. *고정 시퀀스(fixed sequence)*는 고정 콘텐츠인 자식을 시퀀스로 결합한다.

[0338] **고정된 페이로드 파트(Fixed Payloads Parts)**

[0339] 고정된 페이로드는 이하의 종류들의 파트들: FixedPanel 파트, FixedPage 파트, 이미지 파트, 폰트 파트, 인쇄

티켓 파트, 및 기술적 메타데이터 파트를 포함할 수 있는데, 이하, 각각의 파트를 자신의 소제목 하에 설명한다.

[0340] **FixedPanel 파트**

[0341] Fixed-Payload의 다큐먼트 구조는 후술되는 바와 같이, 스파인의 파트로서 FixedPage를 식별한다. 스파인 파트 및 페이지 파트의 관계는 스파인의 관계들 스트림(relationships stream) 내에 정의된다. FixedPanel 파트는 콘텐츠 타입 application/xml+PLACEHOLDER의 것이다.

[0342] Fixed-Payload 콘텐츠의 스파인은 <Document> 요소 내에 <FixedPanel> 요소를 포함함으로써 마크업에서 지정된다. 아래의 예에서, <FixedPanel> 요소는 스파인에 유지되는 페이지들의 소스들을 지정한다.

[0343]
```
<!-- SPINE -->
<Document $XMLNSFIXED$ >
        <FixedPanel>
                <PageContent Source="p1.xml" />
                <PageContent Source="p2.xml" />
        </FixedPanel>
</Document>
```

[0344] <Document> 요소

[0345] <Document> 요소는 속성들을 갖지 않으며, 오직 하나의 자식: <FixedPanel>을 가져야만 한다.

[0346] <FixedPanel> 요소

[0347] <FixedPanel> 요소는 다큐먼트 스파인(document spine)이며, 순서화된 시퀀스의 페이지들을 함께 단일 멀티-페이지 다큐먼트로 논리적으로 결합한다. 페이지들은 항상 자신의 폭 및 높이를 지정하지만, <FixedPanel> 요소는 높이 및 폭을 선택사항으로 지정할 수도 있다. 이러한 정보는 예를 들어, 페이지 크기에 기초하여 다른 표현들중 선택하는 것을 포함해서, 다양한 목적을 위해 사용될 수 있다. <FixedPanel> 요소가 높이 및 폭을 지정하면, 통상 <FixedPanel> 내에서 페이지들의 폭 및 높이에 따라 정렬되는데, 그 치수가 개별 페이지들의 높이 및 폭을 지정하는 것은 아니다.

[0348] 이하의 표는 본 실시예에 따른 FixedPanel 속성들을 요약한다.

[0349]

<FixedPanel> 속성	설명
PageHeight	<FixedPanel>에 포함된 페이지들의 전형적인 높이. 선택사항.
PageWidth	<FixedPanel>에 포함된 페이지들의 전형적인 폭. 선택사항.

[0350] <PageContent> 요소는 <FixedPanel> 요소의 오직 허용 가능한 자식 요소이다. <PageContent> 요소들은 다큐먼트의 페이지 순서와 매칭하는 순차적인 마크업 순서이다.

[0351] <PageContent> 요소

[0352] <PageContent> 요소는 단일 페이지용 콘텐츠의 소스를 말한다. 다큐먼트의 페이지들의 수를 결정하기 위해, <FixedPanel> 내에 포함된 <PageContent> 자식의 수를 카운트한다.

[0353] <PageContent> 요소가 허용 가능한 자식을 갖지 않으며, 페이지의 콘텐츠에 대해 FixedPage 파트라고 일컫는, 단일의 요구되는 속성으로 Source를 갖는다.

[0354] <FixedPanel> 요소에서와 같이, <PageContent> 요소는 단일 페이지의 크기를 반영하는 PageHeight 및 PageWidth 속성을 선택사항으로 포함할 수도 있다. 요구되는 페이지 크기는 FixedPage 파트에서 지정되며; <PageContent>에 대한 선택적인 크기는 권장사항일 뿐이다. <PageContent> 크기 속성들은 다큐먼트 뷰어와 같은 어플리케이션들이 개별 FixedPage 파트들 모두를 로딩 및 파싱하지 않고, 신속하게 다큐먼트에 대한 시각적인 레이아웃 추정치를 만들 수 있도록 한다.

[0355] 이하의 표는 <PageContent> 속성들을 요약하고 속성 기술을 제공한다.

[0356]

<PageContent> 속성	설명
Source	패키지 내의 개별 파트에 유지되는 페이지 콘텐츠를 일컫는 URI 문자열. 콘텐츠는 패키지 내의 파트로서 식별된다. 요구됨

PageHeight	선택적임
PageWidth	선택적임

[0357] 페이지 콘텐츠의 URI 문자열은 패키지에 대한 콘텐츠의 파트 위치를 참조하여야 한다.

[0358] **FixedPage 파트**

[0359] <FixedPanel>의 각각의 <PageContent> 요소는 네임(URI)으로 FixedPage 파트를 참조한다. 각각의 FixedPage 파트는 단일 페이지의 콘텐츠를 렌더링하는 것을 기술하는 FixedPage 마크업을 포함한다. FixedPage 파트는 Content Type application/xml+PLACEHOLDER-FixedPage의 것이다.

[0360] 마크업의 FixedPage 설명

[0361] 이하는 소스 콘텐츠의 마크업이 상기 샘플 스파인 마크업에서 참조되는 페이지를 찾을 수도 있는 방법의 일례이다(<PageContent Source = "p1.xml" />).

[0362]
```
//    /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>
```

[0363] 이하의 표는 FixedPage 성질을 요약하며, 그 설명을 제공한다.

[0364]

FixedPage 성질	설명
PageHeight	요구됨
PageWidth	요구됨

[0365] FixedPage 마크업의 판독 순서

[0366] 일 실시예에서, FixedPage 내에 포함된 Glyphs 자식 요소들의 마크업 순서는 페이지의 텍스트 콘텐츠의 희망하는 판독 순서와 동일해야만 한다. 이러한 판독 순서는, 뷰어의 FixedPage로부터의 순차적인 텍스트의 상호작용적인 선택/복사에, 및 액세스 기술에 의해 순차적인 텍스트로 액세스 하도록 하는 것에 모두 사용될 수도 있다. 마크업 순서와 판독 순서 간의 대응성을 보장하는 것은 FixedPage 마크업을 생성하는 어플리케이션의 책임이다.

[0367] **이미지 파트(Image Parts)**

[0368] 지원 포맷(Supported Formats)

[0369] 본 실시예에 따르면, 다른 포맷들이 사용될 수도 있지만, 리치 패키지의 FixedPage에 의해 사용되는 이미지 파트는 고정된 수의 포맷, 예를 들어, PNG 또는 JPEG로 될 수 있다.

[0370] **폰트 파트(Font Parts)**

[0371] 본 실시예에 따르면, 리치 패키지는 제한된 수의 폰트 포맷들을 지원한다. 본 실시예에서, 지원되는 폰트 포맷은 TrueType 포맷 및 OpenType 포맷을 포함한다.

[0372] 당업자에게는 자명한 바와 같이, OpenType 폰트 포맷은 TrueType 폰트 포맷의 확장으로서, PostScript 폰트 데이터 및 복합적인 인쇄 레이아웃 지원을 추가한다. OpenType 폰트 파일은 PostScript 아웃라인 폰트 또는 TrueType 아웃라인 폰트를 포함하는 표 포맷의 데이터를 포함한다.

[0373] 본 실시예에 따르면, 이하의 폰트 포맷들은 리치 패키지에서 지원되지 않는다: Adobe 타입 1, 비트맵 폰트, 숨겨진 속성의 폰트(계수 여부를 결정하기 위해 시스템 Flag를 사용함), 벡터 폰트, 및 EUDC 폰트(폰트 패밀리 네임이 EUDC).

[0374] 폰트의 서브세팅(Subsetting Fonts)

[0375] 고정된 페이로드는 이하 상술하는 *Glyphs* 요소를 사용하는 모든 텍스트를 표현한다. 본 실시예에서, 포맷이 고정되기 때문에, FixedPayload에 의해 요구되는 도안(glyph)만을 포함하도록 폰트들을 서브세팅할 수 있다. 따

라서, 리치 패키지의 폰트들은 도안의 용도에 기초하여 서브세팅될 수도 있다. 서브세팅된 폰트가 원래의 폰트의 모든 도안들을 포함하지 않더라도, 서브세팅된 폰트는 타당한 OpenType 폰트 파일이어야만 한다.

[0376] **인쇄 티켓 파트(Print Ticket Parts)**

[0377] 인쇄 티켓 파트들은 패키지가 인쇄될 때 사용될 수 있는 설정치들을 제공한다. 이러한 인쇄 티켓은 실질적인 융통성(flexibility)을 성취하도록 다양한 방법들로 첨부될 수 있다. 예를 들어, 인쇄 티켓은 전체 패키지에 "첨부"될 수 있으며, 설정치는 전체 패키지에 영향을 주게 된다. 인쇄 티켓은 구조의 보다 하위의 레벨에서(예를 들어, 개별 페이지들에) 더 첨부될 수 있으며, 인쇄 티켓은 첨부된 파트를 인쇄할 때 사용되는 오버라이드 설정치들을 제공한다.

[0378] **기술적 메타데이터(Descriptive Metadata)**

[0379] 전술한 바와 같이, 기술적 메타데이터 파트는 패키지의 리더가 값을 신뢰성 있게 발견할 수 있게 해주는 속성의 값들을 저장하는 방법을 패키지의 라이터 또는 생산자에게 제공한다. 이러한 속성은 통상 저장소 내의 개별 파트들 뿐만 아니라 패키지에 대한 추가 정보를 전체적으로 기록하는데 사용된다.

[0380] **FixedPage 마크업 기본 사항**

[0381] 본 장은 FixedPage 마크업과 관련된 몇몇 기본 정보를 기술하며, 이하의 섹션들: "고정된 페이로드 및 다른 마크업 표준들", "FixedPage 마크업 모델", "리소스 및 리소스 참조자", 및 "FixedPage 드로잉 모델"을 포함한다.

[0382] **고정된 페이로드 및 다른 마크업 표준들(Fixed Payloads and Other Markup Standards)**

[0383] 리치 패키지의 고정된 페이로드를 위한 FixedPanel 및 FixedPage 마크업은 윈도우즈® Longhorn의 Avalon XAML 마크업으로부터의 부분집합이다. 즉, 고정된 페이로드 마크업이 독립적인 XML 마크업 포맷(본 다큐먼트에 다큐먼트화됨)으로서 혼자이지만, Longhorn 시스템과 동일한 방법으로 로드하고, 원본의 멀티-페이지 다큐먼트의 WYSIWYG 재생을 렌더링한다.

[0384] XAML 마크업의 몇몇 배경으로서, 다음을 고려해 본다. XAML 마크업은 사용자가 객체들의 계층 및 객체들 뒤의 프로그래밍 로직을 XML 베이스 마크업 언어로서 지정할 수 있게 해주는 메카니즘이다. 이는 객체 모델이 XML로 기술되게 하는 기능을 제공한다. 이는 마이크로소프트 코포레이션의 .NET 프레임워크의 CLR(Common Language Runtime)의 클래스들과 같은 확장 가능 클래스들이 XML로 액세스될 수 있게 해준다. XAML 메카니즘은 XML 태그들의 CLR 객체들로의 다이렉트 매핑 및 마크업으로 관련 코드를 표현하는 기능을 제공한다. 다양한 구현예들이 XAML의 CLR-기반의 구현을 특정적으로 사용할 필요가 없음을 알 것이다. CLR-기반의 구현예는 오히려 XAML이 본 명세서에 기술된 실시예들의 문맥에서 사용될 수 있는 한가지 방법을 구성한다.

[0385] 더욱 자세하게는, 도 11과 연계하여 이하를 고려해 본다. 도 11은 CLR 개념(좌측 컴포넌트)의 XML(우측 컴포넌트)로의 일실시예의 매핑을 나타낸다. 네임스페이스는 반사(reflection)이라고 하는 CLR 개념을 사용하여 xmlns 선언에서 발견된다. 클래스들은 직접 XML 태그들에 매핑된다. 특성(properties) 및 이벤트(events)들은 속성(attributes)들에 직접적으로 매핑된다. 이러한 계층구조를 사용하여, 사용자는 XML 마크업 파일들의 임의의 CLR 객체들의 계층구조 트리를 지정할 수 있다. Xaml 파일들은 .xaml 확장자 및 application/xaml+xml의 미디어타입을 갖는 xml 파일들이다. Xaml 파일들은 통상 xmlns 속성을 사용하여 네임스페이스를 지정하는 하나의 루트 태그를 갖는다. 네임스페이스는 다른 타입들의 태그들로 지정될 수도 있다.

[0386] 계속해서, xaml 파일의 태그들은 일반적으로 CLR 객체에 매핑된다. 태그는 요소, 복합 속성, 정의 또는 리소스일 수 있다. 요소들은 실행시간 중에 일반적으로 예증되는 CLR 객체이며 객체들의 계층구조를 형성한다. 복합 특성(compound property) 태그는 부모 태그의 특성을 설정하는데 사용된다. 정의(definition) 태그는 페이지에 코드를 추가하고 리소스를 정의하는데 사용된다. 리소스(resource) 태그는 단지 트리를 리소스로 지정함으로써 객체의 트리를 재사용하는 기능을 제공한다. 정의 태그는 또 다른 태그 내에서 xmlns 속성으로서 정의될 수도 있다.

[0387] 일단 다큐먼트가 마크업으로 적합하게 기술되면(통상 라이터에 의해), 마크업은 (통상 리더에 의해) 파싱 및 처리될 수 있다. 적합하게 구성된 파서(parser)는 루트 태그로부터 어떤 CLR 어셈블리 및 네임스페이스가 태그를 찾기 위해 탐색되어야 하는지를 결정한다. 다수의 경우에 있어서, 파서는 xmlns 속성으로 지정된 URL에서 네임스페이스 정의 파일을 탐색해서 찾는다. 네임스페이스 정의 파일은 어셈블리 및 설치 경로의 네임 및 CLR 네임스페이스 리스트를 제공한다. 파서가 태그와 만날 때, 파서는 어떤 CLR 클래스가 태그와 관련되는지를 태그의

xmlns 및 해당 xmlns의 xmlns 정의 파일을 사용해서 결정한다. 파서는 어셈블리 및 네임스페이스가 정의 파일에 열거되어 있는 순서로 탐색한다. 매치를 찾은 경우, 파서는 클래스의 객체를 예증한다(instantiate).

[0388] 따라서, 상기 참조로 포함되는 어플리케이션에서 보다 충실하게, 전술한 메카니즘은 객체 모델이 마크업 태그를 사용해서 XML 기반의 파일로 표현되게 한다. 객체 모델을 마크업 태그로서 표현하는 이러한 기능은 벡터 그래픽 드로잉, 고정 포맷 다큐먼트, 적응적 플로우 다큐먼트, 및 어플리케이션 UI들을 비동기적으로 또는 동기적으로 생성하는데 사용될 수 있다.

[0389] 본 실시예에서, 고정된 페이로드(Fixed Payload) 마크업은 Avalon XAML 렌더링 프리미티브들의 매우 극미한(nearly completely parsimonious) 부집합(subset)이다. 이것은 Avalon으로 표현될 수 있는 어떠한 것이라도 최대 충실도로 시각적으로 표현한다. 고정된 페이로드 마크업은 Avalon XAML 요소들 및 특성들 + 추가 관례들, 규범적인 형태들, 또는 Avalon XAML에 대한 사용시의 제한 사항들의 부집합이다.

[0390] 정의된 근본적으로 최소한의 고정된 페이로드 마크업 집합은 프린터 RIP 또는 인터랙티브 뷰어 어플리케이션들과 같은 리치 패키지 리더들의 구현 및 테스팅과 관련된 비용을 감소시키며, 관련 파서의 복잡성 및 메모리 풋프린트(footprint)도 감소시킨다. 극미한 마크업 집합은 또한, 포맷 및 그 에코시스템을 내부적으로 더 견고하게 하면서, 리치 패키지 라이터들 및 리더들 사이의 서브세팅의 기회, 에러 또는 불일치들을 최소화한다.

[0391] 최소의 Fixed Payload 마크업에 더하여, 리치 패키지는 하이퍼링크, 섹션/아웃라인 구조 및 네비게이션, 텍스트 선택, 및 다큐먼트 액세스 가능성(accessibility)과 같은 기능들을 갖는 리치 패키지 다큐먼트의 표현들 또는 뷰어들을 지원하기 위해 추가의 시만틱(semantic) 정보에 대한 마크업을 지정한다.

[0392] 끝으로, 전술한 버전화 및 확장 가능성 메카니즘을 사용해서, 최소의 고정된 페이로드 마크업을 특정 타겟의 소비 어플리케이션, 뷰어, 또는 디바이스들을 위한 보다 풍부한 집합의 요소들로 보충할 수 있다.

[0393] **FixedPage 마크업 모델**

[0394] 본 실시예에서, FixedPage 파트는 XML-요소, XML-속성, 및 XML-네임스페이스에 기초하여 XML 기반의 마크업 언어로 표현된다. 본 다큐먼트에서 3개의 XML 네임스페이스들이 FixedPage 마크업에 포함되도록 정의된다. 이러한 네임스페이스는 본 명세(specification)의 어디에서든지 정의되는 버전-컨트롤 요소들 및 속성들을 참조한다. FixedPage 마크업에서 요소들 및 속성들을 위해 사용된 주요 네임스페이스는 "http://schemas.microsoft.com/MMCF-PLACEHOLDER-FixedPage"이다. 마지막으로, FixedPage 마크업은 후술된 제3 네임스페이스를 요구하는 "리소스"의 개념을 도입한다.

[0395] FixedPage 마크업이 XML-요소 및 XML-속성을 사용해서 표현되더라도, 그 명세는 "콘텐츠" 및 "속성들"의 보다 상위 레벨의 추상 모델에 기초한다. FixedPage 요소들은 모두 XML 요소들로서 표현된다. 오직 소수의 FixedPage 요소들만이 자식 XML-요소들로 표현된 "콘텐츠"를 유지할 수 있다. 특성-값(property value)은 XML-속성 또는 자식 XML-요소를 사용해서 표현될 수도 있다.

[0396] FixedPage 마크업은 또한 리소스-사전(Resource-Dictionary) 및 리소스-참조자(Resource-Reference)의 쌍둥이 개념에 의존한다. 하나의 리소스-사전 및 다수의 리소스-참조자들의 결합은 단일의 속성-값이 다수의 FixedPage 마크업 요소들의 다수의 속성들에 의해 공유될 수 있도록 한다.

[0397] <u>FixedPage 마크업의 특성들(Properties in FixedPage Markup)</u>

[0398] 본 실시예에서, FixedPage 마크업 특성의 값을 지정하는데 사용될 수 있는 3개의 마크업 형태들이 있다.

[0399] 특성이 리소스-참조자를 사용해서 지정되는 경우, 특성 네임은 XML-attribute 네임으로서 사용되고, 속성-값(attribute value)을 위한 특수한 구문(syntax)은 리소스 참조자가 있음을 나타낸다. 리소스-참조자를 표현하기 위한 구문은 "리소스 및 리소스-참조자"라는 제목의 장에 설명된다.

[0400] 리소스-참조에 지정되지 않은 어떠한 특성-값(property value)이라도 그 값이 설정중인 특성을 식별하는 중첩 자식 XML-요소(nested child XML-element)를 사용해서 XML로 표현될 수도 있다. 이하, 이러한 "복합-특성 구문"을 설명한다.

[0401] 마지막으로, 몇몇 리소스-참조가 아닌 속성-값들은 간단한 텍스트 문자열로서 표현될 수 있다. 모든 특성-값(property-value)들이 복합-특성 구문(Compound-Property Syntax)을 사용해서 표현될 수 있더라도, 간단한 XML-속성 구문(XML-attribute syntax)을 사용해서 표현될 수도 있다.

[0402] 임의의 주어진 요소에 있어서, 어떠한 속성이라도 값을 지정하는데 사용되는 구문과 무관하게 한 번 이하로 설정될 수도 있다.

[0403] 간단한 특성 구문(Simple Attribute Syntax)

[0404] 간단한 문자열로 표현될 수 있는 특성 값의 경우, XML-속성-구문이 특성-값을 지정하는데 사용될 수도 있다. 예를 들어, "Color"라고 하는 특성을 가진 "SolidColorBrush"라고 하는 FixedPage 마크업 요소가 주어지면, 이하의 구문은 특성값을 지정하는데 사용될 수 있다:

[0405]
```
<!-- Simple Attribute Syntax -->
<SolidColorBrush Color="#FF0000" />
```

[0406] 복합-특성 구문(Compound-Property Syntax)

[0407] 어떤 특성 값들은 간단한 문자열로로 표현될 수 없다. 예를 들어, XML-요소는 특성-값을 기술하는데 사용된다. 이러한 특성 값은 간단한 속성 구문을 사용해서 표현될 수 없지만, 복합-특성 구문을 사용해서 표현될 수 있다.

[0408] 복합-특성 구문에서, 자식 XML-요소가 사용되지만, XML-요소 네임은 점(.)으로 분리된 부모-요소 네임 및 특성 네임의 결합으로부터 유도된다. <SolidColorBrush>로 설정될 수도 있는 속성 "Fill"을 갖는 FixedPage 마크업 요소 <Path>가 주어지면, 이하의 마크업이 <Path> 요소의 "Fill" 특성을 설정하는데 사용될 수 있다:

[0409]
```
<!-- Compound-Property Syntax -->
<Path>
       <Path.Fill>
              <SolidColorBrush Color="#FF0000" />
       </Path.Fill>
       …
</Path>
```

[0410] 복합-특성 구문은 Simple-Attribute 구문이 특성-값을 표현하는데 충분한 경우에도 사용될 수 있다. 따라서, 앞 장의 예는 이하와 같다:

[0411]
```
<!-- Simple Attribute Syntax -->
<SolidColorBrush Color="#FF0000" />
```

[0412] 복합-특성 구문으로 대신 표현될 수 있다:

[0413]
```
<!-- Compound-Property Syntax -->
<SolidColorBrush>
       <SolidColorBrush.Color>#FF0000</SolidColorBrush.Color>
</SolidColorBrush>
```

[0414] 복합-특성 구문을 사용해서 특성-값을 지정하는 경우, "특성들(properties)"을 표현하는 자식 XML-요소들은 "콘텐츠"를 표현하는 자식 XML-요소들 앞에 나타나야만 한다. 개별 복합-특성 자식 XML-요소들의 순서는 중요하지 않으며, 오직 부모-요소의 임의의 "콘텐츠" 전에 함께 나타난다.

[0415] 예를 들어, <Canvas> 요소(후술됨)의 Clip 및 RenderTransform 속성들을 사용할 때, 둘 다 <Canvas>의 임의의 <Path> 및 <Glyphs> 전에 나타나야만 한다:

[0416]
```
<Canvas>
       <!-- First, the property-related child elements -->
       <Canvas.RenderTransform>
              <MatrixTransform Matrix="1,0,0,1,0,0">
       </Canvas.RenderTransform>
       <Canvas.Clip>
              <PathGeometry>
                     …
              </PathGeometry>
       </Canvas.Clip>

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

[0417] **리소스 및 리소스 참조자(Resources and Resource References)**

[0418] 리소스 사전들은 각각 리소스라고 하는 공유 가능한 특성 값들을 유지하는데 사용될 수 있다. FixedPage 마크업 요소 자체인 임의의 특성 값은 리소스 사전에 유지될 수도 있다. 리소스 사전의 각각의 리소스는 네임을 운반한다. 리소스 네임은 속성의 XML-속성으로부터 리소스를 참조하는데 사용될 수 있다.

[0419] 본 실시예에서, <Canvas> 및 <FixedPage> 요소들은 리소스 사전을 운반할 수 있다. 리소스 사전은 "Resources"라고 하는 특성의 <Canvas> 및 <FixedPage> 요소들의 특성으로 마크업에서 표현된다. 그러나, 개별 리소스 값들은 <FixedPage.Resources> 또는 <Canvas.Resources> XML-요소 내에 직접 내장된다. 구문적으로, <Canvas.Resources> 및 <FixedPage.Resources>에 대한 마크업은 "콘텐츠"를 갖는 마크업 요소들과 유사하다.

[0420] 본 실시예에 따르면, <Canvas.Resources> 또는 <FixedPage.Resources>는 <Canvas> 또는 <FixedPage>의 임의의 복합-특성-구문 특성 값들을 우선해야만 한다. 또한, 마찬가지로 <Canvas> 또는 <FixedPage>의 임의의 "콘텐츠" 보다 우선해야만 한다.

[0421] 고정된 페이로드 리소스 사전의 정의(Defining Fixed-Payload Resource Dictionaries)

[0422] 임의의 <FixedPage> 또는 <Canvas>는 <Canvas.Resources> XML-요소를 사용해서 표현된 리소스 사전을 운반할 수 있다. 단일 리소스 사전 내의 각각의 요소는 요소와 관련된 XML-속성을 사용해서 식별되는 고유한 네임을 갖는다. 특성(properties)에 대응하는 속성(attribute)들로부터 이러한 "네임" 속성을 구별하기 위해, 네임 속성이 FixedFormat 요소들의 네임스페이스가 아닌 네임스페이스로부터 취해진다. XML-네임스페이스에 대한 URI는 "http://schemas.microsoft.com /PLACEHOLDER-for-resources"이다. 이하의 일례에서, 두 기하형태가 정의되는데, 하나는 장방형을 위한 것이고 다른 하나는 원을 위한 것이다.

```
<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>
```

[0423]

[0424] 리소스 참조(Referencing Resources)

[0425] 상기 정의된 리소스들 중 하나로 특성 값을 설정하기 위해, 리소스 네임을 { }로 묶는 XML-속성 값을 사용하라. 예를 들어, "{Rectangle}"은 사용될 기하형태를 나타낸다. 이하의 마크업 샘플에서, 사전의 기하형태 객체들에 의해 정의된 장방형 영역은 SolidColorBrush로 채워진다.

```
<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>
```

[0426]

[0427] 본 실시예에 따르면, 리소스 참조는 리소스 사전의 리소스 정의 내에서 발생하지 않아야만 한다.

[0428] 리소스 참조 해제를 위한 스코프 규칙

[0429] 단일 네임이 동일한 리소스 사전에서 두 번 사용되지 않을 수도 있지만, 동일한 네임이 단일 FixedPage 파트 내의 두 개의 상이한 리소스 사전들에서 사용될 수도 있다. 또한, 내부 <Canvas>의 리소스 사전은 몇몇 외부 <Canvas> 또는 <FixedPage>의 리소스 사전에서 정의된 네임을 사용할 수도 있다.

[0430] 리소스 참조가 요소의 속성을 설정하는데 사용될 때, 다양한 리소스 사전들이 소정의 네임의 리소스에 대해 탐색된다. 속성을 가진 요소가 <Canvas>이면, <Canvas>의 리소스 사전은(존재하는 경우) 희망하는 네임의 리소스에 대해 탐색된다. 요소가 <Canvas>가 아니면, <Canvas> 또는 <FixedPage>를 포함하는 탐색은 가장 가까운 것부터 시작된다. 희망하는 네임이 초기에 탐색된 리소스 사전에서 정의되지 않으면, <Canvas> 또는 <FixedPage>를 포함하는 다음으로 가장 가까운 것이 고려된다. 탐색이 루트 <FixedPage> 요소까지 계속되고, 희망 네임의 리소스가 <FixedPage>와 관련된 리소스 사전에서 발견되지 않으면 에러가 발생한다.

[0431] 이하의 일례는 이러한 규칙들을 설명한다.

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

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

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

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

</FixedPage>
```

[0432]

[0433] **FixedPage 드로잉 모델(Drawing Model)**

[0434] FixedPage(또는 중첩 Canvas 자식(nested Canvas child)) 요소는 다른 요소들이 렌더링되는 요소이다. 콘텐츠 배열은 FixedPage(또는 Canvas)에 대해 지정된 특성들, FixedPage(또는 Canvas)의 요소들에 대해 지정된 특성들, 및 고정된 페이로드 네임스페이스에 대해 정의된 구성 규칙(compositional rules)들에 의해 제어된다.

[0435] 요소들을 위치시키기 위해 Canvas를 사용

[0436] 고정 마크업에서, 모든 요소들은 좌표 시스템의 현 원점 (0,0)과 상대적으로 위치된다. 현재의 원점은 요소를 포함하는 FixedPage 또는 Canvas의 각각의 요소에 RenderTransform 속성을 적용함으로써 이동될 수 있다.

[0437] 이하의 일례는 RenderTransform을 통한 요소들의 위치 결정을 도시한다.

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

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

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

[0438]

[0439] 좌표 시스템 및 유닛(Coordinate Systems and Units)

[0440] 본 실시예에 따르면, 좌표 시스템은 초기에 좌표 시스템의 한 유닛이 부동 소수점 값으로 표현된 1인치의 1/96과 동일하고, 좌표 시스템의 원점 (0,0)이 FixedPage 요소의 좌측 상부 코너가 되도록 설정된다.

[0441] RenderTransform 속성은 현재의 좌표 시스템에 유사 변환(affine transform)을 적용하기 위해 임의의 자식 요소에 지정될 수 있다.

[0442] 페이지 치수(Page Dimensions)

[0443] 페이지 크기는 FixedPage 요소에서 "PageWidth" 및 "PageHeight" 파라미터들로 지정된다.

[0444] 구성 규칙(Composition Rules)

[0445] FixedPage는 알파 채널을 갖는 페인터 모델을 사용한다. 본 실시예에 따르면, 구성은 상기 규칙들에 따라 이하의 순서로 발생해야만 한다:

[0446] • FixedPage(또는 임의의 중첩 Canvas)는 마크업에 나타나는 순서로 자식 요소들이 그려지는 무한 표면으로서 생각된다. 상기 표면의 알파 채널은 "0.0"(모두 투명)으로 초기화된다. 실제로, 이상적인 무한 표면은 모든 자식 요소들을 렌더링함으로써 생성된 모드 마크들을 유지할만큼 충분히 큰 비트맵 버퍼로 생각될 수 있다.

[0447] • 표면의 콘텐츠는 FixedPage(또는 Canvas)의 RenderTransform 속성에 의해 지정된 유사 변환을 사용해서 변환된다.

[0448] • 모든 자식 요소들은 FixedPage(또는 Canvas)의 Clip 속성에 의해 클립된 표면으로 렌더링된다(이 또한 RenderTransform 속성을 사용해서 변환됨). 또한 FixedPage는 (0,0,PageWidth,PageHeight)에 의해 지정된 장방형으로 클립한다. 자식 요소가 Opacity 속성 또는 OpacityMask 속성을 가지면, 표면으로 렌더링되기 전에 자식 요소에 적용된다.

[0449] • 마지막으로, FixedPage(또는 Canvas)의 콘텐츠는 그 포함하는 요소로 렌더링된다. FixedPage의 경우에, 포함하는 요소는 물리적 이미징 표면이다.

[0450] 상기 규칙들에 따라 렌더링이 발생한다:

[0451] • 오직 표면상의 마크들을 생성하는 요소들은 "Glyphs" 및 "Path"이다.

[0452] • 다른 모든 렌더링 효과들은 "Glyphs" 및 "Path" 요소들을 "Canvas"상에 위치시키고, 각종 유효 속성들을 적용함으로써 달성될 수 있다.

[0453] **고정된 페이로드 요소들 및 속성들**

[0454] 본 실시예에 따르면, 고정된 페이로드는 페이지들과 그 내용들을 표현하기 위해 마크업에서 사용되는 XML 요소들의 소집합을 포함한다. FixedPanel 파트의 마크업은 <Document>, <FixedPanel> 및 <PageContent> 요소들을 사용해서 다큐먼트 페이지들을 함께 합쳐서 쉽게 인덱싱되는 공통 루트를 야기한다. 각각의 FixedPage 파트는 오직 <Path> 및 <Glyphs> 요소들(함께 모든 드로잉을 실행함)을 갖는 <FixedPage>, 및 함께 그룹화하기 위해 <Canvas> 요소의 페이지 콘텐츠를 표현한다.

[0455] 고정된 페이로드 마크업의 요소 계층구조는 "탑-레벨 요소", "Path, Clip을 위한 기하형태", "Path, Glyphs, 또는 OpacityMask를 채우는데 사용된 브러시들", "FixedPage 또는 Canvas를 위한 리소스 사전", "알파 투명성을 위한 불투명 마크(Opacity mask)", "클리핑 경로" 및 "변환"이라는 제목의 이하의 장들로 요약된다.

[0456] **탑-레벨 요소(Top-level elements)**

[0457] • <Document> [정확하게 FixedPanel 당 하나]

[0458] ◦ **속성**:

[0459] ■ [없음]

[0460] ◦ **자식 요소**:

[0461] ■ <FixedPanel>[정확히 하나]

[0462] • <FixedPanel>

[0463] ◦ **속성**:

[0464] ■ PageHeight[선택적]

[0465] ■ PageWidth[선택적]

[0466] ◦ **자식 요소**:

[0467] ■ <PageContent>[자식 요소들의 1-N]

[0468] • <PageContent>

[0469] ◦ **속성**:

[0470] ■ Source[필요함]

[0471] ■ PageHeight[선택적]

[0472] ■ PageWidth[선택적]

[0473] ◦ **자식 요소**:

[0474] ■ [없음]

[0475] • <FixedPage>

[0476] ◦ **간단한 XML 속성을 통해 직접 표현된 속성**:

[0477] ■ PageHeight[필요함(여기 또는 자식 요소로서)]

[0478] ■ PageWidth[필요함(여기 또는 자식 요소로서)]

[0479] ◦ **XML 자식 요소로서 표현된 리소스 사전 자체**:

[0480] ■ <FixedPage.Resources>

[0481] ○ **XML 자식 요소를 통해 표현된 속성**

[0482] ■ <FixedPage.PageHeight>[필요함(여기 또는 속성으로서)]

[0483] ■ <FixedPage.PageWidth>[필요함(여기 또는 속성으로서)]

[0484] ○ **XML 자식 요소를 통한 콘텐츠:**

[0485] ■ <Canvas>

[0486] ■ <Path>

[0487] ■ <Glyphs>

[0488] • <Canvas>

[0489] ○ **간단한 XML 속성을 통해 직접 표현되는 속성:**

[0490] ■ Opacity

[0491] ○ **리소스 사전 참조를 통해 표현되는 속성:**

[0492] ■ Clip

[0493] ■ RenderTransform

[0494] ■ OpacityMask

[0495] ○ **XML 자식 요소로서 표현된 리소스 사전 자체:**

[0496] ■ <Canvas.Resources>

[0497] ○ **XML 자식 요소를 통해 표현된 속성**

[0498] ■ <Canvas.Opacity>

[0499] ■ <Canvas.Clip>

[0500] ■ <Canvas.RenderTransform>

[0501] ■ <Canvas.OpacityMask>

[0502] ○ **XML 자식 요소를 통한 콘텐츠:**

[0503] ■ <Canvas>

[0504] ■ <Path>

[0505] ■ <Glyphs>

[0506] • <Path>

[0507] ○ **간단한 XML 속성을 통해 직접 표현되는 속성:**

[0508] ■ Opacity

[0509] ○ **리소스 사전 참조를 통해 표현되는 속성:**

[0510] ■ Clip

[0511] ■ RenderTransform

[0512] ■ OpacityMask

[0513] ■ Fill

[0514] ○ XML 자식 요소를 통해 표현되는 속성

[0515] ■ <Path.Opacity>

[0516] ■ <Path.Clip>

[0517] ■ <Path.RenderTransform>

[0518] ■ <Path.OpacityMask>

[0519] ■ <Path.Fill>

[0520] ■ <Path.Data>

[0521] • <Glyphs>

[0522] ○ 간단한 XML 속성을 통해 직접 표현되는 속성:

[0523] ■ Opacity

[0524] ■ BidiLevel

[0525] ■ FontFaceIndex

[0526] ■ FontHintingEmSize

[0527] ■ FontRenderingEmSize

[0528] ■ FontUri

[0529] ■ Indices

[0530] ■ OriginX

[0531] ■ OriginY

[0532] ■ Sideways

[0533] ■ StyleSimulations

[0534] ■ UnicodeString

[0535] ○ 리소스 사전 참조를 통해 표현되는 속성:

[0536] ■ Clip

[0537] ■ RenderTransform

[0538] ■ OpacityMask

[0539] ■ Fill

[0540] ○ XML 자식 요소를 통해 표현되는 속성

[0541] ■ <Glyphs.Clip>

[0542] ■ <Glyphs.RenderTransform>

[0543] ■ <Glyphs.OpacityMask>

[0544] ■ <Glyphs.Fill>

[0545] ■ <Glyphs.Opacity>

[0546] ■ <Glyphs.BidiLevel>

[0547] ■ <Glyphs.FontFaceIndex>

[0548] ■ <Glyphs.FontHintingEmSize>

[0549] ■ <Glyphs.FontRenderingEmSize>

[0550] ■ <Glyphs.FontUri>

[0551] ■ <Glyphs.Indices>

[0552] ■ <Glyphs.OriginX>

[0553] ■ <Glyphs.OriginY>

[0554] ■ <Glyphs.Sideways>

[0555] ■ <Glyphs.StyleSimulations>

[0556] ■ <Glyphs.UnicodeString>

[0557] **Path, Clip을 위한 기하형태**

[0558] • <Path.Data>

[0559] ○ **속성**:

[0560] ■ [없음]

[0561] ○ **단일 XML 자식 요소로서 표현되는 속성**:

[0562] [Path.Data는 상기 자식들 중 정확히 하나의 총 자식을 갖는다]

[0563] ■ <GeometryColection>

[0564] ■ <PathGeometry>

[0565] • <GeometryCollection>

[0566] ○ **속성**:

[0567] ■ CombineMode

[0568] ○ **자식 요소**:

[0569] [1-N 자식들]

[0570] ■ <GeometryCollection>

[0571] ■ <PathGeometry>

[0572] • <PathGeometry>

[0573] ○ **속성**:

[0574] ■ FillRule

[0575] ○ **자식 요소**:

[0576] [1-N 자식들]

[0577] ■ <PathFigure>

[0578] • <PathFigure>

[0579] ○ **속성**:

[0580] ■ [없음]

[0581] ○ **자식 요소**:

[0582] [StartSegment가 제일 먼저 오고, CloseSegment가 마지막으로 온다, 그들 간에는 Poly*의 1-N]

[0583] ■ <StartSegment>

[0584] ■ <PolyLineSegment>

[0585] ■ <PolyBezierSegment>

[0586] ■ <CloseSegment>

[0587] • <StartSegment>

[0588] ○ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0589] ■ Point

[0590] ○ **XML 자식 요소를 통해 표현되는 속성**

[0591] ■ <StartSegment.Point>

[0592] • <PolyLineSegment>

[0593] ○ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0594] ■ Points

[0595] ○ **XML 자식 요소를 통해 표현되는 속성**

[0596] ■ <PolyLineSegment.Points>

[0597] • <PolyBezierSegment>

[0598] ○ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0599] ■ Points

[0600] ○ **XML 자식 요소를 통해 표현되는 속성**

[0601] ■ <PolyBezierSegment.Points>

[0602] **Path, Glyphs, 또는 OpacityMask를 채우는데 사용되는 브러시들**

[0603] • <Path.Fill>

[0604] ○ **속성**:

[0605] ■ [없음]

[0606] ○ **단일 XML 자식 요소로서 표현되는 속성**:

[0607] [Path.Fill은 상기 자식들 중 정확히 하나의 자식을 갖는다]

[0608] ■ <SolidColorBrush>

[0609] ■ <ImageBrush>

[0610] ■ <DrawingBrush>

[0611] ■ <LinearGradientBrush>

[0612] ■ <RadialGradientBrush>

[0613] • <Glyphs.Fill>

[0614] ◦ **속성**:

[0615] ■ [없음]

[0616] ◦ **단일 XML 자식 요소로서 표현되는 속성**:

[0617] [Glyphs.Fill은 상기 자식들 중 정확히 하나의 자식을 갖는다]

[0618] ■ <SolidColorBrush>

[0619] ■ <ImageBrush>

[0620] ■ <DrawingBrush>

[0621] ■ <LinearGradientBrush>

[0622] ■ <RadialGradientBrush>

[0623] • <SolidColorBrush>

[0624] ◦ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0625] ■ Opacity

[0626] ■ Color

[0627] ◦ **XML 자식 요소를 통해 표현되는 속성**

[0628] ■ <SolidColorBrush.Opacity>

[0629] ■ <SolidColorBrush.Color>

[0630] • <ImageBrush>

[0631] ◦ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0632] ■ Opacity

[0633] ■ HorizontalAlignment

[0634] ■ VerticalAlignment

[0635] ■ ViewBox

[0636] ■ ViewPort

[0637] ■ Stretch

[0638] ■ TileMode

[0639] ■ ContentUnits

[0640] ■ ViewportUnits

[0641] ■ ImageSource

[0642] ◦ **리소스 사전 참조를 통해 표현되는 속성**:

[0643] ■ Transform

[0644] ◦ **XML 자식 요소를 통해 표현되는 속성**

[0645] ■ <ImageBrush.Opacity>

[0646] ■ <ImageBrush.Transform>

[0647] ■ <ImageBrush.HorizontalAlignment>

[0648] ■ <ImageBrush.VerticalAlignment>

[0649] ■ <ImageBrush.ViewBox>

[0650] ■ <ImageBrush.ViewPort>

[0651] ■ <ImageBrush.Stretch>

[0652] ■ <ImageBrush.TileMode>

[0653] ■ <ImageBrush.ContentUnits>

[0654] ■ <ImageBrush.ViewportUnits>

[0655] ■ <ImageBrush.ImageSource>

[0656] • <DrawingBrush>

[0657] ◦ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0658] ■ Opacity

[0659] ■ HorizontalAlignment

[0660] ■ VerticalAlignment

[0661] ■ ViewBox

[0662] ■ ViewPort

[0663] ■ Stretch

[0664] ■ TileMode

[0665] ■ ContentUnits

[0666] ■ ViewportUnits

[0667] ◦ **리소스 사전 참조를 통해 표현되는 속성**:

[0668] ■ Transform

[0669] ■ Drawing

[0670] ◦ **XML 자식 요소를 통해 표현되는 속성**

[0671] ■ <DrawingBrush.Opacity>

[0672] ■ <DrawingBrush.Transform>

[0673] ■ <DrawingBrush.HorizontalAlignment>

[0674] ■ <DrawingBrush.VerticalAlignment>

[0675] ■ <DrawingBrush.ViewBox>

[0676] ■ <DrawingBrush.ViewPort>

[0677] ■ <DrawingBrush.Stretch>

[0678] ■ <DrawingBrush.TileMode>

[0679] ■ <DrawingBrush.ContentUnits>

[0680] ■ <DrawingBrush.ViewportUnits>

[0681] ■ <DrawingBrush.Drawing>

[0682] • <DrawingBrush.Drawing>

[0683] ○ **XML 자식 요소를 통한 콘텐츠:**

[0684] ■ <Canvas>

[0685] ■ <Path>

[0686] ■ <Glyphs>

[0687] • <LinearGradientBrush>

[0688] ○ **간단한 XML 속성을 통해 직접 표현되는 속성:**

[0689] ■ Opacity

[0690] ■ MappingMode

[0691] ■ SpreadMethod

[0692] ■ StartPoint

[0693] ■ EndPoint

[0694] ○ **리소스 사전 참조를 통해 표현되는 속성:**

[0695] ■ Transform

[0696] ■ GradientStops

[0697] ○ **XML 자식 요소를 통해 표현되는 속성**

[0698] ■ <LinearGradientBrush.Opacity>

[0699] ■ <LinearGradientBrush.Transform>

[0700] ■ <LinearGradientBrush.MappingMode>

[0701] ■ <LinearGradientBrush.SpreadMethod>

[0702] ■ <LinearGradientBrush.StartPoint>

[0703] ■ <LinearGradientBrush.EndPoint>

[0704] ■ <LinearGradientBrush.GradientStops>

[0705] • <RadialGradientBrush>

[0706] ○ **간단한 XML 속성을 통해 직접 표현되는 속성:**

[0707] ■ Opacity

[0708] ■ Center

[0709] ■ Focus

[0710] ■ RadiusX

[0711] ■ RadiusY

[0712] ○ **리소스 사전 참조를 통해 표현되는 속성:**

[0713] ■ Transform

[0714] ■ GradientStops

[0715] ○ **XML 자식 요소를 통해 표현되는 속성**

[0716] ■ <RadielGradientBrush.Opacity>

[0717] ■ <RadielGradientBrush.Transform>

[0718] ■ <RadielGradientBrush.Center>

[0719] ■ <RadielGradientBrush.Focus>

[0720] ■ <RadielGradientBrush.RadiusX>

[0721] ■ <RadielGradientBrush.RadiusY>

[0722] ■ <RadielGradientBrush.GradientStops>

[0723] • <GradientStops>

[0724] ○ **XML 자식 요소를 통한 콘텐츠:**

[0725] ■ <GradientStop> [상기 자식들의 1-N]

[0726] • <GradientStop>

[0727] ○ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0728] ■ Color

[0729] ■ Offset

[0730] ○ **XML 자식 요소를 통해 표현되는 속성**

[0731] ■ <GradientStop.Color>

[0732] ■ <GradientStop.Offset>

[0733] **FixedPage 또는 Canvas를 위한 리소스 사전**

[0734] • <FixedPage.Resources>

[0735] • <Canvas.Resources>

[0736] 상기 요소들은 리소스 사전들을 논의하는 장에서 전술한 바와 같다.

[0737] **알파 투명성(alpha transparency)을 위한 불투명 마스크(opacity masks)**

[0738] • <Canvas.OpacityMask>

[0739] ○ **속성**:

[0740] ■ [없음]

[0741] ○ **단일 XML 자식 요소로서 표현되는 속성**:

[0742] [Canvas.OpacityMask는 이러한 자식들 중 정확히 하나의 자식을 갖는다]

[0743] ■ <SolidColorBrush>

[0744] ■ <ImageBrush>

[0745] ■ <DrawingBrush>

[0746] ■ <LinearGradientBrush>

[0747] ■ <RadialGradientBrush>

[0748] • <Path.OpacityMask>

[0749] ◦ **속성**:

[0750] ■ [없음]

[0751] ◦ **단일 XML 자식 요소로서 표현되는 속성**:

[0752] [Path.OpacityMask는 상기 자식들 중 정확히 하나의 자식을 갖는다]

[0753] ■ <SolidColorBrush>

[0754] ■ <ImageBrush>

[0755] ■ <DrawingBrush>

[0756] ■ <LinearGradientBrush>

[0757] ■ <RadialGradientBrush>

[0758] • <Glyphs.OpacityMask>

[0759] ◦ **속성**:

[0760] ■ [없음]

[0761] ◦ **단일 XML 자식 요소로서 표현되는속성**:

[0762] [Glyphs.OpacityMask는 상기 자식들 중 정확히 하나의 자식을 갖는다]

[0763] ■ <SolidColorBrush>

[0764] ■ <ImageBrush>

[0765] ■ <DrawingBrush>

[0766] ■ <LinearGradientBrush>

[0767] ■ <RadialGradientBrush>

[0768] **클리핑 경로(Clipping paths)**

[0769] • <Canvas.Clip>

[0770] ◦ **속성**:

[0771] ■ [없음]

[0772] ◦ **단일 XML 자식 요소로서 표현되는 속성**:

[0773] [Canvas.Clip은 상기 자식들 중 정확히 하나의 자식을 갖는다]

[0774] ■ <GeometryCollection>

[0775] ■ <PathGeometry>

[0776] • <Path.Clip>

[0777] ◦ **속성**:

[0778] ■ [없음]

[0779] ◦ **단일 XML 자식 요소로서 표현되는 속성**:

[0780] [Path.Clip은 상기 자식들 중 정확히 하나의 자식을 갖는다]

[0781] ■ <GeometryCollection>

[0782] ■ <PathGeometry>

[0783] • <Glyphs.Clip>

[0784] ◦ **속성**:

[0785] ■ [없음]

[0786] ◦ **단일 XML 자식 요소로서 표현되는 속성**:

[0787] [Glyphs.Clip은 상기 자식들 중 정확히 하나의 자식을 갖는다]

[0788] ■ <GeometryCollection>

[0789] ■ <PathGeometry>

[0790] **변환(Transforms)**

[0791] • <Canvas.RenderTransform>

[0792] ◦ **단일 XML 자식 요소로서 표현되는 속성**:

[0793] ■ <MatrixTransform> [필요함]

[0794] • <Path.RenderTransform>

[0795] ◦ **단일 XML 자식 요소로서 표현되는 속성**:

[0796] ■ <MatrixTransform> [필요함]

[0797] • <Glyphs.RenderTransform>

[0798] ◦ **단일 XML 자식 요소로서 표현되는 속성**:

[0799] ■ <MatrixTransform> [필요함]

[0800] • <MatrixTransform>

[0801] ◦ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0802] ■ Matrix

[0803] ◦ **XML 자식 요소를 통해 표현되는 속성**

[0804] ■ <MatrixTransform.Matrix>

[0805] • <ImageBrush.Transform>

[0806] ◦ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0807] ■ MatrixTransform

[0808] ◦ **XML 자식 요소를 통해 표현되는 속성**

[0809] ■ <ImageBrush.Transform.MatrixTransform>

[0810] • <DrawingBrush.Transform>

[0811] ◦ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0812] ■ MatrixTransform

[0813] ◦ **XML 자식 요소를 통해 표현되는 속성**

[0814] ■ <DrawingBrush.Transform.MatrixTransform>

[0815] • <LinearGradientBrush.Transform>

[0816] ◦ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0817] ■ MatrixTransform

[0818] ◦ **XML 자식 요소를 통해 표현되는 속성**

[0819] ■ <LinearGradientBrush.Transform.MatrixTransform>

[0820] • <RadialGradientBrush.Transform>

[0821] ◦ **간단한 XML 속성을 통해 직접 표현되는 속성**:

[0822] ■ MatrixTransform

[0823] ◦ **XML 자식 요소를 통해 표현되는 속성**

[0824] ■ <RadialGradientBrush.Transform.MatrixTransform>

[0825] FixedPage 마크업

[0826] 각각의 FixedPage 파트는 <FixedPage> 요소에 루팅된 XML 마크업으로 페이지의 콘텐츠를 표현한다. FixedPage 마크업은, 오직 요소 및 속성의 소집합(<Path> 및 <Glyphs> 요소들(함께 모든 드로잉을 함), 및 이들을 그룹화하기 위한 <Canvas> 요소)으로, 라이터와 리더 간의 다큐먼트의 WYSIWYG 충실도를 제공한다.

[0827] **공통 요소 속성(Common Element Properties)**

[0828] FixedPage 마크업의 각각의 요소에 특정한 속성들을 논의하기 전에, 드로잉 및 그룹화 요소들에게 공통인 속성들: Opacity, Clip, RenderTransform 및 OpacityMask를 고려해 본다. 상위 레벨 요소들에게 공통인 속성들일 뿐만 아니라, 구성 규칙(Composition Rules)의 장에 전술한 바와 같이, 부모로부터 자식 요소로의 결과들을 "누적(accumulate)"하는 속성들이다. 누적은 구성 규칙의 어플리케이션의 결과이다. 이하의 표는 이러한 공통 속성들의 요약 설명이며 속성들 각각에 대한 보다 철저한 설명을 제공한다.

[0829]

속성	요소	설명
Opacity	Canvas, Path, Glyphs, 및 SolidColorBrush, ImageBrush, DrawingBrush, LinearGradientBrush, RadialGradientBrush	요소의 균일한 투명성을 정의한다.

[0830]

자식 요소	요소	설명
Clip	Canvas, Path, Glyphs	Clip은 브러시가 캔버스에서 적용될 수 있는 영역을 한정한다.
RenderTransform	Canvas, Path, Glyphs	RenderTransform은 요소의 자식에 대한 새로운 좌표 프레임을 설정한다. MatrixTransform만이 지원됨.
OpacityMask	Canvas, Path, Glyphs	Opacity 속성과 동일한 방식으로 적용되는 알파 값들의 장방형 마크스를 지정하는데, 픽셀별로 상이한 알파 값을 허용한다.

[0831] Opacity 속성

[0832] Opacity는 렌더링할 때 두 요소들을 투명하게 혼합하는데 사용된다(알파 블렌딩). Opacity 속성은 0(완전 투명) 내지 1(완전 불투명)의 범위를 갖는다. 범위 밖의 값들은 마크업 파싱 중에 상기 범위로 클램프된다.

따라서, [-∞...0]은 투명하고 [1...∞]은 불투명하다.

[0833] Opacity 속성은 이하의 계산들을 통해 적용된다(미리 승산되지 않은(non-premultiplied) 소스 및 목적지 컬러들, 둘 다 scRGB로 지정되는 것을 가정함):

[0834] O_E: 요소의 Opacity 속성 또는 OpacityMask의 대응 위치에서의 알파 값

[0835] A_S: 소스 표면에 존재하는 알파 값

[0836] R_S: 소스 표면에 존재하는 레드 값

[0837] G_S: 소스 표면에 존재하는 그린 값

[0838] B_S: 소스 표면에 존재하는 블루 값

[0839] A_D: 목적지 표면(destination surface)에 이미 존재하는 알파 값

[0840] R_D: 목적지 표면에 이미 존재하는 레드 값

[0841] G_D: 목적지 표면에 이미 존재하는 그린 값

[0842] B_D: 목적지 표면에 이미 존재하는 블루 값

[0843] A^*: 목적지 표면에 대한 결과의 알파 값

[0844] R^*: 목적지 표면에 대한 결과의 레드 값

[0845] G^*: 목적지 표면에 대한 결과의 그린 값

[0846] B^*: 목적지 표면에 대한 결과의 블루 값

[0847] 아래첨자 T를 갖는 모든 값들은 임시 값들이다(예를 들면, R_{T1}).

[0848] **단계 1: 소스 알파 값을 opcity 값으로 승산한다**

[0849] $A_S = A_S * O_E$

[0850] **단계 2: 소스 알파를 먼저 승산한다**

[0851] $A_{T1} = A_S$

[0852] $R_{T1} = R_S * A_S$

[0853] $G_{T1} = G_S * A_S$

[0854] $B_{T1} = B_S * A_S$

[0855] **단계 3: 목적지 알파를 미리 승산한다**

[0856] $AT_2 = A_D$

[0857] $RT_2 = R_D * A_D$

[0858] $GT_2 = G_D * A_D$

[0859] $BT_2 = B_D * A_D$

[0860] **단계 3: 혼합한다(Blend)**

[0861] $AT_2 = (1-A_{T1}) * A_{T2} + A_{T1}$

[0862] $RT_2 = (1-A_{T1}) * R_{T2} + R_{T1}$

[0863] $GT_2 = (1-A_{T1}) * G_{T2} + G_{T1}$

[0864] $BT_2 = (1-A_{T1}) * B_{T2} + B_{T1}$

[0865] **단계 4: 선승산을 역으로 한다**

[0866] $AT_2 = 0$이면, 모든 $A^*R^*G^*B^*$를 0으로 설정하라.

[0867] 아니면:

[0868] $A^* = A_{T2}$

[0869] $R^* = R_{T2} / A_{T2}$

[0870] $G^* = G_{T2} / A_{T2}$

[0871] $B^* = B_{T2} / A_{T2}$

[0872] Clip 특성

[0873] Clip 특성은 기하학적 요소들 <GeometryCollection> 또는 <PathGeometry> 중 하나로서 지정된다(세부 사항을 위해서는 Path.Data를 참조).

[0874] Clip 속성은 이하의 방법으로 적용된다:

[0875] • Clip 자식 요소에 의해 기술된 기하학적 요소 내에 속한 모든 렌더링된 콘텐츠는 가시적(visible)이다.

[0876] • Clip 자식 요소에 의해 기술된 기하학 요소 밖에 속한 모든 렌더링된 콘텐츠는 비가시적이다.

[0877] RenderTransform 자식 요소

[0878] MatrixTransform은 오직 요소들에게 유용한 변환 속성이다. 이는 유사 변환을 표현한다. 구문(syntax)은 다음과 같다:

[0879]
```
<X.RenderTransform>
      <MatrixTransform Matrix="1,0,0,1,0,0"/>
</X.RenderTransform>
```

[0880] Matrix 속성에 지정된 6개의 번호들은 m00, m01, m10, m11, dx, dy이다.

[0881] 전체 행렬은 다음과 같다:

[0882] m00 m01 0

[0883] m10 m11 0

[0884] dx dy 1

[0885] 주어진 좌표 X, Y는 RenderTransform으로 변환되어서 상기 계산들을 적용해서 결과 좌표 X', Y'를 산출한다:

[0886] $X' = X * m00 + Y * m10 + dx$

[0887] $Y' = X * m01 + Y * m11 + dy$

[0888] OpacityMask 자식 요소

[0889] OpacityMask는 Fill Brush와 대조적으로 Brush를 지정하는데, 브러시의 알파 채널(상기 Opacity 속성을 참조하라)만이 요소를 렌더링하기 위한 추가 파라미터로서 사용된다. 요소의 각각의 픽셀에 대한 각각의 알파 값은 OpacityMask Brush의 대응 위치에서의 알파 값으로 추가로 승산된다.

[0890] **<Canvas> 요소**

[0891] <Canvas> 요소는 함께 요소들을 그룹화하는데 사용된다. 통상, FixedPage 요소들은 구성된 공통 속성(즉, Opacity, Clip, RenderTransform, OpacityMask)을 공유할 때 <Canvas>에서 함께 그룹화된다. 상기 요소들을 Canvas에서 그룹화함으로써, 공통 속성들이 종종 개별 요소들 대신 캔버스에 적용될 수 있다.

[0892] <Canvas>의 속성 및 자식 요소

[0893] <Canvas> 요소는 전술한 공통 속성들(Opacity, Clip, RenderTransform, OpacityMask)만을 갖는다. 이들은 이하의 표에 기술된 바와 같이 <Canvas> 요소와 함께 사용된다:

[0894]

속성	Canvas에 대한 효과
Opacity	요소의 균일한 투명성을 정의한다.

[0895]

자식 요소	Canvas에 대한 효과
Clip	Clip은 브러시가 Canvas의 자식 요소들에 의해 적용될 수 있는 영역을 기술한다.
RenderTransform	RenderTransform은 다른 캔버스와 같이, 캔버스의 자식 요소들에 대한 새로운 좌표 프레임을 설정한다. MatrixTransform만이 지원됨.
OpacityMask	Opacity 속성과 동일한 방식으로 적용되는 알파 값들의 장방형 마크스를 지정하는데, 픽셀별로 상이한 알파 값을 허용한다.

[0896] 이하의 마크업 일례는 <Canvas>의 사용을 나타낸다.

```
        <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>
```

[0897]

[0898] Canvas 마크업의 리딩 순서에 대하여, 이하를 고려해 본다. FixedPage에서와 같이, Canvas 내에 포함된 Glyphs 자식 요소들의 마크업 순서는 텍스트 콘텐츠의 희망하는 리딩 순서(reading order)와 동일해야만 한다. 이러한 리딩 순서는 뷰어에서 FixedPage로부터의 순차적인 텍스트의 인터랙티브 선택/복사, 및 액세스 가능성 기법(accessibility technology)에 의한 순차적인 텍스트로의 액세스를 가능하게 하는 것, 모두 사용될 수도 있다. FixedPage 마크업을 생성하는 어플리케이션은 마크업 순서 및 리딩 순서 간의 대응성을 보장할 책임이 있다.

[0899] 중첩 Canvas 요소들 내에 포함된 자식 Glyphs 요소들은 Canvas 전후에 발생하는 형제(sibling) Glyphs 요소들 사이에서 같은 라인에 순서화된다.

[0900] 예:

```
<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>
```

[0901]

[0902] **<Path> 요소**

[0903] Path 요소는 기하학적 영역을 기술하는 XML 기반의 요소이다. 기하학적 요소는 채워지거나, 클리핑 경로로서 사용될 수 있는 형태이다. 장방형 및 타원과 같은 공통 기하학적 타입들은 Path 기하형태를 사용하여 표현될

수 있다. 경로는 요구되는 Geometry.Data 자식 요소 및 Fill 또는 Opacity와 같은 렌더링 속성들을 지정함으로써 기술된다.

[0904] <Path>의 특성 및 자식 요소

[0905] 이하의 특성들은 후술하는 바와 같이 <Path> 요소들에 적용될 수 있다:

[0906]

속성	Path에 대한 효과
Opacity	채워진 경로의 균일한 투명성을 정의한다.

[0907]

자식 요소	Path에 대한 효과
Clip	Clip은 브러시가 경로의 기하형태에 의해 적용될 수 있는 영역을 기술한다.
RenderTransform	RenderTransform은 Path.Data에 의해 지정되는 기하형태와 같이, 경로의 자식 요소들에 대한 새로운 좌표 프레임을 설정한다. MatrixTransform만이 지원됨.
OpacityMask	Opacity 속성과 동일한 방식으로 적용되는 알파 값들의 장방형 마크스를 지정하는데, 표면의 상이한 영역들에 대한 상이한 알파 값을 허용한다.
Data	경로의 기하형태를 기술한다.
Fill	경로의 기하형태를 페인트하는데 사용되는 브러시를 기술한다.

[0908] <Path.Data> 자식 요소의 기하형태에 의해 기술되는 영역을 페인트하는 방법을 기술하기 위해, Fill 속성을 사용하라. <Path.Data> 형태들이 드로잉될 수 있는 영역을 제한하기 위해, Clip 속성을 사용하라.

[0909] 기하형태를 기술하기 위한 <Path>의 사용

[0910] 경로의 기하형태는 후술하는 바와 같이 일련의 <Path.Data>의 중첩 자식 요소들로 지정된다. 기하형태는 <PathGeometry> 자식 요소 집합을 포함하는 <GeometryCollection>, 또는 <PathFigures>를 포함하는 단일 <PathGeometry> 자식 요소 중 하나로 표현될 수 있다.

[0911]
```
<Path>
        <Path.Data>
                <GeometryCollection>
                        <PathGeometry>
                                <PathFigure>
                                        …
                                </PathFigure>
                        </PathGeometry>
                </GeometryCollection>
        </Path.Data>
<Path>
```

[0912] 동일한 <GeometryCollection> 또는 <PathGeometry> 요소들은 Canvas, Path, Glyphs의 Clip 속성에서 사용되는 클리핑 경로에 대한 기하형태를 정의한다.

[0913] 이하의 표는 경로 기하형태를 정의하는 자식 요소들의 계층구조를 소개한다.

[0914]

기하학 요소	설명
GeometryCollection	Boolean CombineMode 오퍼레이션을 사용해서 렌더링된 PathGeometry 요소들의 집합
PathGeometry	동일한 FillRule 옵션을 사용해서 각각 채워진 PathFigure 요소들의 집합
PathFigure	하나 이상의 세그먼트 요소들의 집합
StartSegment, PolyLineSegment, PolyBezierSegment, CloseSegment	

[0915] GeometryCollection

[0916] GeometryCollection은 Boolean CombineMode 옵션들에 따라 렌더링하기 위해 함께 결합된 기하학적 객체들의 집합이다. GeometryCollection 요소는 기하학적 형태의 시각적 결합을 구축하기 위한 FixedPage 마크업의 메카니

즘이다.

[0917]

속성	GeometryCollection에 대한 효과
CombineMode	기하학들을 결합하기 위한 상이한 모드들을 지정한다.

[0918] CombineMode 속성은 GeometryCollection의 기하학적 형태들의 집합을 결합하는데 사용되는 Boolean 오퍼레이션을 지정한다. 모드에 따라, 상이한 영역들은 포함 또는 배제된다.

[0919]

CombineMode 옵션	설명
Complement	새로운 영역으로부터 기존 영역이 제거된 결과로 기존 영역이 대체됨을 지정한다. 즉, 기존 영역은 새로운 영역으로부터 배제된다.
Exclude	기존 영역으로부터 새로운 영역이 제거되는 결과로 기존 영역이 대체됨을 지정한다. 즉, 새로운 영역은 기존 영역으로부터 배제된다.
Intersect	두 영역들이 교차를 취함으로써 결합된다.
Union	두 영역들이 둘 다의 합집합을 취함으로써 결합된다.
Xor	두 영역들이 둘 다에 포함되지 않고 일측 또는 타측 에만 포함된 영역을 취함으로써 결합된다.

[0920] CombineMode는 다음과 같이 처리된다:

[0921] **Not Communitative** Complement 및 Exclude는 상호적이지 않으며, 따라서, GeometryCollection의 제1 기하형태 및 각각의 나머지 개별 기하형태들 간에 정의된다. 예를 들어, 집합 {g1, g2, g3}의 경우, Exclude의 CombineMode는 ((g1 exclude g2) and (g1 exclude g3))로 적용된다.

[0922] **Communitative** Boolean operations Union, Xor, Intersect는 상호적이며, 따라서, 순서와는 무관하도록 기하형태에 적용된다.

[0923] PathGeometry

[0924] PathGeometry 요소는 PathFigure 요소들의 집합을 포함한다. PathFigure의 합집합은 PathGeometry의 내부를 정의한다.

[0925]

속성	GeometryCollection에 대한 효과
FillRule	포함된 영역을 기술하는 경로들을 채우기 위한 대안 알고리즘들을 지정한다.

[0926] FillRule 속성에 대해서는, 이하를 고려해 본다. PathGeometry의 채워진 영역은, 그 채워진 속성이 참으로 설정되는, 포함된 모든 PathFigure를 취하고, FillRule을 적용하여 봉입 영역(enclosed area)을 결정함으로써 정의된다. FillRule 옵션들은 Geometry에 포함된 Figure 요소들의 교차 영역들이 어떻게 결합해서 Geometry의 결과 영역을 형성하는지를 지정한다.

[0927] 본 실시예에 따르면, EvenOdd Fill 및 NonZero Fill 알고리즘들이 제공된다.

[0928] EvenOdd Fill 알고리즘은 그 점에서 임의의 방향으로 무한대로 향하는 방사선(ray)을 묘화하고, 형태 세그먼트가 방사선을 교차하는 곳들을 검사함으로써 캔버스상의 포인트의 "내부성(insideness)"을 결정한다. 제로 카운트로 시작해서, 세그먼트가 왼쪽에서 오른쪽으로 방사선을 교차할 때마다 1을 추가하고, 경로 세그먼트가 오른쪽엣 왼쪽으로 방사선을 교차할 때마다 1을 감산한다. 교차수를 카운트 한 후, 결과가 0이면, 포인트는 경로의 *외부(outside)*이다. 그렇지 않다면, *내부(inside)*이다.

[0929] NonZero Fill 알고리즘은 임의의 방향으로 무한대를 나타내는 방사선을 그리고 방사선이 교차하는 소정의 형태로부터 경로 세그먼트들의 수를 카운트함으로써 캔버스에서 포인트의 "내부성(insideness)"을 결정한다. 상기 수가 홀수이면, 포인트는 내부이고; 짝수이면, 포인트는 외부이다.

[0930] PathFigure

[0931] PathFigure 요소는 하나 이상의 라인 또는 커브 세그먼트들의 집합으로 구성된다. 세그먼트 요소들은

PathFigure의 형태를 정의한다. PathFigure는 항상 닫혀진 형태(closed shape)를 정의하여야 한다.

[0932]

속성	PathFigure에 대한 효과
FillRule	봉입 영역(enclosed)을 기술하는 경로들을 채우기 위한 다른 알고리즘들을 지정한다.

[0933] 그림(figure)은 추가된 최종 포인트로부터 각각의 라인 또는 커브 세그먼트가 계속되는 시작점을 필요로 한다. PathFigure 집합의 제1 세그먼트는 StartSegment이어야만 하고, CloseSegment는 최종 세그먼트이어야만 한다. StartSegment는 Point 속성을 갖는다. CloseSegment는 속성을 갖지 않는다.

[0934]

StartSegment 속성	설명
Point	라인 세그먼트의 위치(시작점)

[0935] Path.Data 기하형태의 Fixed-Payload 마크업

[0936] 다음은 Canvas에서의 Path를 묘화하고(drawing) 채우기(filling) 위한 마크업을 제공한다. 이하의 특정 일례에서, 사각 Path는 Canvas상에 묘화되어, 솔리드 그린 브러시(solid green brush)로 채워진다.

```
        <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>
```

[0937]

[0938] 이하의 마크업은 큐빅 Bezier 커브를 기술한다. 즉, PolyBezierSegment 외에, Fixed-Payload 마크업은 큐빅 Bezier 커브를 묘화하기 위한 PolyBezierSegment를 포함한다.

```
<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>
```

[0939]

[0940] **브러시(Brushes)**

[0941] 브러시는 <Path> 요소에 의해 정의된 기하학적 형태들의 내부를 페인트하고, <Glyphs> 요소로 렌더링된 문자 비트맵들을 채우는데 사용된다. 브러시는 또한 <Canvas.OpacityMask>, <Path.OpacityMask>, 및 <Glyphs.OpacityMask>의 알파-투명성(alphs-transparency) 마스크를 정의하는데 사용된다. FixedPage 마크업은 이하의 브러시들을 포함한다:

[0942]

브러시 타입	설명
SolidColorBrush	솔리드 컬러로 정의된 기하학적 영역을 채운다.
ImageBrush	이미지로 영역을 채운다.
DrawingBrush	벡터 드로잉으로 영역을 채운다.
LinearGradientBrush	리니어 그래디언트로 영역을 채운다.
RadialGradientBrush	래디얼 그래디언트로 영역을 채운다.

[0943] 모든 브러시들이 Opacity 속성을 갖더라도, 속성들은 브러시들에 걸쳐 변한다. ImageBrush 및 DrawingBrush는 틸트 기능(tilting capability)들을 공유한다. 두 그래디언트-채우기(gradient-fill) 브러시들은 공통적으로 속성들을 또한 갖는다.

[0944] 이하, 마크업의 브러시 자식 요소의 사용을 설명한다:

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

[0945]

[0946] 브러시의 공통 특성

[0947] 본 실시예에 따르면, 이하의 특성들은 보다 적은 옵션 자식 요소(optional child elements)들을 갖는 간단한 브러시 SolidColorBrush를 제외하고, 모든 브러시들에 적용될 수 있다.

[0948]

속성	브러시 타입	기술
Opacity	모든 브러시	

[0949]

자식 요소	브러시 타입	기술
Transform	SolidColorBrush를 제외한 모든 브러시	브러시의 좌표 스페이스에 적용된 MatrixTransform을 기술한다.

[0950] DrawingBrush 및 ImageBrush를 위한 공통 속성

[0951]

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

[0952] Horizontal Alignment 속성은 채우는 영역 내에서 브러시가 어떻게 수평으로 정렬되는지를 지정한다. Vertical Alignment 속성은 그 채우는 영역 내에서 브러시가 어떻게 수직으로 정렬되는지를 지정한다. ViewBox 속성은 (0,0,0,0)의 디폴트 값을 가지며, 언셋(unset)으로 해석된다. 언셋(unset)인 경우, 어떠한 조정도 발생하지 않으며 Stretch 속성이 무시된다. ViewBox는 콘텐츠에 대한 새로운 좌표 시스템을 지정한다. 즉, ViewPort의 범위 및 원점을 다시 정의한다. Stretch 속성은 상기 콘텐츠가 ViewPort에 매핑되는 방법을 지정하는 것을 돕는다. ViewBox 속성의 값은 화이트스페이스(whitespace) 및/또는 쉼표로 개별화된 4개의 "단위 없는(unitless)" 숫자들 <min-x>, <min-y>, <width> 및 <height>의 리스트이며, 타입이 Rect.이다. ViewBox rect는 경계를 짓는 박스(bounding box)로 매핑되는 사용자 스페이스에서 장방형을 지정한다. 이는 scaleX 및 scaleY 삽입과 동일하게 작용한다. Stretch 속성(옵션이 있는 경우)은 그래픽의 종횡비(aspect ratio)를 보존하기 위한 추가 컨트롤을 제공한다. 추가 변환이 지정된 효과를 달성하기 위해 소정의 요소의 모든 자손들에게 적용된다. Brush에 대한 변환이 있으면, "상술된" 매핑이 ViewBox에 적용된다.

[0953] Stretch 속성은 이하의 모드들: None, Fill, Uniform, UniformToFill을 갖는다.

[0954]

Stretch 속성 옵션	설명
None	디폴트. 원래 크기를 보존한다.
Fill	종횡비가 보존되지 않으며 콘텐츠가 설정된 경계를 채우도록 스케일된다.
Uniform	이미지가 설정된 경계에 알맞게 될 때까지 크기를 균일하게 스케일한다.
UniformToFill	설정된 경계를 채우기 위해 크기를 균일하게 스케일하고 필요하면 클립핑한다.

[0955] 심플 브러시들 및 그 속성들

[0956] Path.Brush 및 Canvas.Brush 자식 요소들은 SolidColorBrush, ImageBrush, DrawingBrush를 포함한다.

[0957] SolidColorBrush는 솔리드 컬러로 정의된 기하학적 영역들을 채운다. 컬러의 알파 컴포넌트가 있으면, Brush의 해당 불투명 속성과의 승산 방식(multiplicative way)으로 결합된다.

[0958]

속성	효과
Color	채워진 요소들에 대한 컬러를 지정한다.

[0959] 다음 일례는 SolidColorBrush를 위해 컬러 속성이 어떻게 표현되는지를 나타낸다.

[0960]
```
<Path>
      <Path.Fill>
            <SolidColorBrush Color="#00FFFF"/>
      </Path.Fill>
      ...
</Path>
```

[0961] ImageBrush는 이미지로 스페이스를 채우는데 사용될 수 있다. ImageBrush를 위한 마크업은 URI가 지정되도록 한다. 모든 다른 속성들이 디폴트 값들로 설정되면, 이미지는 영역의 경계 박스(bounding box)를 채우도록 신장(stretch)된다.

[0962]

속성	효과
ImageSource	이미지 리소스의 URI를 지정한다.

[0963] ImageSource 속성은 지원된 리치 이미지 포맷(Reach Image Format)들 중 하나 또는 상기 타입들 중 하나의 이미지를 가져오는 셀렉터 중 하나를 참조해야만 한다.

[0964] DrawingBrush는 벡터 드로잉으로 스페이스를 채우는데 사용될 수 있다. DrawingBrush는 Drawing 자식 요소를 갖는데, 마크업에서의 그 용도를 설명한다.

[0965]
```
<Path>
      <Path.Fill>
            <DrawingBrush>
                  <DrawingBrush.Drawing>
                        <Drawing>
                              <Path … />
                              <Glyphs … />
                        </Drawing>
                  </DrawingBrush.Drawing>
            </DrawingBrush>
      </Path.Fill>
</Path>
```

[0966] 그래디언트 브러시(Gradient Brushes) 및 그 속성들

[0967] 그래디언트는 그래디언트 브러시의 XML 자식 요소들로서 그래디언트 스톱들의 집합을 지정함으로써 묘화된다. 상기 그래디언트 스톱들은 몇몇 종류의 진행에 따라 컬러들을 지정한다. 본 프레임워크에서 지원되는 두 타입의 그래디언트 브러시들: 리니어(linear) 및 래디얼(radial)이 있다.

[0968] 그래디언트는 지정된 색공간에서 그래디언트 스톱들 간의 보간을 실행함으로써 그려진다. LinearGradientBrush 및 GradientBrush는 이하의 공통 속성들을 공유한다:

[0969]

속성	설명
SpreadMethod	본 속성은 주요 초기 그래디언트 영역 외부의 콘텐츠 영역을 브러시가 어떻게 채워야 하는지를 기술한다. 디폴트 값은 Pad이다.
MappingMode	본 속성은 그래디언트를 기술하는 파라미터들이 객체의 경계 박스와 관련해서 해석되는지를 결정한다. 디폴트 값은 relative-to-bounding-box이다.

[0970]

자식 요소	기술
GradientStops	GradientStop 요소들의 순서화된 시퀀스를 유지한다

[0971] SpreadMethod 속성에 대하여, 이하를 고려한다. SpreadMethod 옵션들은 스페이스가 어떻게 채워지는지를 지정한다. 디폴트 값은 Pad이다.

[0972]

SpreadMethod 속성 옵션	Gradient에 대한 영향
Pad	제1 컬러 및 최종 컬러는 각각 처음 및 끝에서 나머지 스페이스를 채우는데 사용된다.
Reflect	그래디언트 스톱들은 스페이스를 채우기 위해 반복해서 역 순서로 리플레이된다.
Repeat	그래디언트 스톱들은 스페이스가 채워질 때까지 순서대로 반복된다.

[0973] MappingMode 속성

[0974] LinearGradientBrush에 대해, 이하를 고려해 본다. LinearGradientBrush는 벡터과 함께 리니어 그래디언트 브러시(linear gradient brush)를 지정한다.

[0975]

속성	설명
EndPoint	리니어 그래디언트의 끝점. LinearGradientBrush는 StartPoint로부터 EndPoint까지 컬러들을 보간한다. StartPoint는 오프셋 0을 표현하고, EndPoint는 오프셋 1을 표현한다. 디폴트는 1,1이다.
StartPoint	리니어 그래디언트의 시작점.

[0976] 이하의 마크업 일례는 LinearGradientBrush의 사용을 나타낸다. 장방형 경로를 갖는 페이지는 리니어 그래디언트로 채워진다:

```
<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>
```

[0977]

[0978] 본 예는 리니어 그래디언트로 채워진 장방형 경로를 갖는 페이지를 도시한다. Path는 또한 이를 클립핑하는 8각형 형태의 클립 특성을 갖는다.

```
<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>
```

[0979]

[0980] RadialGradient는 리니어 그래디언트에 대한 프로그래밍 모델과 유사하다. 그러나, 리니어 그래디언트가 그래디언트 벡터를 정의하기 위해 시작점 및 끝점을 갖는 반면, 래디얼 그래디언트는 그래디언트 동작을 정의하기 위해 초점(focal point)과 함께 원을 갖는다. 원은 그래디언트의 끝점을 정의한다 - 다시 말해서, 1.0의 그래디언트 스톱은 원의 원주에서의 컬러를 정의한다. 촛점은 그래디언트의 중심을 정의한다. 0.0의 그래디언트 스톱은 초점에서의 컬러를 정의한다.

[0981]

속성	설명
Center	래디얼 그래디언트의 중심점. RadialGradientBrush는 Focus로부터 타원 원주까지 컬러들을 보간한다. 원주는 Center 및 반경에 의해 결정된다. 디폴트는 0.5,0.5이다.
Focus	래디얼 그래디언트의 초점.
RadiusX	래디얼 그래디언트를 정의하는 타원의 X 차원의 반경. 디폴트는 0.5이다.
RadiusY	래디얼 그래디언트를 정의하는 타원의 Y 차원의 반경. 디폴트는 0.5이다.
FillGradient	Pad, Reflect, Repeat

[0982] 알파 및 투명성

[0983] 본 실시예에 따르면, 각각의 요소의 각각의 픽셀은 0.0(완전 투명) 내지 1.0(완전 불투명) 범위의 알파값을 운반한다. 알파값은 투명성의 시각적 효과를 달성하기 위해 요소들을 혼합할 때 사용된다.

[0984] 각각의 요소는 요소의 각각의 픽셀의 알파값이 균일하게 승산되게 하는 Opacity 속성을 가질 수 있다.

[0985] 또한, OpacityMask는 렌더링된 콘텐츠가 목적지로 어떻게 혼합되는지를 제어하게 되는 픽셀당 불투명도의 명세(specification)를 허용한다. OpacityMask에 의해 지정된 불투명도는 콘텐츠의 알파 채널에 이미 존재하도록 발생할 수도 있는 임의의 불투명도와의 승산으로 결합된다. OpacityMask에 의해 지정된 픽셀당 불투명도는 마스크의 각각의 픽셀의 알파 채널을 조사함으로써 결정된다 - 컬러 데이터는 무시된다.

[0986] OpacityMask의 타입은 Brush이다. 이는 Brush의 콘텐츠가 다수의 상이한 방법들로 콘텐츠의 범위에 매핑되는 방법에 대한 명세를 허용한다. 기하형태를 채우기 위해 사용되는 경우와 같이, Brush들은 전체 콘텐츠 스페이스를 채우고, 적합하게 콘텐츠를 신장시키거나 복제하는데 디폴트한다. 이는 ImageBrush가 콘텐츠를 완전히 커

버하기 위해 ImageSource를 신장시키고, GradientBrush가 에지로부터 에지까지 확장함을 의미한다.

[0987] 알파 블렌딩을 위해 필요한 계산들은 전술한 "Opacity 속성"의 장에서 설명하였다.

[0988] 이하의 예는 OpacityMask가 Glyphs 요소에 대한 "페이드 효과(fade effect)"를 생성하는데 사용된다. 본 일례의 OpacityMask는 불투명 블랙으로부터 투명 블랙으로 페이드하는 리니어 그래디언트이다.

```
//    /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>
```

[0989]

[0990] **리치 다큐먼트(Reach Document)의 이미지**

[0991] *FixedPage*에서, 이미지는 포함된 영역들을 채운다. *FixedPage*에 이미지를 배치하기 위해, 먼저 페이지에서 영역이 지정되어야 한다. 영역은 Path 요소의 기하형태에 의해 정의된다.

[0992] Path 요소의 Fill 특성은 기술된 영역에 대한 채우기(fill) 콘텐츠를 지정한다. 이미지는 ImageBrush에 의해 영역에 그려지는 채우기의 한 타입이다. 모든 브러시들은 브러시 콘텐츠를 적합하게 신장 또는 반복(틸팅: tilting)함으로써 전체 영역을 채우게 되는 디폴트 동작을 갖는다. ImageBrush의 경우, ImageSource 속성에 의해 지정된 콘텐츠는 영역을 완전히 커버하도록 신장된다.

[0993] 이하의 마크업은 이미지를 Canvas에 배치하는 방법을 설명한다.

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

[0994]

[0995] 장방형(rectangular) Path 요소를 포함해서, 다수의 이미지들이 장방형이기에, 리소스 사전은 마크업을 간단히 하는데 유용할 수도 있다. 다음, Path는 RenderTransform 속성(전술하였음)을 사용해서 위치될 수도 있다.

```
<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>
```

[0996]

[0997] **컬러**

[0998] 컬러는 scRGB 또는 sRGB 표기법을 사용해서 예를 들어 설명되는 마크업으로 지정될 수 있다. scRGB 명세는 "IEC 61966-2-2 scRGB"로서 공지되어 있으며 www.iec.ch로부터 획득될 수 있다.

[0999] 이하의 표는 ARGB 파라미터들을 나타낸다.

[1000]

네임	설명
R	현 컬러의 레드 scRGB 컴포넌트
G	현 컬러의 그린 scRGB 컴포넌트
B	현 컬러의 블루 scRGB 컴포넌트
A	현 컬러의 알파 scRGB 컴포넌트

[1001] 컬러 매핑(Color Mapping)

[1002] 현재, 컬러 문맥(color context)을 지정하는 메타데이터로 컬러화된 요소들에 태그를 다는 것이 고려중이다. 상기 메타데이터는 ICC 컬러 프로파일, 또는 다른 컬러 정의 데이터를 포함할 수 있다.

[1003] **<Glyphs> 요소**

[1004] 텍스트는 Glyphs 요소를 사용해서 고정된 페이로드에서 표현된다. 요소는 인쇄 및 리치 다큐먼트에 대한 요구사항들을 만족시키도록 설계된다.

[1005] Glyphs 요소는 이하의 속성들의 결합을 가질 수도 있다.

[1006]

속성	목적	마크업 표현(Glyphs 요소)
Origin	런에 있어서 제1 글리프의 원점. 글리프는 어드밴스 벡터의 리딩 에지와 베이스라인이 상기 점을 교차하도록 배치된다.	OriginX 및 OriginY 특성들에 의해 지정됨
FontRenderingEmsize	표면 단위를 묘화하는데 있어서의 폰트 크기(디폴트 1/96 인치)	Length 길이 단위로 측정됨
FontHintingEmsize	포인트들의 힌트 크기. 폰트들은 보다 굵은 세로줄 및 보다 작은 크기들의 보다 개방된 보울들과 같이 상이한 크기들의 미묘한 차이들을 생성하기 위해, 퓨어 스케일링이 할 수 있는 바와 동일한 스타일처럼 보이는 결과들을 야기하도록 힌팅을 포함할 수도 있다. 이는 자동으로 처리되는 디바이스 픽셀 해상도를 위한 힌팅과 동일하지 않다. 날짜(2003년 3월)에 대해 어떠한 공지된 폰트들도 크기 힌팅을 포함하지 않는다. 디폴트 값은 -12pts이다.	폰트의 포인트 크기를 표현하는 더블로 측정됨.
GlyphIndices	상기 런(run)을 표현하는 16 비트 글리프 숫자들의 어레이	인덱스 특성 파트. 표현을 위해서 후술된 바를 참조하라.
AdvanceWidths	GlyphIndices의 각각의 글리프에 대해 하나씩, 어드밴스 폭들의 어레이. 런중 (n>0)의 제n 글리프의 공칭 원점은 제n-1 글리프의 공칭 원점 + 런 어드밴스 벡터에 따라 추가된 제n-1어드밴스 폭이다. 베이스 글리프들은 일반적으로 0이 아닌 어드밴스 폭을 가지며, 글리프들을 결합해서 일반적으로 제로 어드밴스 폭을 갖는다.	인덱스 특성 파트. 표현을 위해서 후술된 바를 참조하라.
GlyphOffsets	글리프 오프셋 어레이. 글리프에 대한 최종 원점을 생성하기 위해 위에서 계산된 공칭 글리프 원점에 추가된다. 베이스 글리프들은 일반적으로 (0,0)의 글리프 오프셋을 가지며, 글리프들을 결합해서 일반적으로 가장 가까운 우선 베이스 글리프의 상부에 정확하게 배치하는 오프셋을 갖는다.	인덱스 속성 파트. 표현을 위해서 후술된 바를 참조하라.
GlyphTypeface	런중 모든 글리프들이 그려지는 물리적 폰트.	FontUri, FontFaceIndex 및 StyleSimulations 속성들

UnicodeString	선택적* 상기 글리프 런에 의해 표현된 문자들의 어레이. *Win32 프린터 드라이버들로부터 생성된 GlyphRuns의 경우, Win32 ExtTextOut(ETO_GLYPHINDEX)에 의해 원래 인쇄된 텍스트는 Unicode 코드포인트들 없이 글리프 인덱스들과 함께 드라이버에 전달된다. 이러한 경우, 생성된 Glyphs 마크업, 및 구성된 GlyphRun 객체는 코드포인트를 생략한다. 코드포인트 없이, 고정 포맷 뷰어에서 자르기 및 붙이기 또는 탐색과 같은 기능은 유용하지 않으며, 텍스트 디스플레이는 가능하다.	예
ClusterMap	UnicodeString에서 캐릭터 당 하나의 엔트리. 각각의 값은 UnicodeString의 대응 캐릭터를 표현하는 GlyphIndices의 제1 글리프의 오프셋을 제공한다. 다수의 캐릭터들이 단일 글리프에 매핑하는 경우, 또는 단일 캐릭터가 다수의 글리프들에 매핑하는 경우, 또는 다수의 캐릭터들이 다수의 글리프들에 나눌 수 없게 매핑하는 경우, 캐릭터 또는 캐릭터(들) 및 글리프 또는 글리프(들)은 클러스터라고 한다. ClusterMap의 모든 엔트리들은 클러스터의 제1 글리프의 GlyphIndices 어레이의 오프셋에 매핑한다.	인덱스 속성 파트. 표현을 위해서 후술된 바를 참조하라.
Sideways	글리프들이 측면에서 레이아웃된다. 디폴트에 의해, 글리프들은 수평 텍스트에 있는 것처럼 렌더링되며, 원점은 웨스턴 베이스라인 원점에 대응한다. 사이드웨이 플래그가 설정되면, 글리프는 측면에서 회전되고, 원점은 회전되지 않은 글리프의 상부 중심이다.	예
BidiLevel	Unicode 알고리즘 bidi 중첩 레벨. 숫자적으로 짝수인 값들은 좌우 레이아웃을 의미하고, 숫자적으로 홀수인 값들은 우좌 레이아웃을 의미한다. 우좌 레이아웃은 제1 글리프의 오른측에 런 원점을 배치하고, 어드밴스 벡터의 포지티브 값들은 다음 글리프들을 이전 글리프의 좌측에 배치한다.	예
Brush	글리프들을 그리는데 사용되는 전경 브러시.	Shape Fill 속성으로부터 선택됨.
Language	런의 언어, 통상 마크업의 xml:lang 속성으로부터 온다.	xml:lang 속성에 의해 지정됨.

[1007] **텍스트 마크업의 개요**

[1008] 글리프 메트릭스(Glyph metrics)

[1009] 각각의 글리프는 다른 글리프들과 정렬되는 방법을 지정하는 메트릭스를 정의한다. 일 실시예에 따른 일례의 메트릭스는 도 12에 도시되어 있다.

[1010] 어드밴스 폭 및 결합 마크(Combining Marks)

[1011] 일반적으로, 폰트 내의 글리프들은 베이스 글리프들이거나 또는 베이스 글리프에 첨부될 수도 있는 결합 마크들이다. 베이스 글리프들은 통상 0이 아닌 어드밴스 폭, 및 0,0 오프셋 벡터를 갖는다. 결합 마크는 통상 0의 어드밴스 폭을 갖는다. 오프셋 벡터는 결합 마크의 위치를 조정하는데 사용될 수도 있으며, 따라서, 결합 마크들에 대해서 non 0,0 값을 가질 수도 있다.

[1012] 글리프 런의 각각의 글리프는 위치를 제어하는 3개의 값들을 갖는다. 값들은 원점, 어드밴스 폭 및 글리프 오프셋이며, 각각 후술한다:

[1013] • **원점:** 각각의 글리프는 공칭 원점을 갖도록 가정되며, 런의 제1 글리프에 있어서, 공칭 원점이 런의 원점이다.

[1014] • **어드밴스 폭:** 각각의 글리프의 어드밴스 폭은 상기 글리프 원점과 관련해서 다음 글리프의 원점을 제공한다. 어드밴스 벡터는 항상 런 진행 방향으로 그려진다.

[1015] • **글리프 오프셋(베이스 또는 마크):** 글리프 오프셋 벡터는 상기 글리프 위치를 공칭 원점과 관련해서 조정한다.

[1016] **문자, 글리프 및 클러스터 맵**

[1017] 클러스터 맵은 Unicode 코드포인트 당 하나의 엔트리를 포함한다. 엔트리의 값은 상기 코드포인트를 표현하는 GlyphIndices 어레이의 제1 글리프의 오프셋이다. 대안으로서, 코드포인트가 비분할형(indivisible) 문자 클러스터를 표현하는 코드포인트들의 그룹의 파트이면, GlyphIndices 어레이의 제1 글리프는 해당 클러스터를 표현하는 제1 글리프의 오프셋을 표현한다.

[1018] 클러스터 매핑

[1019] 클러스터 맵은 일대일, 다대일, 일대다, 또는 다대다인 코드포인트-글리프 매핑을 표현할 수 있다. 일대일 매핑은 각각의 코드포인트가 정확하게 하나의 글리프에 의해 표현되는 경우이며, 도 13의 클러스터 맵 엔트리들은 0, 1, 2,... 이다.

[1020] 다대일 매핑은 두 개 이상의 코드포인트들이 단일 글리프에 매핑하는 경우이다. 상기 코드포인트들에 대한 엔트리들은 글리프 인덱스 버퍼의 해당 글리프의 오프셋을 지정한다. 도 14의 일례에서, 'f' 및 'i' 캐릭터들은 다수의 세리프 폰트들의 공통 타이프세팅 관례에서와 같이 합자(ligature)에 의해 대체되었다.

[1021] 일대다 매핑의 경우, 도 15를 참조해서 후술하는 바를 고려해 본다. 'Sara Am'은 이전 베이스 캐릭터의 상부에 있는 파트(링(ring)), 및 베이스 캐릭터의 오른쪽에 있는 파트(후크(hook))를 포함한다. 태국어 텍스트가 마이크로-져스티파이(micro-justified)되는 경우, 후크는 베이스 문자로부터 멀리 있게 되며, 링이 베이스 캐릭터 상부에 있어서, 다수의 폰트들이 링 및 훅을 개별 글리프들로 인코드한다. 한 코드포인트가 두 개 이상의 글리프들에게 매핑될 때, 해당 코드포인트에 대한 *ClusterMap*의 값은 코드포인트를 표현하는 *GlyphIndices* 어레이의 제1 글리프를 참조한다.

[1022] 다대다 매핑에 대해, 도 16을 참조해서 후술하는 바를 고려해 본다. 몇몇 폰트들에서, 캐릭터 클러스터에 대한 코드포인트들의 비분할 그룹은 하나 보다 많은 글리프에 매핑한다. 예를 들어, 이는 힌두어 스크립트들을 지원하는 폰트들에 공통이다. 코드포인트들의 비분할 그룹이 하나 이상의 글리프들에 매핑할 때, 코드포인트들 각각에 대한 ClusterMap의 값은 코드포인트를 표현하는 GlyphIndices 어레이의 제1 글리프를 참조한다.

[1023] 이하의 일례는 타밀어 단어 □□□□의 유니코드 및 글리프 표현들을 나타낸다. 처음 두 개의 코드포인트들은 결합해서 3개의 글리프들을 생성한다.

[1024] 클러스터를 지정함(Specifying Clusters)

[1025] 클러스터 명세(specification)는 non 1:1 클러스터(매핑이 1 캐릭터:1 글리프 보다 더 복잡함)의 제1 글리프에 대한 글리프 명세에 우선한다.

[1026] 각각의 클러스터 명세는 이하의 형태를 갖는다:

[1027] (ClusterCodepointCount[:ClusterGlyphCount])

[1028]

클러스터 명세 파트	타입	목적	디폴트 값
ClusterCodepointCount	양수 정수	상기 클러스터를 형성하기 위해 결합하는 16 비트 유니코드 코드포인트들의 수	1
ClusterGlyphCount	양수 정수	상기 클러스터를 형성하기 위해 결합하는 16 비트 글리프 인덱스들의 수	1

[1029] **<Glyphs> 마크업**

[1030] Glyphs 요소는 URI, 페이스 인덱스 및 상술된 다른 속성들의 집합으로서 폰트를 지정한다. 예를 들어:

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

[1031]

[1032] 각각의 글리프 명세는 이하의 형태를 갖는다:

[1033] [GlyphIndex][,[Advance][,[uOffset][,[vOffset][,[Flags]]]]]

[1034] 글리프 명세의 각각의 파트는 임의선택적이다:

[1035]

글리프 명세 파트	목적	디폴트 값
GlyphIndex	렌더링 물리 폰트의 글리프 인덱스	내부 텍스트의 대응 유니코드 코드포인트의 폰트 문자 맵 표에 의해 정의됨
Advance	본 글리프의 원점과 관련해서 다음 글리프를 배치. sideways 및 BidiLevel 속성들에 의해 정의된 어드밴스 방향으로 측정됨. 폰트 em 크기의 1/100으로 측정됨. 어드밴스는 라운딩 에러가 누적되지 않도록 계산돼야만 한다. 상기 요구 사항을 달성하는 방법에 대해 후술된 바를 참조하라.	폰트 HMTX 또는 VMTX 폰트 메트릭 표들에 의해 정의됨.
uOffset, vOffset	본 글리프를 이동하기 위한 글리프 원점과 관련된 오프셋. 통상 베이스 캐릭터에 마크들을 첨부하는데 사용된다. 폰트 em 크기의 1/100으로 측정됨.	0,0
Flags	베이스 글리프 및 결합 마크를 구별한다.	10(베이스 글리프)

[1036] 라운딩 에러 누적 없이 어드밴스를 계산하는 것에 대해, 다음을 생각해 보라. 각각의 어드밴스 값은 다음 글리프의 정확하게 라운드되지 않은 원점 - 선행 글리프들의 계산된(즉, 라운드된) 어드밴스 폭들의 합으로서 계산돼야만 한다. 이러한 방법으로, 각각의 글리프는 정확한 위치의 em의 0.5% 내로 위치 지정된다.

[1037] 글리프 마크업 일례들

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

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

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

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


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

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


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

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

[1038]

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

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

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

<!-- 'ёжик в тумане' using pre-composed 'ё' -->

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

<!-- 'ёжик в тумане' using composition of 'е' and diaeresis -->

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


<!-- 'ёжик в тумане' Forced rendering right-to-left showing
combining mark in logical order -->

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

</Canvas>
```

[1039]

[1040] **글리프 마크업(Glyphs Markup) 크기의 최적화**

[1041] 글리프 인덱스들 및 어드밴스 폭들과 같은 마크업 세부사항들은 타겟화된 클라이언트가 신뢰성 있게 재생할 수 있으면 마크업에서 생략될 수 있다. 이하의 옵션들은 공통 사용 간단한 스크립트들의 극적인 최적화를 허용한다.

[1042] 글리프 인덱스 최적화 마크업

[1043] Unicode 스트링의 캐릭터들의 위치들과 글리프 스트링의 글리프들의 위치들 간에 일대일 매핑이 있으며, 글리프 인덱스가 폰트의 CMAP(캐릭터 매핑) 표의 값이고, Unicode 캐릭터가 모호하지 않은 구문들을 갖는 경우, 글리프 인덱스는 마크업에서 생략될 수도 있다.

[1044] 글리프 인덱스는 캐릭터 대 글리프 매핑이 다음과 같은 경우 마크업에서 제공돼야만 한다:

[1045] • 두 개 이상의 코드포인트들이 단일 글리프(합자)를 형성하는 것과 같이, 일대일이 아닌 경우, 또는

[1046] • 하나의 코드포인트가 다수의 글리프들을 생성하는 경우, 또는

[1047] • OpenType 기능의 어플리케이션을 통한 것과 같이, 임의의 다른 형태의 글리프 치환이 발생한 경우.

[1048] 글리프 인덱스는 렌더링 엔진이 폰트의 CMAP(문자 매핑) 표와 상이한 글리프를 치환할 수도 있는 경우 마크업에서 제공돼야만 한다. 글리프 인덱스는 희망하는 글리프 표현이 폰트의 CMAP 표에 있지 않은 경우 제공돼야만 한다.

[1049] 글리프 위치 최적화 마크업

[1050] 글리프 어드밴스 폭은 요구되는 어드밴스 폭이 정확하게 폰트의 HMTX(수평 메트릭스) 또는 VMTX(수직 메트릭스)

표들의 글리프에 대한 것일 때 마크업에서 생략될 수도 있다.

[1051] 글리프 수직 오프셋은 제로인 경우 마크업에서 생략될 수도 있다. 이는 베이스 캐릭터에 대해 거의 항상 참이며, 보다 간단한 스크립트들의 결합 마크에 대해서도 통상 참이다. 그러나, 이는 종종 아라비아어 및 힌두어와 같은 보다 복잡합 스크립트들의 결합 마크에 대해서는 거짓이다.

[1052] 글리프 플래그 최적화 마크업

[1053] 글리프 플래그는 정규 정판 우선 순위를 갖는 베이스 글리프들에 대해 생략될 수도 있다.

[1054] **결론**

[1055] 상술된 모듈러 콘텐츠 프레임워크 및 다큐먼트 포맷 방법 및 시스템은 다큐먼트 중심 콘텐츠를 구성, 패키지화, 배포 및 렌더링하기 위한 빌딩 블록 집합을 제공한다. 빌딩 블록은 소프트웨어 및 하드웨어 시스템들이 다큐먼트들을 신뢰성 있고 일관되게 생성, 교환 및 디스플레이할 수 있게 해주는 다큐먼트 포맷에 대한 플랫폼 독립 프레임워크를 정의한다. 본 *리치 패키지* 포맷은 리치 패키지의 콘텐츠가 광범위한 환경들에서 광범위한 시나리오들에 걸쳐 디바이스들 및 어플리케이션들 간의 최대 충실도로 디스플레이 또는 인쇄될 수 있는 방식으로 페이지가 매겨지거나 또는 미리 페이지가 매겨진 다큐먼트들을 저장하기 위한 포맷을 제공한다. 본 발명이 구조적 특징 및/또는 방법 단계들에 특정한 언어로 기술되었지만, 첨부된 청구항들에서 정의된 발명이 상술된 특정 특징들 또는 단계들로 제한되는 것이 아님을 알 것이다. 오히려, 특정 특징들 및 단계들은 청구된 발병을 구현하는 양호한 형태들로서 기술된다.

[1056] 본 발명에 따른 모듈러 콘텐츠 프레임워크 및 다큐먼트 포맷 방법 및 시스템이 제공된다. 프레임워크 및 포맷은 다큐먼트 중심 콘텐츠를 구성, 패키지화, 분산 및 렌더링하기 위한 빌딩 블록 집합을 정의한다. 빌딩 블록은 소프트웨어 및 하드웨어 시스템들이 다큐먼트들을 신뢰성 있고 일관되게 생성, 교환 및 디스플레이할 수 있게 해주는 다큐먼트 포맷에 대한 플랫폼 독립 프레임워크를 정의한다. 프레임워크 및 포맷은 유연하고 확장 가능한 방식으로 설계되어 왔다.

[1057] 일반 프레임워크 및 포맷 외에, 리치 패키지 포맷이라고 공지된 특정 포맷이 일반 프레임워크를 사용해서 정의된다. 리치 패키지 포맷은 페이지가 매겨진 다큐먼트들을 저장하기 위한 포맷이다. 리치 패키지의 콘텐츠는 광범위한 환경들에서 광범위한 시나리오들에 걸쳐 디바이스들 및 어플리케이션들 간의 최대 충실도로 디스플레이 또는 인쇄될 수 있다.

도면의 간단한 설명

[0006] 도 1은 일 실시예에 따른 일례의 프레임워크 및 포맷의 컴포넌트들의 블록도.

[0007] 도 2는 일 실시예에 따른 다수의 파트들을 포함하는 다큐먼트를 유지하는 일례의 패키지의 블록도.

[0008] 도 3은 일 실시예에 따라 패키지를 생성하는 일례의 라이터 및 패키지를 판독하는 리더를 도시한 블록도.

[0009] 도 4는 3개의 개별 페이지들을 함께 묶는 일례의 파트를 도시한 도면.

[0010] 도 5는 일 실시예에 따라 리포트의 영어 표현 및 불어 표현을 모두 포함하는 재무 보고서를 생성하도록 배열된 일례의 셀렉터 및 시퀀스들을 도시한 도면.

[0011] 도 6은 일 실시예에 따라 패키지에 대한 통신을 위해 함께 작업하는 라이터 및 리더의 일례들을 도시한 도면.

[0012] 도 7은 다큐먼트의 다수의 파트들을 인터리브하는 일례를 도시한 도면.

[0013] 도 8 및 도 9는 도 7에 도시된 다큐먼트의 다수의 파트들을 패키지화하는 상이한 일례들을 도시한 도면.

[0014] 도 10은 일 실시예에 따라 패키지에서 구성하거나 발견될 수 있는 유효 타입들의 파트들 각각 및 일례의 리치 패키지를 도시한 도면.

[0015] 도 11은 일 실시예에 따른 공통 언어 런타임 개념들의 XML로의 일례의 매핑을 도시한 도면.

[0016] 도 12는 일 실시예에 따른 업라이트 및 사이드웨이 글리프 메트릭스(upright and sideways glyph metrics)를 둘 다 도시한 도면.

[0017] 도 13은 일 실시예에 따른 일대일 클러스터 맵을 도시한 도면.

[0018] 도 14는 일 실시예에 따른 다대일 클러스터 맵을 도시한 도면.

[0019] 도 15는 일 실시예에 따른 일대다 클러스터 맵을 도시한 도면.

[0020] 도 16은 일 실시예에 따른 다대다 클러스터 맵을 도시한 도면.

[0021] <도면의 주요 부분에 대한 부호의 설명>

[0022] 100 : 프레임워크 및 포맷 102 : 프레임워크

[0023] 104 : 리치 패키지 포맷 200 : 패키지

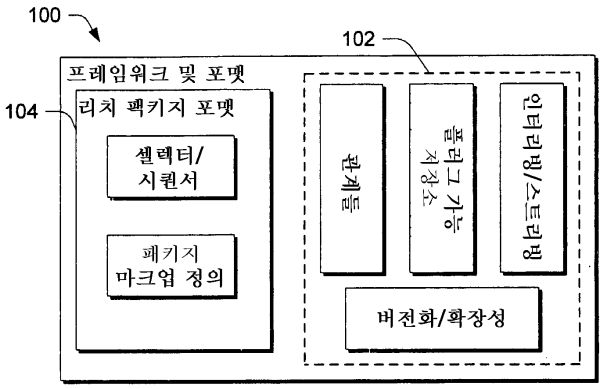
[0024] 202 : XML 마크업 파트 204 : 폰트 파트

[0025] 206 : 페이지 파트 500 : 셀렉터

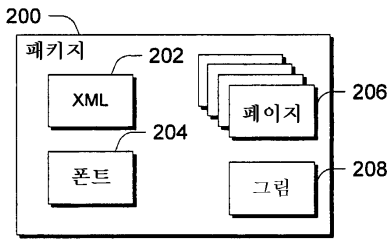
[0026] 502, 504 : 시퀀스

도면

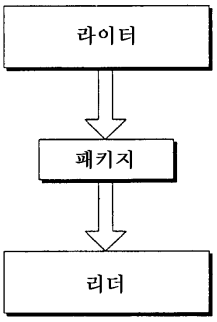
도면1



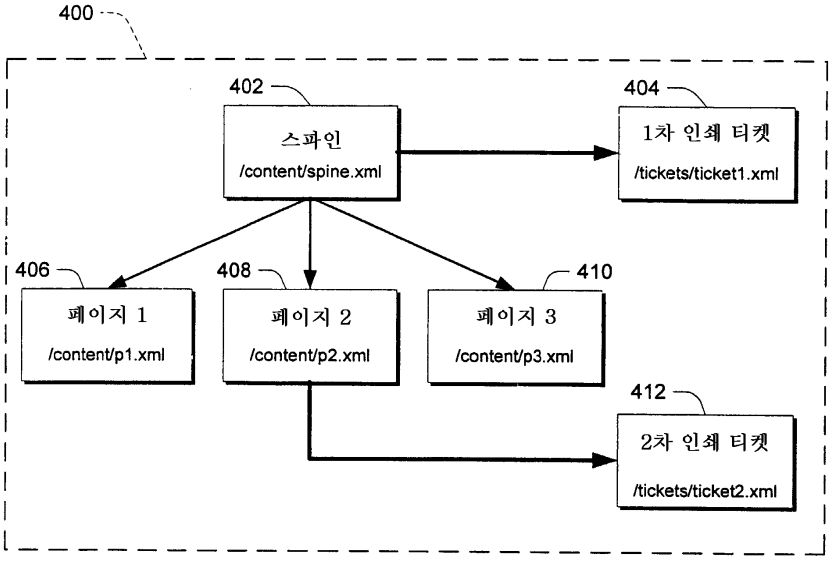
도면2



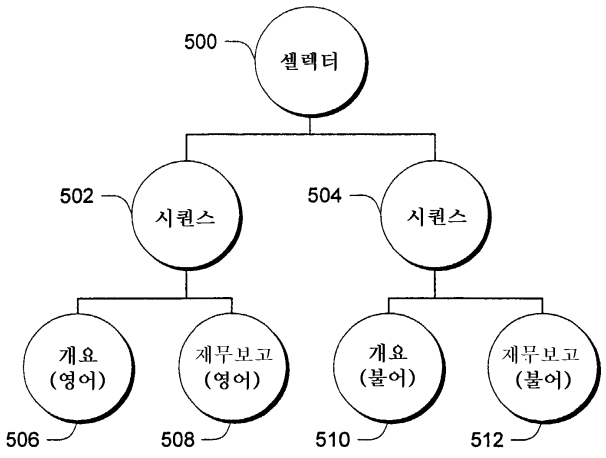
도면3


라이터
패키지
리더

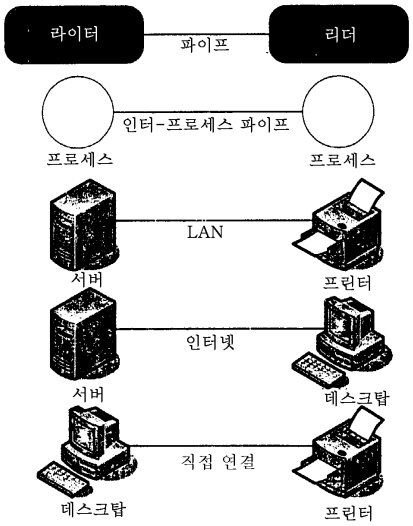
도면4


400
402
스파인
/content/spine.xml
404
1차 인쇄 티켓
/tickets/ticket1.xml
406
페이지 1
/content/p1.xml
408
페이지 2
/content/p2.xml
410
페이지 3
/content/p3.xml
412
2차 인쇄 티켓
/tickets/ticket2.xml

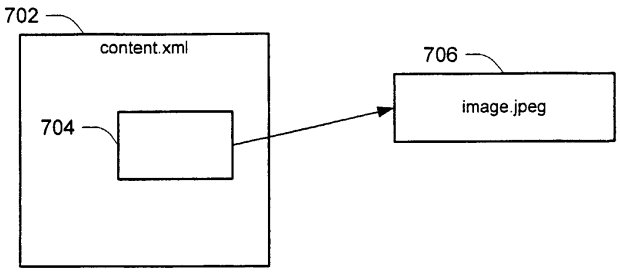
도면5


500
셀렉터
502
시퀀스
504
시퀀스
개요
(영어)
제무보고
(영어)
개요
(불어)
제무보고
(불어)
506
508
510
512

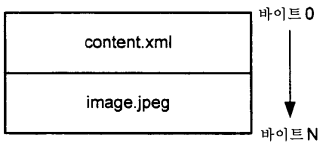
도면6



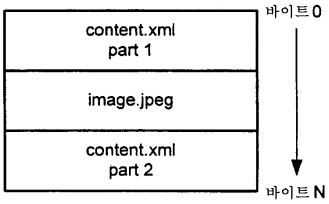
도면7



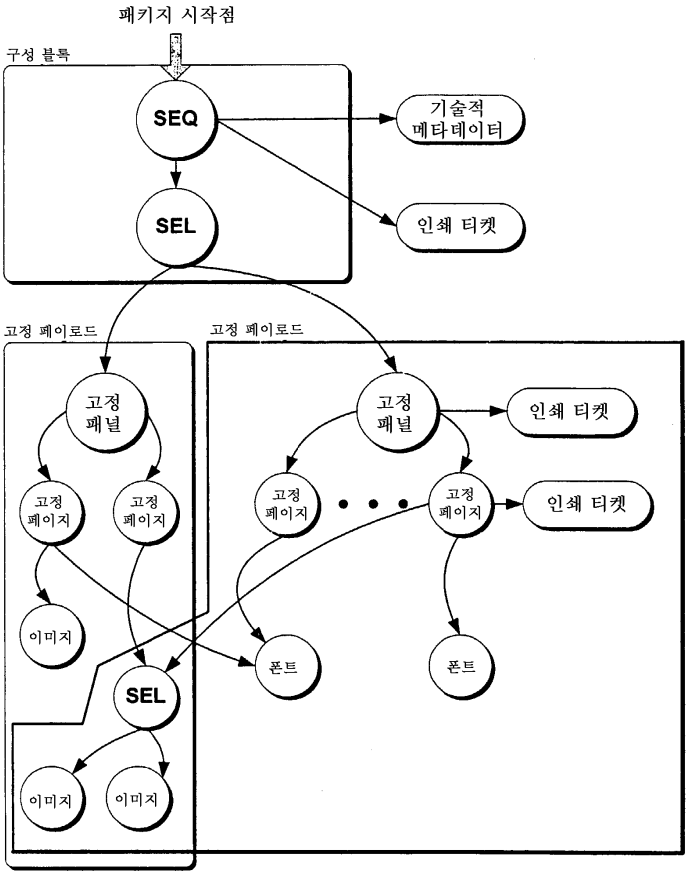
도면8



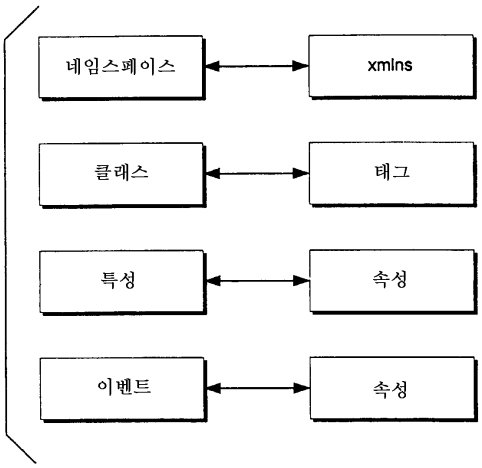
도면9



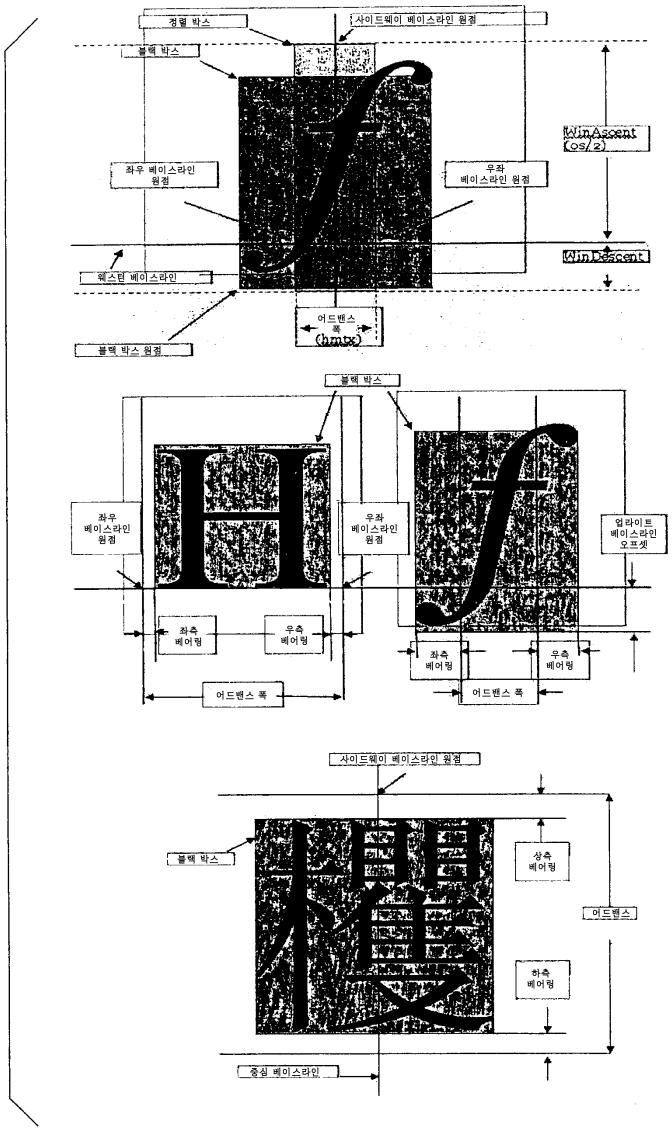
도면10


패키지 시작점
구성 블록
SEQ
SEL
기술적 메타데이터
인쇄 티켓
고정 페이로드
고정 페이로드
고정 패널
고정 페이지
고정 페이지
이미지
SEL
이미지
이미지
폰트
고정 패널
인쇄 티켓
고정 페이지
고정 페이지
인쇄 티켓
폰트

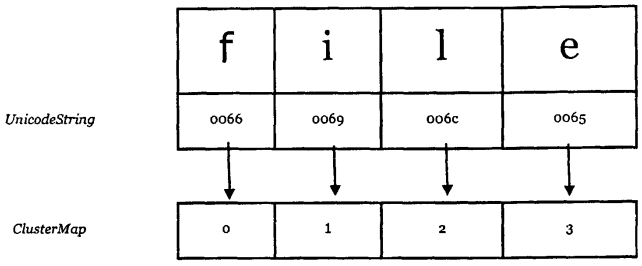
도면11


네임스페이스
xmlns
클래스
태그
특성
속성
이벤트
속성

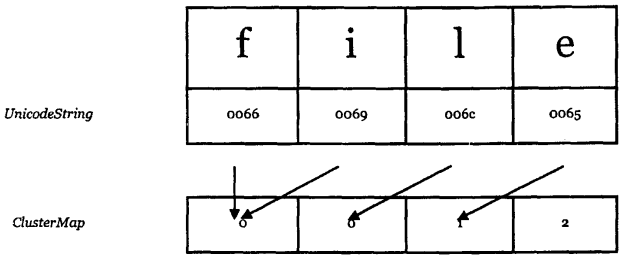
도면12


경계 박스
사이드웨이 베이스라인 원점
블랙 박스
WinAscent (os/2)
좌우 베이스라인 원점
우좌 베이스라인 원점
WinDescent
웨스턴 베이스라인
어드밴스 폭 (hmtx)
블랙 박스 원점
블랙 박스
좌우 베이스라인 원점
우좌 베이스라인 원점
업라이트 베이스라인 오프셋
좌측 베어링
우측 베어링
어드밴스 폭
좌측 베어링
우측 베어링
어드밴스 폭
사이드웨이 베이스라인 원점
블랙 박스
상측 베어링
어드밴스
하측 베어링
중심 베이스라인

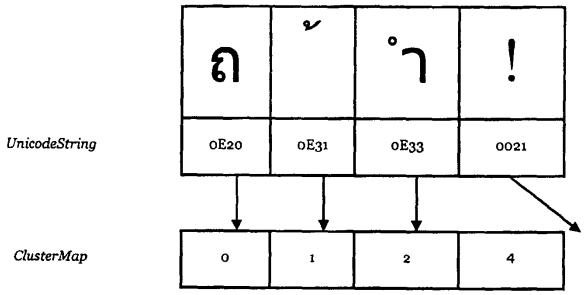
도면13


f
i
l
e
UnicodeString
0066
0069
006c
0065
ClusterMap
0
1
2
3

도면14



도면15



도면16