



Industria y Comercio

SUPERINTENDENCIA

05-59685

Delegatura de propiedad Industrial

División de Nuevas Creaciones

IE

SOLICITUD

PATENTE DE INVENCION

21 **EXPEDIENTE No**

05-59685

54 **TÍTULO**

Copie de integracion de Medios

51 **CLASIFICACIÓN INTERNACIONAL**

71 **SOLICITANTE**

Microsoft Corporation

DOMICILIO

Redmond Washington Estados Unidos de A

74 **APODERADO**

Dario Cardenas Novu

**APROVADO
FU 13-05-2005**

22 **BOGOTÁ, D. C.,**

PETITORIO



SUPERINDUSTRIA Y COMERCIO

Radicación : 85859685 00000000

Folios: 999

Fecha (AMD): 2005-06-28 12:41:42

Trámite : 011 PCT-PATENT 379 FASE NACI 411 PRESENTAC

Dependencia: 2026 DIVISION DE NUEVAS CREACIONES

FORMULARIO UNICO DE SOLICITUD DE PATENTE PCT ENTRADA EN FASE NACIONAL

①

SOLICITUD DE:

☒

Patente de Invención

☐

Patente de Modelo de Utilidad

☐

Capítulo I

☒

Capítulo II

②

SOLICITANTE
(71)

Nombre: MICROSOFT CORPORATION

Dirección: One Microsoft Way Redmond, Washington 98052, Estados Unidos de America

Nacionalidad o Domicilio: REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA.

Lugar de Constitución: Estado de Washington, Estados Unidos de América

Teléfono: 3123600

Fax: 3122410

E-mail: general@cardenasycardenas.com

IDENTIFICACION

C.C. ☐

NIT ☐

C.E. ☐

Otro ☒

Cual

Número

③

REPRESENTANTE
O APODERADO

Nombre: DARIO CARDENAS NAVAS

Dirección: Carrera 7 No 71-52 Torre B Piso 9

Teléfono: 3123600

Fax: 3122410

E-Mail: general@cardenasycardenas.com

IDENTIFICACION

C.C. ☒

NIT ☐

C.E. ☐

Otro ☐

Cual

No 17.066.629
BTA

TP 1.250
C.S.J.

④

INVENTOR(ES)
(72)

1. Nombre:

1. Sriram Subramanian
2. Leonardo E. Blanco
3. Donald B. Curtis
4. Joseph S. Bada
5. Gerhard A. Schneider
6. Greg D. Schechter
7. Adam M. Smith
8. Eric S. Vandenberg
9. Matthew W. Calkins
10. Kevin T. Gallo
11. Michael Stokes
12. Rajat Goel

2. Dirección:

1. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
2. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
3. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
4. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
5. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
6. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
7. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
8. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
9. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
10. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
11. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América
12. One Microsoft Way Redmond, Washington 98052, Estados Unidos de América

3. Nacionalidad
o Domicilio:

1. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
2. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
3. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
4. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
5. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
6. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
7. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
8. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
9. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
10. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
11. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA
12. REDMOND, WASHINGTON, ESTADOS UNIDOS DE AMERICA

5 PCT (86)

Solicitud Internacional No. PCT/US2004/024570 Fecha Julio 28, 2004
Publicación Internacional No. WO 2005/045584 Fecha 19-05-2005

6 Título(54): CAPA DE INTEGRACIÓN DE MEDIOS

7 Clasificación Internacional (51):

8 Prioridad

(33)País de Origen	(32)Fecha	(31)N° de Solicitud
<u>U.S.A.</u>	<u>Octubre 23, 2003</u>	<u>10/693,630</u>

SI ☒ NO ☐

Comprobante de Pago No.: 04-44,601

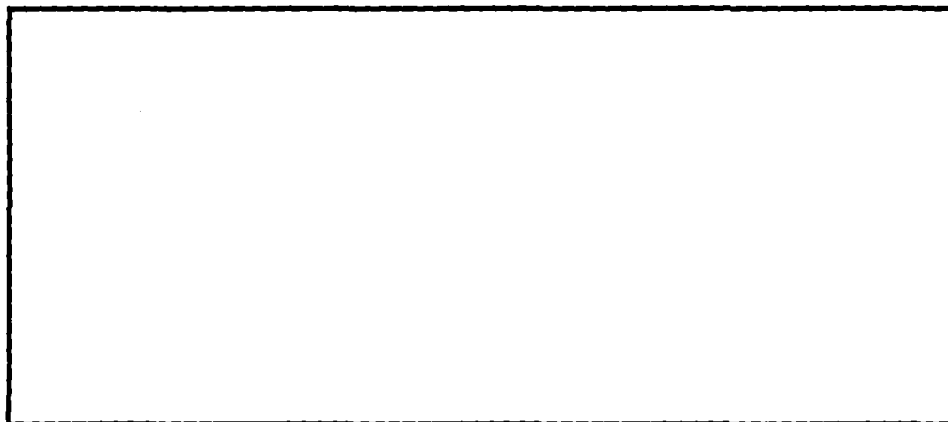
Fecha Junio 15, 2005

10

ANEXOS

- ☒ Comprobante de pago de la tasa de presentación de la solicitud.
- ☒ Documento que acredita la existencia y representación legal cuando el solicitante sea persona jurídica. (Protocolo 16.069)
- ☒ Poderes, si fuere el caso. (Protocolo 16.069)
- ☐ Documento de cesión del inventor al solicitante o a su causante.
- ☐ Declaraciones
- ☒ Copia de la solicitud internacional. (Resumen, descripción, reivindicaciones, dibujos)
- ☒ Traducción de la solicitud internacional al castellano. (Resumen, descripción, reivindicaciones, dibujos)
- ☐ Copia del examen preliminar internacional efectuado por la Autoridad Internacional de Examen Preliminar (IPEA).
- ☐ Traducción del examen preliminar internacional efectuado por la IPEA.
- ☐ De ser el caso, copia del contrato de acceso.
- ☐ De ser el caso, documento que acredite la licencia o autorización de uso de conocimientos tradicionales de las comunidades indígenas.
- ☐ De ser el caso, certificado de depósito del material biológico.
- ☒ Arte final 12x12.
- ☐ De ser el caso, información sobre otras solicitudes de patente o títulos obtenidos en el extranjero por el mismo titular o su causante, relacionadas parcial o totalmente con la invención de esta solicitud.
- ☒ De ser el caso, modificaciones efectuadas a la solicitud internacional.
- ☐ Otros.

11 FIGURA CARACTERISTICA



12

Solicito la Concesión de la patente.

NOMBRE: DARIO CARDENAS NAVAS

FIRMA:

OK C.C. 17.066.629 BTA TP. 1.250 un
C.S.J.

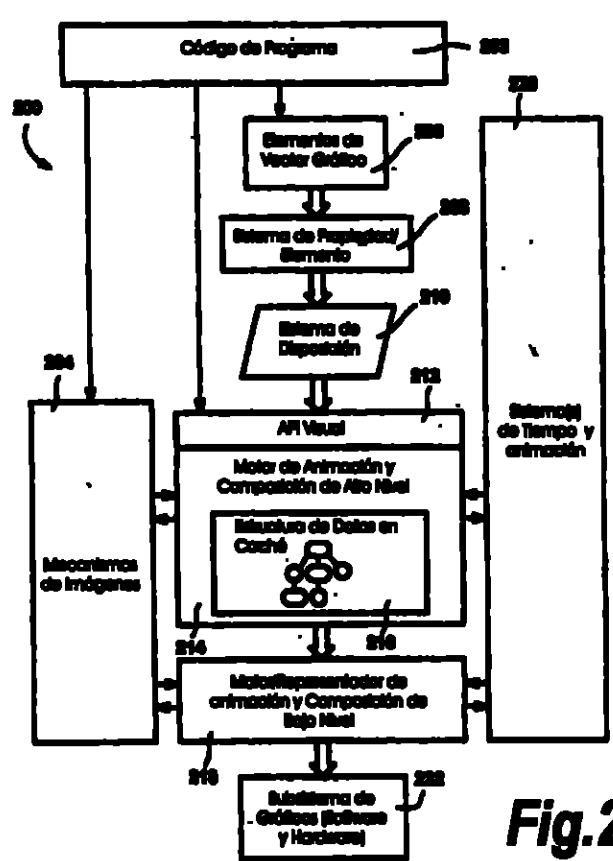


Fig.2

SUPERINTENDENCIA DE INDUSTRIA Y COMERCIO

NIT : 800176089-2

RECIBO OFICIAL DE CAJA : 05 - 44,601
FECHA : JUNIO 15 DE 2005

SUPERINTENDENCIA DE INDUSTRIA Y COMERCIO
Radicacion : 05053685 00000000 Folios: 999
Fecha (AMD): 2005-06-20 12:41:43
Tramite : 011 PCT-PATENT 379 FASE MACI 411 PRESENTAC
Dependencia: 2820 DIVISION DE NUEVAS CREACIONES

***** CONSIGNACION *****

DEPOSITANTE	TIPO PAGO	BANCO	CUENTA	No. PAGO	FECHA PGO	Vr. PAGO
CARDENAS Y CARDENAS P.I. CONSIGNACION		BANCO POPULAR	050-00110-6	28104303	15/06/2005	6,722,000.00
CARDENAS Y CARDENAS P.I. CONSIGNACION		BANCO POPULAR	050-00110-6	28104304	15/06/2005	1,010,000.00

***** CONCEPTO *****

CANT.	RENTISTICO	CONCEPTO	TOTAL CONCEPTO
1	50005-01-01 SOLICITUDES	1 TRAMITES DE SOL. DE PATENTE DE IN	443,000.00
		TOTAL :	\$ 443,000.00

SON: CUATROCIENTOS CUARENTA Y TRES MIL PESOS

RESPONSABLE : _____

RECIBO DE CAJA APLICADO AL EXPEDIENTE No. _____

CARDENAS & CARDENAS
ABOGADOS

8

PCT

REQUEST

The undersigned requests that the present international application be processed according to the Patent Cooperation Treaty.

For receiving Office use only

International Application No.

International Filing Date

Name of receiving Office and "PCT International Application"

Applicant's or agent's file reference
(if desired) (12 characters maximum) 4121

Box No. I TITLE OF INVENTION

Media Integration Layer

Box No. II APPLICANT

☐ This person is also inventor

Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Telephone No.
425-882-8080

Facsimile No.

Teleprinter No.

Applicant's registration No. with the Office

State (that is, country) of nationality:

US

State (that is, country) of residence:

US

This person is applicant
for the purposes of:all designated
Statesall designated States except
the United States of Americathe United States
of America onlythe States indicated in
the Supplemental Box

Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)

Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)

Subramanian, Sriram
6209 104th Avenue NE
Kirkland, Washington 98033
US

This person is:



applicant only



applicant and inventor

inventor only (If this check-box is
marked, do not fill in below.)

Applicant's registration No. with the Office

State (that is, country) of nationality:

US

State (that is, country) of residence:

US

This person is applicant
for the purposes of:all designated
Statesall designated States except
the United States of Americathe United States
of America onlythe States indicated in
the Supplemental Box☒ Further applicants and/or (further) inventors are indicated on a continuation sheet.

Box No. IV AGENT OR COMMON REPRESENTATIVE; OR ADDRESS FOR CORRESPONDENCE

The person identified below is hereby/has been appointed to act on behalf of the applicant(s) before the competent International Authorities as:



agent

common
representative

Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country.)

Michalik, Albert S.
704 - 228th Avenue NE, Suite 193
Sammamish, WA 98074
US

Telephone No.
425-836-3030Facsimile No.
425-836-3003

Teleprinter No.

Agent's registration No. with the Office
37,395

☐ Address for correspondence: Mark this check-box where no agent or common representative is/has been appointed and the space above is used instead to indicate a special address to which correspondence should be sent.

Continuation of Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)	
If none of the following sub-boxes is used, this sheet should not be included in the request.	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Blanco, Leonardo E. 9935 225th Avenue NE Redmond, Washington 98053 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: Uruguay	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Curtis, Donald B. 13533 SE 52nd Street Bellevue, Washington 98006 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: US	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Beda, Joseph S. 2437 East Lake Washington Blvd. Seattle, Washington 98112 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: Canada	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Schneider, Gerhard A. 8 West Aloha Street, # 304 Seattle, Washington 98119 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: Germany	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
☒ Further applicants and/or (further) inventors are indicated on another continuation sheet.	

A

Continuation of Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)	
<i>If none of the following sub-boxes is used, this sheet should not be included in the request.</i>	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Schechter, Greg D. 1127 37th Avenue East Seattle, Washington 98112 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: US	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Smith, Adam M. 12310 NE 92 Street #205 Kirkland, Washington 98033 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: US	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Vandenberg, Eric S. 1419 2nd Avenue W #201 Seattle, Washington 98119 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: Canada	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Calkins, Matthew W. 503 E Thomas Street #204 Seattle, Washington 98102 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: US	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
☒ Further applicants and/or (further) inventors are indicated on another continuation sheet.	

Continuation of Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)	
If none of the following sub-boxes is used, this sheet should not be included in the request.	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Gallo, Kevin T. 19235 222nd Way NE Woodinville, Washington 98072 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: US	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Stokes, Michael 1946 E. Stonybrook Court Eagle, Idaho 83616 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: US	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.) Goel, Rajat 121 Vine St. #2206 Seattle, Washington 98121 US	This person is: ☐ applicant only ☐ applicant and inventor ☒ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality: Canada	State (that is, country) of residence: US
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
Name and address: (Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)	This person is: ☐ applicant only ☐ applicant and inventor ☐ inventor only (If this check-box is marked, do not fill in below.) Applicant's registration No. with the Office
State (that is, country) of nationality:	State (that is, country) of residence:
This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box	
☐ Further applicants and/or (further) inventors are indicated on another continuation sheet.	

Box No. V DESIGNATIONS

The filing of this request constitutes under Rule 4.9(a), the designation of all Contracting States bound by the PCT on the international filing date, for the grant of every kind of protection available and, where applicable, for the grant of both regional and national patents.

However,

- ☐ DE Germany is not designated for any kind of national protection
- ☐ KR Republic of Korea is not designated for any kind of national protection
- ☐ RU Russian Federation is not designated for any kind of national protection

(The check-boxes above may be used to exclude (irrevocably) the designations concerned in order to avoid the ceasing of the effect, under the national law, of an earlier national application from which priority is claimed. See the Notes to Box No. V as to the consequences of such national law provisions in these and certain other States.)

Box No. VI PRIORITY CLAIM

The priority of the following earlier application(s) is hereby claimed:

Filing date of earlier application (day/month/year)	Number of earlier application	Where earlier application is:		
		national application: country or Member of WTO	regional application:* regional Office	international application: receiving Office
item (1) 23 October 2003	10/693,630	US		
item (2)				
item (3)				

☐ Further priority claims are indicated in the Supplemental Box.

The receiving Office is requested to prepare and transmit to the International Bureau a certified copy of the earlier application(s) (only if the earlier application was filed with the Office which for the purposes of this international application is the receiving Office) identified above as:

☒ all items ☐ item (1) ☐ item (2) ☐ item (3) ☐ other, see Supplemental Box

* Where the earlier application is an ARIPO application, indicate at least one country party to the Paris Convention for the Protection of Industrial Property or one Member of the World Trade Organization for which that earlier application was filed (Rule 4.10(b)(ii)):

Box No. VII INTERNATIONAL SEARCHING AUTHORITY

Choice of International Searching Authority (ISA) (if two or more International Searching Authorities are competent to carry out the international search, indicate the Authority chosen; the two-letter code may be used):

ISA / US

Request to use results of earlier search; reference to that search (if an earlier search has been carried out by or requested from the International Searching Authority):

Date (day/month/year)

Number

Country (or regional Office)

Box No. VIII DECLARATIONS

The following declarations are contained in Boxes Nos. VIII (i) to (v) (mark the applicable check-boxes below and indicate in the right column the number of each type of declaration):

Number of
declarations

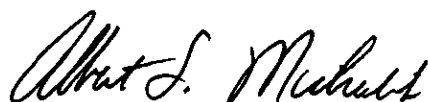
☐ Box No. VIII (i)	Declaration as to the identity of the inventor	:
☐ Box No. VIII (ii)	Declaration as to the applicant's entitlement, as at the international filing date, to apply for and be granted a patent	:
☐ Box No. VIII (iii)	Declaration as to the applicant's entitlement, as at the international filing date, to claim the priority of the earlier application	:
☐ Box No. VIII (iv)	Declaration of inventorship (only for the purposes of the designation of the United States of America)	:
☐ Box No. VIII (v)	Declaration as to non-prejudicial disclosures or exceptions to lack of novelty	:

Box No. IX CHECK LIST; LANGUAGE OF FILING

This international application contains:	This international application is accompanied by the following item(s) (mark the applicable check-boxes below and indicate in right column the number of each item):	Number of items
(a) in paper form , the following number of sheets:	1. ☒ fee calculation sheet	1
request (including declaration sheets) : 6	2. ☐ original separate power of attorney	
description (excluding sequence listing and/or tables related thereto) : 323	3. ☐ original general power of attorney	
claims : 9	4. ☒ copy of general power of attorney; reference number, if any:	1
abstract : 1	5. ☐ statement explaining lack of signature	
drawings : 29	6. ☐ priority document(s) identified in Box No. VI as item(s):	
Sub-total number of sheets : 368	7. ☐ translation of international application into (language):	
sequence listing :	8. ☐ separate indications concerning deposited microorganism or other biological material	
tables related thereto :	9. ☐ sequence listing in computer readable form (indicate type and number of carriers)	
(for both, actual number of sheets if filed in paper form, whether or not also filed in computer readable form; see (c) below)	(i) ☐ copy submitted for the purposes of international search under Rule 13ter only (and not as part of the international application) :	
Total number of sheets : 368	(ii) ☐ (only where check-box (b)(i) or (c)(i) is marked in left column) additional copies including, where applicable, the copy for the purposes of international search under Rule 13ter	
(b) ☐ only in computer readable form (Section 801(a)(i))	(iii) ☐ together with relevant statement as to the identity of the copy or copies with the sequence listing mentioned in left column	
(i) ☐ sequence listing	10. ☐ tables in computer readable form related to sequence listing (indicate type and number of carriers)	
(ii) ☐ tables related thereto	(i) ☐ copy submitted for the purposes of international search under Section 802(b-quater) only (and not as part of the international application)	
(c) ☐ also in computer readable form (Section 801(a)(ii))	(ii) ☐ (only where check-box (b)(ii) or (c)(ii) is marked in left column) additional copies including, where applicable, the copy for the purposes of international search under Section 802(b-quater)	
(i) ☐ sequence listing	(iii) ☐ together with relevant statement as to the identity of the copy or copies with the tables mentioned in left column	
(ii) ☐ tables related thereto	11. ☒ other (specify): Return Postcard	1
Type and number of carriers (diskette, CD-ROM, CD-R or other) on which are contained the		
☐ sequence listing:		
☐ tables related thereto:		
(additional copies to be indicated under items 9(ii) and/or 10(ii), in right column)		
Figure of the drawings which should accompany the abstract: 2	Language of filing of the international application: English	

Box No. X SIGNATURE OF APPLICANT, AGENT OR COMMON REPRESENTATIVE

Next to each signature, indicate the name of the person signing and the capacity in which the person signs (if such capacity is not obvious from reading the request).



Albert S. Michalik, Agent

For receiving Office use only	
1. Date of actual receipt of the purported international application:	2. Drawings: ☐ received: ☐ not received:
3. Corrected date of actual receipt due to later but timely received papers or drawings completing the purported international application:	
4. Date of timely receipt of the required corrections under PCT Article 11(2):	
5. International Searching Authority (if two or more are competent): ISA /	6. ☐ Transmittal of search copy delayed until search fee is paid

For International Bureau use only
Date of receipt of the record copy by the International Bureau:

11

PCT

FEE CALCULATION SHEET

Annex to the Request

For receiving Office use only

International Application No.

Applicant's or agent's
file reference

4121

Date stamp of the receiving Office

Applicant

Microsoft Corporation

CALCULATION OF PRESCRIBED FEES

1. TRANSMITTAL FEE \$300.00 T

2. SEARCH FEE \$300.00 S

International search to be carried out by
(If two or more International Searching Authorities are competent to carry out the international search, indicate the name of the Authority which is chosen to carry out the international search.)

3. INTERNATIONAL FILING FEE

Where items (b) and/or (c) of Box No. IX apply, enter Sub-total number of sheets } 368
Where items (b) and (c) of Box No. IX do not apply, enter Total number of sheets }

i1 first 30 sheets \$1134.00 i1

i2 338 x \$12.00 = \$4056.00 i2
number of sheets fee per sheet
in excess of 30

i3 additional component (only if sequence listing and/or tables related thereto are filed in computer readable form under Section 801(a)(i), or both in that form and on paper, under Section 801(a)(ii)):

400 x = i3
fee per sheet

Add amounts entered at i1, i2 and i3 and enter total at I I

(Applicants from certain States are entitled to a reduction of 75% of the international filing fee. Where the applicant is (or all applicants are) so entitled, the total to be entered at I is 25% of the international filing fee.)

4. FEE FOR PRIORITY DOCUMENT (if applicable) \$20.00 P

5. TOTAL FEES PAYABLE \$5,810.00

Add amounts entered at T, S, I and P, and enter total in the TOTAL box

TOTAL

MODE OF PAYMENT

- ☐ authorization to charge deposit account (see below) ☐ postal money order ☐ cash ☐ coupons
☐ cheque ☐ bank draft ☐ revenue stamps ☒ other (specify): Credit Card Pay

AUTHORIZATION TO CHARGE (OR CREDIT) DEPOSIT ACCOUNT

(This mode of payment may not be available at all receiving Offices)

- ☐ Authorization to charge the total fees indicated above.
☒ (This check-box may be marked only if the conditions for deposit accounts of the receiving Office so permit) Authorization to charge any deficiency or credit any overpayment in the total fees indicated above.
☐ Authorization to charge the fee for priority document.

Receiving Office: RO/ US

Deposit Account No.: 50-1618

Date: July 28, 2004

Name: Albert S. Michalik

Signature: Albert S. Michalik

72

12

PCT

GENERAL POWER OF ATTORNEY

(for several international applications filed under the Patent Cooperation Treaty)

(PCT Rule 90.5)

The undersigned person(s):

(Family name followed by given name: for a legal entity, full official designation. The address must include postal code and name of country.)

Microsoft Corporation
One Microsoft Way
Redmond, Washington, USA 98052-6399

hereby appoint(s) the following person as:

☒ agent

☐ common representative

Name and address

(Family name followed by given name: for a legal entity, full official designation. The address must include postal code and name of country.)

MICHALIK, Albert S. (Reg. No. 37,395); WYLIE, Roger D. (Reg. No. 36,974);
WHITAKER, John E. (Reg. No. 42,222)

MICHALIK & WYLIE, PLLC
14645 Bel-Red Road
Suite 103
Redmond, Washington USA 98007

to represent the undersigned before

☒ all the competent International Authorities

☐ the International Searching Authority only

☐ the International Preliminary Examining Authority only

in connection with any and all international applications filed by the undersigned with the following Office

United States Patent and Trademark Office

and to make or receive payments on behalf of the undersigned. _____ as receiving Office

Signature(s) (where there are several persons, each of them must sign; next to each signature, indicate the name of the person signing and the capacity in which the person signs, if such capacity is not obvious from reading this power):

Microsoft Corporation

by: Kate Sako
KATE SAKO

Title: ASSISTANT SECRETARY

Date: 2/29/2000

73

PATENT COOPERATION TREATY

306.233.0

13

From the RECEIVING OFFICE

To

ALBERT S. MICHALIK
704-228TH AVENUE NE, SUITE 193
SAMMAMISH, WASHINGTON 98074

25-APP

306.233.26

704-228TH AVENUE NE

PCT

NOTIFICATION CONCERNING PAYMENT
OF PRESCRIBED FEES

(PCT Rules 14, 15 and 16 and Administrative
Instructions, Sections 102bis(s), 304,
323(b), 707(b) and 803)

Date of mailing
(day/month/year)

24 Aug 2004

Applicant's or agent's file reference

4121

PAYMENT DUE

see item 3 for time limits

International application No.

PCT/US2004/024570

International filing date/Date of receipt

(day/month/year)

28 Jul 2004

Priority date (day/month/year)

23 Oct 2003

Applicant

MICROSOFT CORPORATION

1. The applicant is hereby notified that this receiving Office has received.



the payment of all the prescribed fees, and



an overpayment, which will be refunded in due course



no or insufficient payment of the prescribed fees and the applicant is hereby invited to pay the balance due, as summarized under item 2, within the time limit(s) indicated under item 3.

2. Fees and payment calculation:

5,810.00

Total fees payable

5,810.00

Amount paid

0.00

Balance



The details of the calculation are given in the Annex.

3. Time limit(s) for payment and amount(s) payable (Rules 14.1, 15.4 and 16.1(f)):



within ONE MONTH from the date of receipt of the international application (for the transmittal fee (if any), the search fee and the international filing fee). The amount payable for each fee is the amount applicable on the date of receipt of the international application



within 16 MONTHS from the priority date (only for the fee for priority document). The applicant's attention is drawn to the fact that the request made by the applicant under Rule 17.1(b) will be considered not to have been made unless the fee is paid within that time limit.

4. Additional observations (if necessary):



The search copy will not be transmitted to the International Searching Authority until the search fee is paid (therefore the start of the international search will be delayed) (Rule 23.1(a) and (b)).

Name and mailing address of the receiving Office

Mail Stop PCT, Commissioner for Patents
P.O. Box 1450, Alexandria, VA 22313-1450

Facsimile No. 703-305-3230

Authorized officer

Nhu Thuy Tran

Telephone No. 703-605-1537

Form PCT/RO/102 (January 2004)

ANNEX TO FORM PCT/RO/102
CALCULATION OF THE PRESCRIBED FEES

International application No
PCT: L S2004-024570

T Transmittal Fee

Prescribed amount: 300.00 **T**
Amount paid: 300.00
Balance: 0.00

☒ correct amount
☐ overpayment
☐ balance due

S Search Fee

Prescribed amount: 300.00 **S**
Amount paid: 300.00
Balance: 0.00

☒ correct amount
☐ overpayment
☐ balance due

I International Filing Fee

Fixed amount for first 30 sheets: 1,134.00 **I1**
338 x 12.00 = 4,056.00 **I2**
Number of sheets in excess of 30 Fee per sheet
Additional component: 400 x 0.00 = 0.00 **I3**
Fee per sheet

Reduction where the international application is filed
(See PCT Applicant's Guide, Volume I, General Part,
for details on the availability of this reduction)

using the PCT-EASY software: 0.00 **r**

or

in electronic form where the text of the
description, claims and abstract is not in
character coded format: 0.00 **r**

or

in electronic form where the text of the
description, claims and abstract is in character
coded format: 0.00 **r**

Sub-total: 5,190.00 **I1+I2+I3-r**

Prescribed total amount (The amount to be entered at I is the sub-total
entered at (I1+I2+I3-r), except where the applicant is (or all applicants
are) entitled to a reduction of 75%, in which case the amount to be
entered at I is 25% of the sub-total (I1+I2+I3-r). Certain applicants from
certain States are entitled to a reduction of 75% of the international
filing fee: see Notes to the Fee Calculation Sheet as annexed to the
Request Form, PCT/RO/101, for details)

Amount paid: 5,190.00 **I**
Balance: 0.00

☒ correct amount
☐ overpayment
☐ balance due

P Fee for Priority Document

Prescribed amount: 20.00 **P**
Amount paid: 20.00
Balance: 0.00

☒ correct amount
☐ overpayment
☐ balance due

PATENT COOPERATION TREATY

From the RECEIVING OFFICE

PCT

NOTIFICATION OF THE INTERNATIONAL
APPLICATION NUMBER AND OF THE
INTERNATIONAL FILING DATE

(PCT Rule 20.5(c))

To
ALBERT S. MICHALIK
704-228TH AVENUE NE, SUITE 193
SAMMAMISH, WASHINGTON 98074Date of mailing
(day/month/year) 24 Aug 2004Applicant's or agent's file reference
4121

IMPORTANT NOTIFICATION

International application No.
PCT/US2004/024570International filing date (day/month/year)
28 Jul 2004Priority date (day/month/year)
23 Oct 2003

Applicant

MICROSOFT CORPORATION

Title of the invention

MEDIA INTEGRATION LAYER

1. The applicant is hereby notified that the international application has been accorded the international application number and the international filing date indicated above.

2. The applicant is further notified that the record copy of the international application

☒ was transmitted to the International Bureau on 24 Aug 2004☐ has not yet been transmitted to the International Bureau for the reason indicated below and a copy of this notification has been sent to the International Bureau*☐ because the necessary national security clearance has not yet been obtained☐ because (reason to be specified)

* The International Bureau monitors the transmittal of the record copy by the receiving Office and will notify the applicant (with Form PCT/IB/301) of its receipt. Should the record copy not have been received by the expiration of 14 months from the priority date, the International Bureau will notify the applicant (Rule 22.1(c)).

3. FOREIGN TRANSMITTAL LICENSE INFORMATION

Completed by NT

☐ Additional license for foreign transmittal not required. This subject matter is covered by a license already granted or the equivalent U.S. national application. Refer to that license for information concerning its scope.☐ License for foreign transmittal not required. 37 CFR 5.11(c)(1) or 37 CFR 5.11(c)(2). However, a license may be required for additional subject matter. See 37 CFR 5.15(b).☒ Foreign transmittal license granted. 35 U.S.C. 184; 37 CFR 5.11 on 19 Aug 2004☒ 37 CFR 5.15(a)☐ 37 CFR 5.15(b)

Name and mailing address of the receiving Office

Mail Stop PCT, Commissioner for Patents
P.O. Box 1450, Alexandria, VA 22313-1450

Facsimile No. 703-305-3230

Authorized officer

Nhu Thuy Tran

Telephone No. 703-605-1537

Form PCT/RO/105 (July 1992)

PATENT COOPERATION TREATY

16

From the INTERNATIONAL SEARCHING AUTHORITY

PCT

NOTIFICATION OF RECEIPT
OF SEARCH COPY

(PCT Rule 25.1)

To:

ALBERT S. MICHALIK
704-228TH AVENUE NE, SUITE 193
SAMMAMISH, WASHINGTON 98074Date of mailing
(day/month/year)

24 Aug 2004

Applicant's or agent's file reference
4121

IMPORTANT NOTIFICATION

International application No.
PCT/US2004/024570International filing date (day/month/year)
28 Jul 2004Priority date (day/month/year)
23 Oct 2003

Applicant

MICROSOFT CORPORATION

1. Where the International Searching Authority and the receiving Office are not the same Office:

The applicant is hereby notified that the search copy of the international application was received by this International Searching Authority on the date indicated below.

Where the International Searching Authority and the receiving Office are the same Office:

The applicant is hereby notified that the search copy of the international application was received on the date indicated below

24 Aug 2004

(date of receipt).

2 ☐ The search copy was accompanied by a nucleotide and/or amino acid sequence listing or tables related thereto in computer readable form

3. Time limit for establishment of international search report and written opinion of the International Searching Authority

The applicant is informed that the time limit for establishing the international search report and the written opinion of the International Searching Authority is three months from the date of receipt indicated above or nine months from the priority date, whichever time limit expires later (Rules 42 I and 43bis I(a)).

4. A copy of this notification has been sent to the International Bureau and, where the first sentence of paragraph 1 applies, to the receiving Office

Name and mailing address of the ISA/

Mail Stop PCT, Commissioner for Patents
P.O. Box 1450, Alexandria, VA 22313-1450

Facsimile No. 703-305-3230

Authorized officer

Nhu Thuy Tran

Telephone No. 703-605-1537

Form PCT/ISA/202 (January 2004)

+X

PATENT COOPERATION TREATY

PCT

NOTIFICATION OF RECEIPT OF
RECORD COPY

(PCT Rule 24.2(a))

From the INTERNATIONAL BUREAU

To:

MICHALIK, Albert, S.
Suite 193
704-228th Avenue NE
Sammamish, WA 98074
United States of America

306233.26

Date of mailing (day/month/year) 22 September 2004 (22.09.2004)	IMPORTANT NOTIFICATION
Applicant's or agent's file reference 4121	International application No. PCT/US2004/024570

The applicant is hereby notified that the International Bureau has received the record copy of the international application as detailed below.

Name(s) of the applicant(s) and State(s) for which they are applicants:

MICROSOFT CORPORATION (for all designated States except US)

International filing date : 28 July 2004 (28.07.2004)
Priority date(s) claimed : 23 October 2003 (23.10.2003)
Date of receipt of the record copy
by the International Bureau : 30 August 2004 (30.08.2004)
List of designated Offices :

AP : BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW
EA : AM, AZ, BY, KG, KZ, MD, RU, TJ, TM
EP : AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, MC, NL, PL, PT, RO, SE, SI, SK, TR
OA : BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG
National : AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW

ATTENTION

The applicant should carefully check the data appearing in this Notification. In case of any discrepancy between these data and the indications in the international application, the applicant should immediately inform the International Bureau.

In addition, the applicant's attention is drawn to the information contained in the Annex, relating to:

- ☒ time limits for entry into the national phase - see updated important information (as of April 2002)
☐ requirements regarding priority documents (if applicable)

A copy of this Notification is being sent to the receiving Office and to the International Searching Authority.

The International Bureau of WIPO 34, chemin des Colombettes 1211 Geneva 20, Switzerland Facsimile No. (41-22) 338.89.75	Authorized officer: A. ZOLTANSKI (Fax : 338 89 75) Telephone No. (41-22) 338 8608

INFORMATION ON TIME LIMITS FOR ENTERING THE NATIONAL PHASE

The applicant is reminded that the "national phase" must be entered before each of the designated Offices indicated on the cover sheet of this Notification by paying national fees and furnishing translations, as prescribed by Articles 22 and 39 and the applicable national laws. In addition, the applicant may also have to comply with other special requirements applicable in certain Offices. It is the applicant's responsibility to ensure the necessary steps to enter the national phase are taken in a timely fashion. Most Offices do not issue reminders to applicants in connection with the entry into the national phase.

The applicable time limit for entering the national phase will, subject to what is said in the following paragraph, be **30 MONTHS** from the priority date, not only in respect of any elected Office if a demand for international preliminary examination is filed before the expiration of 19 months from the priority date (see Article 39(1)), but also in respect of any designated Office, in the absence of filing of such demand, where Article 22(1) as modified with effect from 1 April 2002 applies in respect of that designated Office. For further details, see PCT Gazette No. 44/2001 of 1 November 2001, pages 19926, 19932 and 19934, as well as the PCT Newsletter, October and November 2001 and February 2002 issues.

In practice, time limits other than the 30-month time limit will continue to apply, for various periods of time, in respect of certain designated or elected Offices. For regular updates on the applicable time limits (20, 21, 30 or 31 months, or other time limit), Office by Office, refer to the PCT Gazette ("Section IV" part published on a weekly basis), to the PCT Newsletter (on a monthly basis) and to the relevant National Chapters in Volume II of the PCT Applicant's Guide (the paper version of which is updated usually twice a year and the Internet version of which is updated usually on a weekly basis). Finally, a cumulative table of all applicable time limits for entering the national phase is available from WIPO's Internet site, via links from various pages the site including those of the Gazette, Newsletter and Guide, at <http://www.wipo.int/pct/en/index.html>.

Information about the requirements for filing a demand for international preliminary examination is set out in the PCT Applicant's Guide, Volume I/A, Chapter IX. Note that only an applicant who is a national or resident of a PCT Contracting State which is bound by Chapter II has the right to file a demand for international preliminary examination (at present, all PCT Contracting States are bound by Chapter II).

REQUIREMENTS REGARDING PRIORITY DOCUMENTS

For applicants who have not yet complied with the requirements regarding priority documents, the following is recalled.

Where the priority of an earlier national, regional or international application is claimed, the applicant must submit a copy of the said earlier application, certified by the authority with which it was filed ("the priority document") to the receiving Office (which will transmit it to the International Bureau) or directly to the International Bureau, before the expiration of 16 months from the priority date, provided that any such priority document may still be submitted to the International Bureau before that date of international publication of the international application, in which case that document will be considered to have been received by the International Bureau on the last day of the 16-month time limit (Rule 17.1(a)).

Where the priority document is issued by the receiving Office, the applicant may, instead of submitting the priority document, request the receiving Office to prepare and transmit the priority document to the International Bureau. Such request must be made before the expiration of the 16-month time limit and may be subjected by the receiving Office to the payment of a fee (Rule 17.1(b)).

If the priority document concerned is not submitted to the International Bureau or if the request to the receiving Office to prepare and transmit the priority document has not been made (and the corresponding fee, if any, paid) within the applicable time limit indicated under the preceding paragraphs, any designated State may disregard the priority claim, provided that no designated Office may disregard the priority claim concerned before giving the applicant an opportunity, upon entry into the national phase, to furnish the priority document within the time limit which is reasonable under the circumstances (Rule 17.1(c)).

Where several priorities are claimed, the priority date to be considered for the purposes of computing the 16-month time limit (and all other PCT time limits) is the filing date of the earliest application whose priority is claimed (Article 2(xi)(b)).

MEDIA INTEGRATION LAYER

FIELD OF THE INVENTION

The invention relates generally to computer systems, and more particularly to
5 the processing of graphical and other video information for display on computer systems.

BACKGROUND OF THE INVENTION

The limits of the traditional immediate mode model of accessing graphics on
10 computer systems are being reached, in part because memory and bus speeds have not kept up with the advancements in main processors and/or graphics processors. In general, the current (e.g., WM_PAINT) model for preparing a frame requires too much data processing to keep up with the hardware refresh rate when complex graphics effects are desired. As a result, when complex graphics effects are attempted with
15 conventional graphics models, instead of completing the changes that result in the perceived visual effects in time for the next frame, the changes may be added over different frames, causing results that are visually and noticeably undesirable.

A new model for controlling graphics output is described in United States Patent Application Serial Nos. 10/184,795, 10/184,796, 10/185,775, 10/401,717,
20 10/402,322 and 10/402,268, assigned to the assignee of the present invention and hereby incorporated by reference. This new model provides a number of significant improvements in graphics processing technology. For example, U.S. Serial No. 10/184,795 is generally directed towards a multiple-level graphics processing system and method, in which a higher-level component (e.g., of an operating system)
25 performs computationally intensive aspects of building a scene graph, updating animation parameters and traversing the scene graph's data structures, at a relatively low operating rate, in order to pass simplified data structures and/or graphics commands to a low-level component. Because the high-level processing greatly simplifies the data, the low-level component can operate at a faster rate, (relative to the
30 high-level component), such as a rate that corresponds to the frame refresh rate of the graphics subsystem, to process the data into constant output data for the graphics subsystem. When animation is used, instead of having to redraw an entire scene with

changes, the low-level processing may interpolate parameter intervals as necessary to obtain instantaneous values that when rendered provide a slightly changed scene for each frame, providing smooth animation.

U.S. Serial No. 10/184,796 describes a parameterized scene graph that
5 provides mutable (animated) values and parameterized graph containers such that program code that wants to draw graphics (e.g., an application program or operating system component) can selectively change certain aspects of the scene graph description, while leaving other aspects intact. The program code can also reuse already-built portions of the scene graph, with possibly different parameters. As can
10 be appreciated, the ability to easily change the appearance of displayed items via parameterization and/or the reuse of existing parts of a scene graph provide substantial gains in overall graphics processing efficiency.

U.S. Serial No. 10/185,775 generally describes a caching data structure and related mechanisms for storing visual information via objects and data in a scene
15 graph. The data structure is generally associated with mechanisms that intelligently control how the visual information therein is populated and used. For example, unless specifically requested by the application program, most of the information stored in the data structure has no external reference to it, which enables this information to be optimized or otherwise processed. As can be appreciated, this provides efficiency and
20 conservation of resources, e.g., the data in the cache data structure can be processed into a different format that is more compact and/or reduces the need for subsequent, repeated processing, such as a bitmap or other post-processing result.

While the above improvements provide substantial benefits in graphics processing technology, there still needs to be a way for programs to effectively use this
25 improved graphics model and its other related improvements in a straightforward manner. What is needed is a comprehensive yet straightforward model for programs to take advantage of the many features and graphics processing capabilities provided by the improved graphics model and thereby output complex graphics and audiovisual data in an efficient manner.

30

SUMMARY OF THE INVENTION

Briefly, the present invention provides a media integration layer which provides an API for programmers to accomplish possibly complex composition effects within their applications in a straightforward manner, while leveraging the graphics processing unit in a manner that does not adversely impact normal application performance. One aspect provides the ability to combine different media types (e.g., 2D, 3D, Video, Audio, text, imaging and so forth) and animate them together smoothly and seamlessly.

The MIL provides a graphics architecture for multi-stage composition, and a programming model that allows for functional parity at the programmatic and scripted interfaces. An API and script allows the creation of a retained structure or scene description that is composited when rendered, yet includes areas that have a more immediate-mode nature.

Via the interfaces, the MIL provides access to a data structure for storing visual information so that applications can take advantage of the graphics capabilities provided by the computer hardware. The interfaces support an element object model and a vector graphics markup language for using that element object model in a manner that allows program code developers to consistently interface with a scene graph data structure to produce graphics. The data structure may also be used for either directly rendering or for "compiling" the visual information so that it can be provided to a lower level graphics system for fast composition and animation.

The vector graphics element object model generally corresponds to shape elements and other elements including image and video elements that correlate with a scene graph object model of the scene graph. Markup may be parsed into data including elements in an element tree that is translated into the objects of a scene graph data structure. Other markup may be translated directly into data and calls that create the scene graph objects. The markup language provides distinct ways to describe an element, including a simple string format or complex property syntax, which may be named, enabling reuse in other locations in the markup.

An aspect of the MIL is the integration of animation and timing across the API set, providing animation as an inherent base-level concept. To facilitate smooth animation, the MIL provides a multiple-level graphics processing system and method

from an Element.

PreLayoutReadImpl This will be called on an AnimationEffect if it is dirty before we run layout in the RenderQueueItem. This is the time when the AnimationEffect should read values it needs. The reason reading and writing are separate is that reading causes layout to run immediately and if every AnimationEffect reads and writes in turn it will slow the whole process down.

PreLayoutWriteImpl This will be called on an AnimationEffect if it is dirty before we run layout in the RenderQueueItem. Although we don't guarantee the order in which AnimationEffects will be processed, we do guarantee that all dirty AnimationEffects will have had OnPreLayoutRead called before this is called.

PostLayoutReadImpl This will be called on an AnimationEffect if it is dirty after we run layout in the RenderQueueItem. If the IsAlwaysDirty flag is not set, the dirty flag on this AnimationEffect will have been set to false and it will have been removed from the list of AnimationEffects to be processed during the next RenderQueueItem. If the AnimationEffect calls SetDirty in this method, it will effectively keep it dirty for processing during the next RenderQueueItem. If the AnimationEffect would like to stay dirty, it's much more efficient to set the IsAlwaysDirty flag.

PRIMITIVE TYPES

The base length unit in the MIL is a double, whereby the other primitive types and APIs are based on doubles. Generally these doubles are evaluated as user units
5 that are initially equal to 1/96th of an inch. For colors, each of the color channels is represented by a float, and not a double. For angle measurements, double values are in degrees. When a float or double is evaluated as a time measurement, it is assumed to be seconds. ...

A Time structure represents a particular point in time or a span of time. In
10 addition, a special time value called "Indefinite" represents either a point in time

infinitely in the future or a span of time infinitely long. Time values are designed to be used in a property system, so a special value called "Unspecified" may be used to clear a property, or to indicate explicitly that the property is not set. Time values are internally stored as integer counts:

5

10

15

20

25

30

```
time:
    ( full-clock-value | partial-clock-value | timecount-value | special-value )
full-clock-value:
    hours ":" minutes ":" seconds ( "." fraction )?
partial-clock-value:
    minutes ":" seconds ( "." fraction )?
timecount-value:
    timecount ( "." fraction )? ( metric )?
special-value:
    ( "Indefinite" | "Unspecified" )
metric:
    "h" | "min" | "s" | "ms"
hours:
    digit+
minutes:
    2digit
seconds:
    2digit
fraction:
    digit+
timecount:
    digit+
2digit:
    digit digit
digit:
    "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

266

In addition to the above grammar, note that “minutes” and “seconds” need to be specified in the range “00” to “59” to be considered valid. Also, if the “timecount-value” format is used without units, the value is assumed to be in seconds. The following are a few examples of Time values and their meanings:

Time	Value
02:30:03	2 hours, 30 minutes and 3 seconds
50:00:10.25	50 hours, 10 seconds and 250 milliseconds
02:33	2 minutes and 33 seconds
00:10.5	10 seconds and 500 milliseconds
3.2h	3 hours and 12 minutes
45min	45 minutes
30s	30 seconds
5.45ms	5.45 milliseconds
12.467	12 seconds and 467 milliseconds
1d	1 day

5

The Time structure is used to store a single Time value:

10

15

20

```

public struct System.Windows.Media.Animation.Time : IComparable
{
    public Time(int milliseconds);

    public bool IsFinite { get; }
    public static Time Abs(Time t);
    public int CompareTo(Time other);
    public override bool Equals(object obj);
    public override int GetHashCode();
    public static Time Max(Time a, Time b);
    public static Time Min(Time a, Time b);
    public override string ToString();
    public static Time operator +(Time a, Time b);
    public static Time operator -(Time t);

```

267

```

public static Time operator -(Time a, Time b);
public static Time operator *(double s, Time t);
public static Time operator *(Time t, double s);
public static Time operator /(Time t, double s);
5 public static double operator /(Time a, Time b);
public static Time operator %(Time a, Time b);
public static bool operator ==(int n, Time time);
public static bool operator ==(Time time, int n);
public static bool operator ==(Time a, Time b);
10 public static bool operator !=(int n, Time time);
public static bool operator !=(Time time, int n);
public static bool operator !=(Time a, Time b);
public static bool operator >=(Time a, Time b);
public static bool operator <=(Time a, Time b);
15 public static bool operator >(Time a, Time b);
public static bool operator <(Time a, Time b);

public static readonly Time Immediately;
public static readonly Time Indefinite;
20 public static readonly int MaxValue;
public static readonly int MinValue;
public static readonly Time Unspecified;
}

```

25 Other basic types are set forth below, in which the following notation is used:

- *: 0 or more
 - +: 1 or more
 - ?: 0 or 1
 - {n}: n times
 - (): grouping
 - |: separates alternatives
- 30

- double quotes surround literals

	wsp:
	whitespace+
5	digit:
	"0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
	hex-digit:
10	digit "a" "A" "b" "B" "c" "C" "d" "D" "e" "E" "f" "F"
	digit-sequence:
	digit+
15	sign:
	"-" "+"
	exponent:
	("e" "E") sign? digit-sequence
20	fractional-constant:
	(digit-sequence? "." digit-sequence) (digit-sequence ".")
	floating-point-constant:
25	(fractional-constant exponent?) (digit-sequence exponent)
	integer-constant:
	digit-sequence
30	comma:
	","
	comma-wsp:

263

(wsp comma? wsp*) | (comma wsp*)

number:

(sign? (integer-constant | floating-point-constant)) | "Unset"

5

number-wsp:

number wsp*

coordinate:

10 number-wsp

coordinate-pair:

coordinate comma-wsp coordinate

coordinate-pairs:

15 (coordinate-pair comma-wsp)* coordinate-pair

filename:

wsp* ("" valid-filename-char+ "" | valid-filename-char-no-whitespace) wsp*

20

Markup syntax for colors:

numeric-color:

"#" (hex-digit{3} | hex-digit{4} | hex-digit{6} | hex-digit{8})

25

color:

numeric-color | [a-zA-Z]+

The Colors object contains static members, comprising many well known colors such as Red and Blue:

30

28

```
public sealed class Colors
{
    public static Color AliceBlue { get; }
    public static Color AntiqueWhite { get; }
    public static Color Aqua { get; }
    public static Color Aquamarine { get; }
    public static Color Azure { get; }
    public static Color Beige { get; }
    public static Color Bisque { get; }
    public static Color Black { get; }
    public static Color BlanchedAlmond { get; }
    public static Color Blue { get; }
    public static Color BlueViolet { get; }
    public static Color Brown { get; }
    public static Color BurlyWood { get; }
    public static Color CadetBlue { get; }
    public static Color Chartreuse { get; }
    public static Color Chocolate { get; }
    public static Color Coral { get; }
    public static Color CornflowerBlue { get; }
    public static Color Cornsilk { get; }
    public static Color Crimson { get; }
    public static Color Cyan { get; }
    public static Color DarkBlue { get; }
    public static Color DarkCyan { get; }
    public static Color DarkGoldenrod { get; }
    public static Color DarkGray { get; }
    public static Color DarkGreen { get; }
    public static Color DarkKhaki { get; }
    public static Color DarkMagenta { get; }
    public static Color DarkOliveGreen { get; }
    public static Color DarkOrange { get; }
```

```

public static Color DarkOrchid { get; }
public static Color DarkRed { get; }
public static Color DarkSalmon { get; }
public static Color DarkSeaGreen { get; }
public static Color DarkSlateBlue { get; }
public static Color DarkSlateGray { get; }
public static Color DarkTurquoise { get; }
public static Color DarkViolet { get; }
public static Color DeepPink { get; }
public static Color DeepSkyBlue { get; }
public static Color DimGray { get; }
public static Color DodgerBlue { get; }
public static Color Firebrick { get; }
public static Color FloralWhite { get; }
public static Color ForestGreen { get; }
public static Color Fuchsia { get; }
public static Color Gainsboro { get; }
public static Color GhostWhite { get; }
public static Color Gold { get; }
public static Color Goldenrod { get; }
public static Color Gray { get; }
public static Color Green { get; }
public static Color GreenYellow { get; }
public static Color Honeydew { get; }
public static Color HotPink { get; }
public static Color IndianRed { get; }
public static Color Indigo { get; }
public static Color Ivory { get; }
public static Color Khaki { get; }
public static Color Lavender { get; }
public static Color LavenderBlush { get; }
public static Color LawnGreen { get; }

```

268

```

public static Color LemonChiffon { get; }
public static Color LightBlue { get; }
public static Color LightCoral { get; }
public static Color LightCyan { get; }
public static Color LightGoldenrodYellow { get; }
public static Color LightGray { get; }
public static Color LightGreen { get; }
public static Color LightPink { get; }
public static Color LightSalmon { get; }
public static Color LightSeaGreen { get; }
public static Color LightSkyBlue { get; }
public static Color LightSlateGray { get; }
public static Color LightSteelBlue { get; }
public static Color LightYellow { get; }
public static Color Lime { get; }
public static Color LimeGreen { get; }
public static Color Linen { get; }
public static Color Magenta { get; }
public static Color Maroon { get; }
public static Color MediumAquamarine { get; }
public static Color MediumBlue { get; }
public static Color MediumOrchid { get; }
public static Color MediumPurple { get; }
public static Color MediumSeaGreen { get; }
public static Color MediumSlateBlue { get; }
public static Color MediumSpringGreen { get; }
public static Color MediumTurquoise { get; }
public static Color MediumVioletRed { get; }
public static Color MidnightBlue { get; }
public static Color MintCream { get; }
public static Color MistyRose { get; }
public static Color Moccasin { get; }

```

```
public static Color NavajoWhite { get; }
public static Color Navy { get; }
public static Color OldLace { get; }
public static Color Olive { get; }
public static Color OliveDrab { get; }
public static Color Orange { get; }
public static Color OrangeRed { get; }
public static Color Orchid { get; }
public static Color PaleGoldenrod { get; }
public static Color PaleGreen { get; }
public static Color PaleTurquoise { get; }
public static Color PaleVioletRed { get; }
public static Color PapayaWhip { get; }
public static Color PeachPuff { get; }
public static Color Peru { get; }
public static Color Pink { get; }
public static Color Plum { get; }
public static Color PowderBlue { get; }
public static Color Purple { get; }
public static Color Red { get; }
public static Color RosyBrown { get; }
public static Color RoyalBlue { get; }
public static Color SaddleBrown { get; }
public static Color Salmon { get; }
public static Color SandyBrown { get; }
public static Color SeaGreen { get; }
public static Color SeaShell { get; }
public static Color Sienna { get; }
public static Color Silver { get; }
public static Color SkyBlue { get; }
public static Color SlateBlue { get; }
public static Color SlateGray { get; }
```

```

public static Color Snow { get; }
public static Color SpringGreen { get; }
public static Color SteelBlue { get; }
public static Color Tan { get; }
public static Color Teal { get; }
public static Color Thistle { get; }
public static Color Tomato { get; }
public static Color Transparent { get; }
public static Color Turquoise { get; }
public static Color Violet { get; }
public static Color Wheat { get; }
public static Color White { get; }
public static Color WhiteSmoke { get; }
public static Color Yellow { get; }
public static Color YellowGreen { get; }
}

```

A point struct is set forth below:

```

5 public struct System.Windows.Point
{
    public Point(); // initializes to 0,0
    public Point(double x, double y);

    public static Boolean Equals(Point point1, Point point2)
    public virtual Boolean Equals(Object o)
10
    public Double X { get; set; }
    public Double Y { get; set; }

    public virtual Int32 GetHashCode()
15
    public static Point operator+(Point point, Vector vector)

```

269

```

5 public static Boolean operator==(Point point1, Point point2)
  public static Boolean operator!=(Point point1, Point point2)
  public static Point operator*(Point point, Matrix matrix)
  public static Point operator-(Point point, Vector vector)
  public static Vector operator-(Point point1, Point point2)

  // This will take the absolute value of X and Y, since Size cannot be negative
  public static explicit operator Size(Point point) //
  public static explicit operator Vector(Point point)

10
  public static Point Add(Point point, Vector vector)
  public static Point Multiply(Point point, Matrix matrix)
  public Void Offset(Double dx, Double dy)
  public static Point Subtract(Point point, Vector vector)
15 public static Vector Subtract(Point point1, Point point2)
  public virtual String ToString()
  }

```

Markup syntax for a point object:

```

20 point:
    coordinate-pair

```

A Vector object is set forth below:

```

public struct System.Windows.Vector
{
  public Vector(); // initializes to 0,0
  public Vector(double x, double y);

  public double X { get; set; }
  public double Y { get; set; }
}

```

270

```

public static Boolean Equals(Vector vector1, Vector vector2)
public virtual Boolean Equals(Object obj)

public Double Length { get; }
public Double LengthSquared { get; }
public Double X { get; set; }
public Double Y { get; set; }

public virtual Int32 GetHashCode()

public static Point operator+ (Vector vector, Point point)
public static Vector operator+ (Vector vector1, Vector vector2)
public static Vector operator/ (Vector vector, Double scalar)
public static Boolean operator==(Vector vector1, Vector vector2)
public static Boolean operator!=(Vector vector1, Vector vector2)
    // return the dot product: vector1.X*vector2.X + vector1.Y*vector2.Y
public static Double operator*(Vector vector1, Vector vector2)
public static Vector operator* (Double scalar, Vector vector)
public static Vector operator* (Vector vector, Double scalar)
public static Vector operator* (Vector vector, Matrix matrix)
public static Vector operator-(Vector vector1, Vector vector2)
public static Vector operator- (Vector vector)

    // This will take the absolute value of X and Y, since Size cannot be negative
public static explicit operator Point (Vector vector)
public static explicit operator Size (Vector vector)

public static Vector Subtract(Vector vector1, Vector vector2)
public static Double Multiply(Vector vector1, Vector vector2)
public static Vector Multiply(Double scalar, Vector vector)
public static Vector Multiply(Vector vector, Double scalar)
public static Vector Multiply(Vector vector, Matrix matrix)

```

```

public static Vector Divide(Vector vector, Double scalar)
public static Point Add(Vector vector, Point point)
public static Vector Add(Vector vector1, Vector vector2)

public Void Normalize()
    // return the angle required to rotate v1 into v2, in degrees
    // This will return a value between (-180, 180] degrees
public static Double AngleBetween(Vector vector1, Vector vector2)
public static Double CrossProduct(Vector vector1, Vector vector2)

    // return the Determinant: vector1.X*vector2.Y - vector1.Y*vector2.X
public static Double Determinant(Vector vector1, Vector vector2)

public virtual String ToString()

}

```

Markup syntax for a Vector object:

```

size:
    coordinate-pair

```

5

Note that size is not a Vector, and thus it cannot be added, subtracted or transformed. Also, size cannot have a negative width or height. An `ArgumentException` will be thrown if an attempt is made to set one.

10 With respect to rectangles, the name Rect is used instead of Rectangle so as to not collide with the Rectangle element. While it is possible to specify a negative width and height, there is no standard way to normalize the rectangle and many operations on the rectangle may give non-intuitive results. Rect's Width and Height cannot be set to non-negative values. If such values are passed to the constructor, the result will be normalized. If Width or Height is set to negative, this will result in an

15 `ArgumentException`.

272


```

// This is only set if the GeometryHitTestParameters.IntersectionDetail
public IntersectionDetail IntersectionDetail { get; }
}

```

5

To illustrate use of delegates, consider a hit tester who wants the first top most hit using anonymous delegates:

```

public Visual HitTest(Visual visual, Point pt)
{
10   Visual visualHit = null;

    visual.HitTest(
        new PointHitTestParameters(pt),
        null,
15   HitTestResultDelegate(HitTestResult htr) {
        visualHit = htr.Visual;
        return HitTestResultBehavior.Stop;
    }
    );
20   return visualHit;
}

```

Another example is a hit tester who wants to return all of the visuals that were hit:

25

```

public Visual[] HitTest(Visual visual, Point pt)
{
    ArrayList visualsHit = new ArrayList();

30   visual.HitTest(
        new PointHitTestParameters(pt),
        null,

```

250

```
HitTestResultDelegate(HitTestResult htr) {  
    visualsHit.Add(htr.Visual);  
    return HitTestResultBehavior.Continue;  
}  
5    );  
}
```

The hit tester uses enums to control hit test filtering and result behavior:

```
10 public enum HitTestFilterBehavior  
    {  
        //  
        ContinueSkipChildren,  
        ContinueSkipVisualAndChildren,  
        ContinueSkipVisual,  
15    Continue,  
        Stop  
    }
```

20 The HitTestFilterBehavior enum controls filtering behavior in that specifying SkipChildren hit tests this visual, but not its children visuals. SkipVisualAndChildren specifies not to hit test the visual or children visuals. SkipVisual specifies not to hit test the visual, but hit test any children visuals. Continue specifies to hit test this visual and its children visuals. Stop specifies to not hit test any visuals in the visual tree and return to caller.

25 The HitTestResultBehavior enum controls hit test behavior:

```
public enum HitTestResultBehavior  
{  
    Stop,  
30    Continue  
}
```

28

Stop specifies to return to the hit test entry, skipping any further filter or hit test operations. Continue specifies to hit test against the next visual.

Although hit test identifiers may be used to mark specific content for positive hit identification, performance was poor because such a model broke up the render stream, added to walk overhead and was difficult to manage when hit testing. With the combining of element and visual into a unified type, the basic level of granularity is the visual itself, and controls can structure themselves to get the level of granularity they desire.

The control author writes logic for hit by overriding HitTestCore and doing its own computation and/or using the services described below.

The following are some examples that demonstrate the power of these services.

A first example demonstrates a control having a public HitRegion property that represents the hit sensitive region of the control. Note that the hit region need not match the rendered contents, and may be optimized by some applications. If the hit region is not set (`_hitRegion == null`) the control defers to the base implementation services to determine a hit.

```

public class HitRegionControl : Control           // derives from visual.
{
    private Geometry _hitRegion;

    public Geometry HitRegion
    {
        get
        {
            return _hitRegion;
        }
        set
        {
            _hitRegion = value;
        }
    }
}

```

```
protected virtual PointHitTestResult HitTestCore(PointHitTestParameters
    htParams)
{
5   bool IsHit = (_hitRegion != null) ?
        _hitRegion.DoesContain(htParams.Point) : IsHitRenderContents(htParams);

    return isHit ? new PointHitTestResult(this, htParams.Point) : null;
}

10 protected virtual GeometryHitTestResult HitTestCore(
    GeometryHitTestParameters htParams)
{
    IntersectionDetail intersectDetail = (_hitRegion != null) ?
15     _hitRegion.DoesContain(
        htParams.Geometry,
        htParams.ComputeIntersectionDetail) : HitTestRenderContents(htParams);

    return (intersectDetail != IntersectionDetail.Empty) ?
20     new GeometryHitTestResult(
        this,
        htParams.Geometry,
        intersectDetail) : null;
}

25 protected virtual Rect HitTestBounds
{
    get
    {
        return (_hitRegion != null) ? _hitRegion.Bounds : GetContentBoundingBox()
30     }
}
}
```

R22

To override IsHit behavior use additional support services.

The Geometry classes perform a hit test against its interior area:

```

5 public abstract class Geometry : Changeable
{
    public virtual bool DoesContain(Point point);

    public virtual bool DoesContain(Geometry geometry);
    public virtual IntersectionDetail DoesContainWithDetail(Geometry geometry);
10 }

```

The visual provides protected functions to hit test against the rendered contents (of itself). If a visual is retained, this will trigger a content validation. This helper examines the drawing instruction stream stored on the visual, one instruction at a time, hit testing the point or geometry for each with the rendered geometry.

```

20 public class Visual : DependencyObject, IDisposable, IVisual
{
    protected Rect VisualDescendantBounds { get; }
    protected Rect VisualContentBounds { get; }
}

```

The code will return if the image pixel at the point is above the alpha threshold. The point is in visual space and the transform is to device space where the pixel based test occurs.

```

    public class ImageData : ImageSource
{
    public virtual bool HitTestImageContents(
        HitTestParameters htParams,
        Transform transform);
30 }

```

ANIMATION

The animation system is comprised of two main sets of components, namely a timing control engine and a set of animation objects. The timing engine is a service that can be used by any objects that exhibit time-varying behaviors, the main examples
 5 being animations and audio or video media objects. Animation objects implement a set of functions that map time spans into other data types, which are then used as inputs into other higher-level objects.

Graphical animation is achieved by associating an animation collection with a rendering operation. For example, the `IDrawingContext.DrawLine` method takes a pen
 10 and two end points. One of the end points may be associated with a collection of `PointAnimation` objects, in which case the line will move over time. Similarly, the pen may have an associated collection of `ColorAnimation` objects. In such cases, each animation used in a rendering operation may be run on a separate clock, sometimes referred to as a called a "timeline." Once an animated primitive is drawn, the rendering
 15 system takes care of redrawing the scene at regular intervals. Each time a frame is rendered the current value of the animations involved in the scene is computed, based on the elapsed time (in most cases measured by the system clock), and then the animated primitives are redrawn.

Programming animations requires an understanding both of the animation
 20 objects provided by the system as well as the timing engine driving those animations. The following terms are used in several places in this section.

A Timing model is provided in which timed objects participate in a hierarchical timing system where individual timelines have attributes that define their behavior relative to their parent timeline, or, for top-level timelines, relative to a root
 25 "document" (or "page", or "frame") timeline. Timing attributes are a set of parameters that defines the time behavior of an object. Timing attributes are exclusively descriptive and have no run-time state. In addition, timing attributes are immutable.

A timeline is an instance of a timing entity that maintains a run-time state according to a set of timing attributes. A timeline defines the concept of "now" for a
 30 timed object. A timing-tree is a data structure containing a set of timelines arranged in a hierarchical fashion. The relationship between the timelines is defined by a set of inheritance rules and by the timing attributes associated with each timeline.

A timed object is any object that exhibits a time-varying behavior. The description of the time behavior of a timed object is specified by a set of timing attributes, whereas its run-time timing state is maintained by one or more timelines. An animation function is a function that takes a base value of a particular data type as an input and produces a value of the same type as its output. An animation function may or may not take other implicit or explicit input parameters, such as the current time value of a Timeline. In that respect, an animation function may not be constant, in that the same input may produce different outputs at different times.

A modifier is an object that implements an animation function and is used to modify the value of a property of an Element, some other complex object, or a parameter to a rendering call. A timed modifier is a modifier that is associated with a Timeline, and whose animation function explicitly depends on the run-time state of that Timeline. An animation is a timed modifier that implements a certain known set of animation functions.

An animation collection is a collection of modifiers that process the same data type. An animation collection chains the output of a modifier to the input of another, creating a modifying pipeline. Since the whole collection takes one input and produces one output, the collection itself behaves as a modifier.

Timelines can be thought of as stopwatches that control time-varying processes, such as the playback of a video clip or an animation. Times that are specified in the attributes of a timeline are relative to something. In most cases they are relative to a parent timeline, but for timelines at the root of the tree the values are relative to "document time," where document time is an implicit timeline that starts when the application is launched, or when a page or frame is navigated. The clock in a timeline is exposed in two ways: as an offset from the begin point, or as a progress ratio between 0 and 1. The latter is simply the ratio of the current time to the duration.

The simplest timeline has a begin time and a duration. For example, a timeline with a begin time of three seconds and a duration of five seconds "begins" three seconds after the reference $t=0$ time (by default, the moment application is loaded), and "ends" five seconds later. During those five seconds the timeline is said to be "on." If this timeline controls an animation, this animation is changing (e.g., moving)

for that time, but it is static before and after. FIG. 29 shows a timeline with a Begin time of three and a Duration of five.

A timeline can also be programmed to repeat its behavior. This repeat can be specified as an iteration count, or a repeat duration. The timeline goes through as many begin to end runs as necessary to fill the requested count or duration. If the repeat count is not an integer value, the last iteration is interrupted in the middle. FIG. 30 shows a timeline with Begin=3, Duration=5 and RepeatDuration=17 (meaning the animation will repeat, every five seconds, until seventeen seconds after the begin time or twenty seconds).

The begin time for a timeline is normally relative to its parent timeline (or to document time), but a begin time may also be specified relative to another timeline's begin or end. In such a situation, every begin (or end) in the source timeline causes a corresponding begin to be scheduled for the target timeline. FIG. 31 shows a timeline with a begin time 3s after another's timeline.

When a timeline reaches the end point it is immediately turned "off." At that time, the timed object that it controls ceases to have an effect on the presentation. For example, if the timed object is an animation then when the controlling timeline reaches the end point the animation is removed, i.e., it reverts back to its base value. There are cases, however, where it is desirable to have the final steady state of an animation be frozen at the last value. In other words, the timeline progresses from 0 to 1 between the begin and end points, but after the end point it remains "on" with a progress of 1. This is called a "fill" behavior. FIG. 32 represents a timeline with Begin=3, Duration=5 and Fill=Freeze

Time flows linearly from a progress value of 0 to a progress value of 1, from the point of view of a timeline. However, the relationship between the passage of time inside a timeline and inside its parent can be altered from the default direct correlation, in that time may be reversed in a timeline so that it appears to flow backwards, the rate at which time passes may be sped up or slowed down by a multiplicative factor, and/or the progress curve may be morphed so that instead of progressing linearly from 0 to 1, it accelerates from a standstill at the begin point to a maximum rate of progress, and then decelerates towards a standstill at the end point. This produces an "ease-in, ease-out" effect for any animations controlled by this timeline.

226

More particularly, the progress/time curve is linear by default. When this linear curve is used to control certain animations, the user perceives a "jerk" effect at the begin and end points because the animation starts and ends suddenly. For those cases, a timeline can be programmed to accelerate the passage of time from a standstill to a maximum rate using a smooth acceleration curve. Similarly, time may be programmed to decelerate towards zero near the end point. The acceleration and deceleration effects are specified as the percentage of the duration spent in the acceleration or deceleration phases. The two values are positive, and their sum is not to exceed one. FIG. 33 shows A timeline with Begin=3, Duration=10, Acceleration=0.2 and Deceleration=0.4

One straightforward time manipulation involves programming a timeline to go from a progress value of zero to one, and then back to zero. In that case, the timeline is active for twice the specified duration, once for the "forward" part and once again for the "backwards" part. FIG. 34 shows a timeline with Begin=3, Duration=5 and AutoReverse=True.

The apparent passage of time for a timeline can be faster or slower than that for its parent, by a constant factor. By default this factor is 1, which means time in a timeline and its parent passes at the same rate. If instead this value is greater than one, then time for the timeline progresses at a faster rate than for its parent. For example, a factor of three makes the timeline go between the begin and end points three times faster than the specified duration. Conversely, if the factor is between zero and one then time passes at a slower rate. If the factor is negative then time in the timeline always appears to move backwards relative to its parent. Note that the begin time itself is an offset in the frame of reference of that parent timeline. As a result, while the duration of the timeline is affected by the speed factor, the begin time is not. FIG. 35 shows a timeline with Begin=3, Duration=5 and Speed=0.5

Timelines may be organized in a tree structure. Every document, frame or window has some implicit "root" timeline, which can be thought of as representing real-world, wall-clock time. However, time $t=0$ for the root timeline is the time when that timeline is created, that is, when a document is loaded, a frame is navigated or a window is opened. --

Given the hierarchical nature of the timing system, it makes sense to refer to the passage of time as occurring in one of three frames of reference. A simple frame

of reference is the frame of reference experienced by an individual timeline. In this frame of reference, the progress value of a timeline is always 0 at $t=0$ and 1 at $t=d$, where d is the simple duration of the timeline. The duration of a timeline is always specified in the simple frame of reference. The parent timeline's frame of reference is the simple frame of reference for the timeline that is the parent of any given timeline. For example, the Begin time for a timeline is always specified in the parent timeline's frame of reference. The global frame of reference is the simple frame of reference of the root timeline. In that frame of reference, time $t=5s$ occurs five seconds after the timeline is created, and a duration of 10s lasts for exactly ten real-world seconds.

In addition, the various timing control rules apply to timing subtrees, including that if a timeline is active, its parent timeline also needs to be active. Conversely, if a timeline is not active, then none of its children may be active, and none may be started. If a timeline is explicitly paused (via a call to the `ITimingControl.Pause` method) then its children are implicitly paused. When that timeline is resumed, any of its children that were not explicitly paused are also resumed. If a timeline is started (for any of a variety of reasons, including crossing a repeat point) then its children are reset.

A timeline may be explicitly parented to another timeline, in which case the shape of the timing tree is explicit and clear. In many cases, however, it is useful to let the system parent a timeline automatically, based on some default timing parent. A timeline that is not explicitly parented is said to be auto-parented, and its effective parent timeline depends on how the timeline is used. Two types of auto-parenting are supported: parenting to the visual parent, or parenting to the root.

The visual parent of a timeline is determined implicitly by how the timeline is used. For example, if the timeline controls a color animation which, in turn, animates a brush used as the background of some visual V , then V is the "visual parent" of that timeline. If that Visual has an associated default timeline, that timeline is the parent of our original timeline in this example. Otherwise, the parent of the Visual is examined, recursively. The root of a Visual tree is always implicitly associated with the root timeline, so if the Visual is in the Visual tree then any auto-parented timelines within it are guaranteed to be parented somewhere in the timing tree. However, if the Visual is not yet in the Visual tree then its timelines remain outside of the timing tree until such time as the Visual gets inserted into a tree.

The default "root" timeline is also defined by visual parentage, except that, in that case, the closest Visual parent that has a timeline is not necessarily used. Rather, with root parentage a timeline is always associated with the highest Visual in the tree (which may be a Frame or Window object, or the root Visual associated with a
5 VisualManager).

Once a timeline is automatically parented, it may need to be re-parented if anything happens that changes the implicit default parent timeline. For example, if the immediate visual parent of the timeline doesn't initially have a default timeline of its own but then one is set then the timeline needs to be re-parented. This re-parenting
10 occurs automatically. Auto-parenting and re-parenting are implemented by the IAnimatable interface, described below.

Timelines and timed objects share a number of behaviors in common. For example, an animation can be paused or restarted, and a list of animations may be active or inactive. To maintain consistency, timed objects implement one or more
15 interfaces that allow access to timing methods and properties.

The ITimingControl interface is implemented by timed objects that can be controlled at run-time:

```
public interface System.Windows.Media.Animation.ITimingControl
{
20 // Timing attributes
    double Acceleration { get; set; }
    bool AutoReverse { get; set; }
    TimeSyncValue Begin { get; set; }
    double Deceleration { get; set; }
25 Time Duration { get; set; }
    TimeSyncValue End { get; set; }
    TimeEndSync EndSync { get; set; }
    TimeFill Fill { get; set; }
    TimeFill FillDefault { get; set; }
30 Timeline ParentTimeline { get; set; }
    double RepeatCount { get; set; }
    Time RepeatDuration { get; set; }
```

```

TimeRestart Restart { get; set; }
TimeRestart RestartDefault { get; set; }
double Speed { get; set; }

5 // Run-time timing state
int CurrentRepeat { get; }
Time CurrentTime { get; }
void Disable();
void Enable();
10 bool IsChanging { get; }
bool IsEnabled { get; }
bool IsForwardProgressing { get; }
bool IsOverridingBaseValue { get; }
bool IsPaused { get; }
15 bool IsReversed { get; }
double Progress { get; }

// Run-time timing control
void BeginIn(Time offset);
20 void Disable();
void Enable();
void EndIn(Time offset);
void Pause();
void Resume();
25 void Seek(Time offset, TimeSeekOrigin origin);
void OverrideSpeed(double speed);

// Event notifications
event EventHandler Begun { add; remove; }
30 event EventHandler Changed { add; remove; }
event EventHandler Ended { add; remove; }
event EventHandler Paused { add; remove; }

```

```
void IList.Remove(object value);
public void RemoveAt(int index);

// Extra functions
5 public int Capacity { get; set; }
  public PathFigure this[int index] { get; set; }
  public int Add(PathFigure figure);
  public bool Contains(PathFigure figure);
  public int IndexOf(PathFigure figure);
10 public int IndexOf(PathFigure figure, int startIndex);
  public int IndexOf(PathFigure figure, int startIndex, int count);
  public int LastIndexOf(PathFigure figure);
  public int LastIndexOf(PathFigure figure, int startIndex);
  public int LastIndexOf(PathFigure figure, int startIndex, int count);
15 public void Insert(int index, PathFigure figure);
  public void Remove(PathFigure figure);
  public void AddRange(PathFigureCollection figureCollection);
  public void SetRange(int index, PathFigureCollection figureCollection);
  public void InsertRange(int index, PathFigureCollection figureCollection);
20 public void RemoveRange(int index, int count);
  public PathFigureCollection GetRange(int index, int count);
}
```

A PathFigure is a sub-section of a Geometry that defines a segment collection.

25 This segment collection is a single connected series of two-dimensional PathSegment objects. The PathFigure can be a closed shape with a defined area, or a connected series of Segments that define a curve, but no enclosed area. The PathFigure class includes a number of convenience functions to allow simpler construction of a PathFigure from explicit ArcTo/LineTo/(and other) method calls without requiring a

30 PathSegment object. The explicit AddSegment call can be used to add a composed Segment.

158

```

public sealed class PathFigure : Animatable
{
    public PathFigure();
    public PathFigure(params PathSegment[] segments);
5   public PathFigure(ICollection segmentCollection);
    public PathFigure(ICollection segmentCollection, bool isFilled);
    public new PathFigure Copy();

    [DefaultValue(true)]
10   public bool IsFilled { get; set; }

    // Start the figure at an initial point.
    public void StartAt(Point pt);
    public void StartAt(Point pt, PointAnimationCollection ptAnimations);
15

    // Adds a line segment from the current point to the specified
    // new point. The current point is updated to the new location.
    public void LineTo(Point pt);
    public void LineTo(Point pt, PointAnimationCollection ptAnimations);
20

    // Adds a polyline starting at the current point, then to
    // each point in the points array. The current point is
    // updated to the last point of the array.
    public void PolyLineTo(Point[] points);
25

    public void ArcTo(
        Point pt,
        Size sz,
        double xRotation,
30   bool largeArcFlag,
        bool sweepFlag);

```

30

```

public void ArcTo(
    Point pt,
    PointAnimationCollection ptAnimations,
    Size sz,
    SizeAnimationCollection szAnimations,
    double xRotation,
    DoubleAnimationCollection xRotationAnimations,
    bool largeArcFlag,
    bool sweepFlag);

// Adds a cubic bezier segment using the current point as the
// first control point. The current point is updated to the
// the destination point.
public void BezierTo(
    Point pt1,
    Point pt2,
    Point ptDest);

public void BezierTo(
    Point pt1,
    PointAnimationCollection pt1Animations,
    Point pt2,
    PointAnimationCollection pt2Animations,
    Point ptDest,
    PointAnimationCollection ptDestAnimations);

// Adds cubic Bezier sections using the current point as the
// first control point, then iterating through 3 points at a
// time. The current point is updated to the last point of
// the array. When the number of points passed in is not a
// multiple of 3, we throw an invalid arg exception.
public void PolyBezierTo(Point[] points);

```

51

```

// Adds a quadratic Bezier section using the current point
// as the first control point. The current point is updated to
// destination point.
5 public void QuadraticBezierTo(Point pt1, Point ptDest);

public void QuadraticBezierTo(
    Point pt1,
    PointAnimationCollection pt1Animations,
10    Point ptDest,
    PointAnimationCollection ptDestAnimations);

// Adds quadratic Bezier sections using the current point
// as the first control point, then iterating through 2 points
15 // at a time. The current point is updated to the last
    // point of the array. When the number of points passed in is
    // not a multiple of 2, we throw an invalid arg exception.
    public void PolyQuadraticBezierTo(Point[] points);

20 // Close the figure. No new segments can be added.
    public void CloseFigure();

// Stroke the segments of new figure. Default is true.
    public void StrokeNewSegments(bool strokeNewSegments);
25

// Add a new Segment to the PathFigure
    public void AddSegment(PathSegment segment);
    public PathSegmentCollection Segments { get; set; }

30 public PathFigure GetFlattenedPathFigure(float flatness);
}

```


52

A figure requires a starting point, since each segment maintains continuity relative to the last point added. To specify the starting point `StartAt(pt)` or `Add(new StartSegment(pt))` is called. After adding segments, to add a properly closing segment that joins the last point back to the starting point, `CloseFigure()` or `Add(new CloseSegment())` is used. The start and close segments appear in the segment collection.

An exception is thrown if a `PathFigure` is built and the `StartSegment` is not the first segment in the collection or the `CloseSegment`, if present, is not at the last segment in the collection. The `StartSegment` and `CloseSegment` are not valid at any other positions in the figure, an exception being a completely empty segment collection.

The `PathFigure.IsFilled` property explicitly controls whether or not the contained area of a closed figure is to be used for hit-testing, rendering, and clipping. If this property is set to false, then only the outline of the `PathFigure` will be used, and its contained area will not contribute to the overall area of the `PathGeometry`. The default value for this property is true.

To enumerate the contents of a `PathFigure` as points, one straightforward way is to flatten the figure and examine the resulting `PathSegmentCollection`. The flattening process is lossy with respect to animation and curve segment parameters, however the raw point data is exposed through a `PolyLineSegment` for further point processing.

A `PathSegmentCollection` is a collection of `PathSegment` objects and the primary contents defining a `PathFigure`:

```
25 public sealed class PathSegmentCollection : Animatable, IAddChild, IList
    {
        public PathSegmentCollection();
        public PathSegmentCollection(int capacity);
        public PathSegmentCollection(params PathSegment[] segments);
30 public PathSegmentCollection(ICollection segments);
        public new PathSegmentCollection Copy();
```

```

// IEnumerable
public IEnumerator GetEnumerator();

// ICollection
5 public int Count { get; }
  public bool IsSynchronized { get; }
  public object SyncRoot();
  public void CopyTo(Array array, int index);

10 // IList
  public bool IsFixedSize { get; }
  bool IList.IsReadOnly { get; }
  object IList.this[int index] { get; set; }
  int IList.Add(object value);
15 public void Clear();
  bool IList.Contains(object value);
  int IList.IndexOf(object item);
  void IList.Insert(int index, object value);
  void IList.Remove(object value);
20 public void RemoveAt(int index);

// Extra functions
  public int Capacity { get; set; }
  public PathSegment this[int index] { get; set; }
25 public int Add(PathSegment segment);
  public bool Contains(PathSegment segment);
  public int IndexOf(PathSegment segment);
  public int IndexOf(PathSegment segment, int startIndex);
  public int IndexOf(PathSegment segment, int startIndex, int count);
30 public int LastIndexOf(PathSegment segment);
  public int LastIndexOf(PathSegment segment, int startIndex);
  public int LastIndexOf(PathSegment segment, int startIndex, int count);

```

SA

```

5  public void Insert(int index, PathSegment segment);
    public void Remove(PathSegment segment);
    public void AddRange(PathSegmentCollection segmentCollection);
    public void SetRange(int index, PathSegmentCollection segmentCollection);
    public void InsertRange(int index, PathSegmentCollection segmentCollection);
    public void RemoveRange(int index, int count);
    public PathSegmentCollection GetRange(int index, int count);

```

10 A PathSegment represents a section of the outline of a PathFigure. Simple straight-line segments, elliptical-arc segments, cubic bezier segments, and quadratic bezier segments can be combined together to form a PathFigure.

```

15 public abstract class PathSegment : Animatable
    {
        internal PathSegment();
        public new PathSegment Copy();

        [DefaultValue(true)]
        public bool IsStroked { get; set; }
20  }

    public sealed class StartSegment : PathSegment
    {
        public StartSegment();
        public StartSegment(Point point);
        public StartSegment(Point point, PointAnimationCollection pointAnimations);
        public new StartSegment Copy();

        [Animations("PointAnimations")]
        public Point Point { get; set; }
        public PointAnimationCollection PointAnimations { get; set; }
30  }

```

text

55

```

public sealed class CloseSegment : PathSegment
{
    public CloseSegment(bool isStroked);
5    public new CloseSegment Copy();
}

public sealed class LineSegment : PathSegment
{
10    public LineSegment();
    public LineSegment(Point point, bool isStroked);
    public LineSegment(
        Point point,
        PointAnimationCollection pointAnimations,
15    bool isStroked);
    public new LineSegment Copy();

    [Animations("PointAnimations")]
    public Point Point { get; set; }
20    public PointAnimationCollection PointAnimations { get; set; }
}

public sealed class BezierSegment : PathSegment
{
25    public BezierSegment();
    public BezierSegment(
        Point point1,
        Point point2,
        Point point3,
30    bool isStroked); --
    public BezierSegment(
        Point point1, PointAnimationCollection point1Animations,

```

36

```

    Point point2, PointAnimationCollection point2Animations,
    Point point3, PointAnimationCollection point3Animations,
    bool isStroked);
public new BezierSegment Copy();

5
    [Animations("Point1Animations")]
    public Point Point1 { get; set; }
    public PointAnimationCollection Point1Animations { get; set; }

10
    [Animations("Point2Animations")]
    public Point Point2 { get; set; }
    public PointAnimationCollection Point2Animations { get; set; }

    [Animations("Point3Animations")]
15
    public Point Point3 { get; set; }
    public PointAnimationCollection Point3Animations { get; set; }
}

public sealed class QuadraticBezierSegment : PathSegment
20
{
    public QuadraticBezierSegment();
    public QuadraticBezierSegment(
        Point point1,
        Point point2,
25
        bool isStroked);
    public QuadraticBezierSegment(
        Point point1, PointAnimationCollection point1Animations,
        Point point2, PointAnimationCollection point2Animations,
        bool isStroked);
30
    public new QuadraticBezierSegment Copy();

    [Animations("Point1Animations")]

```

```

public Point Point1 { get; set; }
public PointAnimationCollection Point1 Animations { get; set; }

[Animations("Point2Animations")]
5 public Point Point2 { get; set; }
public PointAnimationCollection Point2Animations { get; set; }
}

public sealed class PolyLineSegment : PathSegment
10 {
    public PolyLineSegment();
    public PolyLineSegment(Point[] points, bool isStroked);
    public PolyLineSegment(ICollection points, bool isStroked);
    public new PolyLineSegment Copy();
15
    public void AddPoint(Point point);
    public PointCollection Points { get; set; }
}

20 public sealed class PolyBezierSegment : PathSegment
{
    public PolyBezierSegment();
    public PolyBezierSegment(Point[] points, bool isStroked);
    public PolyBezierSegment(ICollection points, bool isStroked);
25 public new PolyBezierSegment Copy();

    public void AddPoint(Point point);
    public PointCollection Points { get; set; }
}
30
public sealed class PolyQuadraticBezierSegment : PathSegment
{

```

58

```

public PolyQuadraticBezierSegment(Point[] points, bool isStroked);
public PolyQuadraticBezierSegment(ICollection points, bool isStroked);
public new PolyQuadraticBezierSegment Copy();

5   public void AddPoint(Point point);
    public PointCollection Points { get; set; }
    }

10  public sealed class ArcSegment : PathSegment
    {
        public ArcSegment();
        public ArcSegment(
            Point point,
            Size arcSize,
15    double xRotation,
            bool largeArcFlag,
            bool sweepFlag,
            bool isStroked);
        public ArcSegment(
20    Point point,
            PointAnimationCollection pointAnimations,
            Size arcSize,
            SizeAnimationCollection arcSizeAnimations,
            double xRotation,
25    DoubleAnimationCollection xRotationAnimations,
            bool largeArcFlag,
            bool sweepFlag,
            bool isStroked);
        public new ArcSegment Copy();

30    [Animations("PointAnimations")]
        public Point Point { get; set; }
    }

```

```

public PointAnimationCollection PointAnimations { get; set; }

[Animations("SizeAnimations")]
public Size Size { get; set; }
5 public SizeAnimationCollection SizeAnimations { get; set; }

[Animations("XRotationAnimations")]
public double XRotation { get; set; }
10 public DoubleAnimationCollection XRotationAnimations { get; set; }

public bool LargeArc { get; set; }
public bool SweepFlag { get; set; }
}

```

15 Each of the PathSegment objects that are added to a PathFigure also have the PathSegment.IsStroke property. If a PathSegment has this property set to true, then the particular PathSegment will contribute to the stroked area of the PathGeometry when rendering with a Pen. This also applies to hit-testing and to an explicit Widen of the PathGeometry. The particular behavior when switching between stroked and non-

20 stroked PathSegment sections in a PathFigure is the same as if the Pen specified dashed, in that the appropriate dash caps will be applied at the PathSegment ends.

The following sample demonstrates markup that uses a PathGeometry, PathFigure, and PathSegments:

```

25 <Path Fill="#4000FF00" Stroke="Yellow" >
    <Path.Data>
        <PathGeometry Transform="translate(225 25)" FillRule="EvenOdd">
            <PathGeometry.Figures>
                <PathFigureCollection>
                    <PathFigure IsFilled="True">
30             <PathFigure.Segments>
                <PathSegmentCollection>

```

167


```

5      <StartSegment Point="0 0" IsStroked="False" />
      <LineSegment Point="100 0" />
      <BezierSegment Point1="125 25" Point2="125 75"
Point3="100 100" />
      <QuadraticBezierSegment Point1="50 50" Point2="0 100" />
      <ArcSegment Point="100 150" Size="100 100"
XRotation="45" LargeArc="False" SweepFlag="True" />
      <PolyLineSegment Points="100 175 0 175" />
      <PolyBezierSegment Points="50 225 50 275 0 300 50 325 50
10    375 0 400" />
      <PolyQuadraticBezierSegment Points="50 450 0 500 50 550 0
600" />
      <CloseSegment IsStroked="True" />
      </PathSegmentCollection>
      </PathFigure.Segments>
15    </PathFigure>
      </PathFigureCollection>
      </PathGeometry.Figures>
      </PathGeometry>
20    </Path.Data>
    </Path>

```

A RectangleGeometry defines a rectangle or rounded corner rectangle geometry object. The radius X and radius Y refer to the axis aligned radial length of the rounded corners:

```

25    public sealed class RectangleGeometry : Geometry
    {
        public RectangleGeometry();
        public RectangleGeometry(Rect rect);
30    public RectangleGeometry(
        Rect rect,
        double radiusX,

```

```

    double radiusY);
    public RectangleGeometry(
        Rect rect, RectAnimationCollection rectAnimations,
        double radiusX, DoubleAnimationCollection radiusXAnimations,
5      double radiusY, DoubleAnimationCollection radiusYAnimations);
    public RectangleGeometry(
        Rect rect, RectAnimationCollection rectAnimations,
        double radiusX, DoubleAnimationCollection radiusXAnimations,
        double radiusY, DoubleAnimationCollection radiusYAnimations,
10      Transform);
    public new RectangleGeometry Copy();

    [Animation("RectAnimations")]
    public Rect Rect { get; set; }
15    public RectAnimationCollection RectAnimations { get; set; }

    [Animation("RadiusXAnimations")]
    [DefaultValue(0.0)]
    public double RadiusX { get; set; }
20    public DoubleAnimationCollection RadiusXAnimations { get; set; }

    [Animation("RadiusYAnimations")]
    [DefaultValue(0.0)]
    public double RadiusY { get; set; }
25    public DoubleAnimationCollection RadiusYAnimations { get; set; }
    }

```

The following sample demonstrates markup that uses a RectangleGeometry:

```

30  <Path Fill="#4000FF00">
    <Path.Data>
        <RectangleGeometry Rect="125 125 75 75" RadiusX="10"
        RadiusY="5"></RectangleGeometry>
    </Path.Data>
</Path>

```

```
</Path.Data>
</Path>
```

5 An EllipseGeometry defines an elliptical area given axis aligned radial X and Y lengths:

```
public sealed class EllipseGeometry : Geometry
{
    public EllipseGeometry();
    public EllipseGeometry(Rectangle rect);
10    public EllipseGeometry(
        Point center,
        double radiusX,
        double radiusY);
    public EllipseGeometry(
15       Point center, PointAnimationCollection centerAnimations,
        double radiusX, DoubleAnimationCollection radiusXAnimations,
        double radiusY, DoubleAnimationCollection radiusYAnimations);
    public EllipseGeometry(
20       Point center, PointAnimationCollection centerAnimations,
        double radiusX, DoubleAnimationCollection radiusXAnimations,
        double radiusY, DoubleAnimationCollection radiusYAnimations,
        Transform transform);
    public new EllipseGeometry Copy();

25    [Animation("CenterAnimations")]
    public Point Center { get; set; }
    public PointAnimationCollection CenterAnimations { get; set; }

    [Animation("RadiusXAnimations")]
30    public double RadiusX { get; set; }
    public DoubleAnimationCollection RadiusXAnimations { get; set; }
```

```
[Animation("RadiusYAnimations")]
public double RadiusY { get; set; }
public DoubleAnimationCollection RadiusYAnimations { get; set; }
}
```

5

The following sample demonstrates markup that uses an EllipseGeometry:

```
<Path Fill="#4000FF00">
  <Path.Data>
    <EllipseGeometry Center="50 300" RadiusX="50"
      RadiusY="75"></EllipseGeometry>
  </Path.Data>
</Path>
```

10

A LineGeometry defines a line segment between two points and therefore contains no fill area:

15

```
public sealed class LineGeometry : Geometry
{
  public LineGeometry();
  public LineGeometry(Point pt1, Point pt2);
  public LineGeometry(
    Point startPoint,
    PointAnimationCollection startPointAnimations,
    Point endPoint,
    PointAnimationCollection endPointAnimations);
  public LineGeometry(
    Point startPoint,
    PointAnimationCollection startPointAnimations,
    Point endPoint,
    PointAnimationCollection endPointAnimations,
    Transform transform);
  public new LineGeometry Copy();
}
```

20

25

30

7/7/2

```

5  [Animation("Point1Animations")]
    public Point StartPoint { get; set; }
    public PointAnimationCollection StartPointAnimations { get; set; }

    [Animation("Point2Animations")]
    public Point Point2 { get; set; }
    public PointAnimationCollection Point2Animations { get; set; }
  }

```

10 The following sample demonstrates markup that uses a LineGeometry:

```

15  <Path Stroke="#4000FF00"
    StrokeThickness="20"
    StrokeStartLineCap="Flat"
    StrokeEndLineCap="Triangle"
    StrokeDashCap="Round"
    StrokeLineJoin="Bevel"
    StrokeMiterLimit="100"
    StrokeDashOffset="0"
20  StrokeDashArray="1.0 2.0">
    <Path.Data>
      <LineGeometry StartPoint="125 25" EndPoint="175 75"></LineGeometry>
    </Path.Data>
  </Path>
25

```

IMAGING

30 ImageSource is an abstract class, comprising a basic building block for the imaging pipeline. An ImageSource conceptually represents a single, constant set of pixels at a certain size and resolution. For example, an ImageSource may be a single frame in an image file that a Decoder could provide, or it may be the results of a transform that operates on a certain ImageSource of its own. An ImageSource is not

multiframe or animate. An ImageSource is changeable, not because its own properties can be changed, but because the properties of its sub-classes can potentially be changed.

- 5 For performance reasons, ImageSources support providing unmanaged access to the image, using the IMILBitmapSource interface. If the sub-class of ImageSource does not provide it, then the ImageSource base class does (using a wrapper class).

```
namespace System.Windows.Media
{
    public abstract class ImageSource : Changeable
    {
        /// Native format of the image's data.
        /// If the ImageSource is directly readable, this is the format the
        /// pixels will be in when they are read.
        public abstract PixelFormat Format { get; }

        /// Whether the ImageSource can convert its data to the specified format.
        /// If not, a format converter could be used to do the conversion.
        /// Note: for best performance, ImageSources should provide support for
        /// PixelFormat32bppPARGB.
        public virtual bool CanConvertTo(PixelFormat targetPixelFormat)

        /// Width, in pixels, of the image.
        public abstract int PixelWidth { get; }

        /// Height, in pixels, of the image.
        public abstract int PixelHeight { get; }

        /// Horizontal DPI of the image.
        public abstract double DpiX { get; }

        /// Vertical DPI of the image.
        public abstract double DpiY { get; }
```

/// Get the width of the image in measure units (96ths of an inch).

public double Width { get; }

/// Get the width of the image in measure units (96ths of an inch).

public double Height { get; }

/// Get the Palette for the image, if there is one.

ImagePalette Palette { get; }

/// Copy the pixel data from the image into the array of pixels that has the

/// specified stride, starting at the pixelOffset (specified in number of

/// pixels from the beginning). The pixels should be copied into

/// the specified pixelFormat. To find out if the pixelFormat is supported,

/// call CanConvertTo first.

**public void Copy(PixelFormat pixelFormat, byte[] pixels,
int stride, int pixelOffset)**

/// Copy a rect of pixel data from the image into the array of pixels that has

/// the specified stride, starting at the pixelOffset (specified in number of

/// pixels from the beginning). The pixels should be copied into

/// the specified pixelFormat. To find out if the pixelFormat is supported,

/// call CanConvertTo first. An empty rect (one with a width and/or height

/// of 0), means to ignore the rect and copy the entire image.

**public abstract void Copy(IntegerRect sourceRect, PixelFormat pixelFormat,
byte[] pixels, int stride, int pixelOffset);**

/// If there is an embedded thumbnail, return it.

/// Otherwise, return null. This method does NOT create a

/// thumbnail for images that don't already have one.

public virtual ImageSource EmbeddedThumbnail { get; set; }

/// Get a thumbnail of the image.

776

```

/// This method always returns a thumbnail that is in approximately the
/// same aspect ratio as the original image.
///
/// If there is an embedded thumbnail, this method returns that
/// thumbnail, scaled to the specified size. If there is not an embedded
/// thumbnail, the image itself is scaled to the specified size.
///
/// The specified width and height of the thumbnail must both be > 0.
/// We choose whether to use the specified width value or the specified
/// height value by determining which will give the largest possible
/// thumbnail, while maintaining the aspect ratio and staying less than
/// or equal to both the width and height values.
///
/// Therefore, to specify that you wish to use only the width value, you
/// would use Int32.MaxValue for the height value.
public virtual ImageSource GetThumbnail(ImageSizeOptions sizeOptions);

/// Provides access to this image's metadata.
public virtual ImageMetadata Metadata { get; set; }

        /// Returns an IMILBitmapSource for this ImageSource.
        unsafe internal IntPtr BitmapSource { get; }
    }
}

```

ImageData is a sub-class of **ImageSource**. **ImageData** implements **ImageSource** for several different sources of images:

```

HICON
HBITMAP
Pixel data in managed memory.
Pixel data in unmanaged memory
Images in a System.IO.Stream (requiring a decoder)

```


Another ImageSource

5 **ImageData** provides services, including caching the decoded image in system
 memory, cropping the image to a specified source rect (to the cache) and sizing the
 image to a specified decode width and height (to the cache). For image decoding
 10 scenarios, **ImageData** enables the specification of which decoder to use or automatic
 codec discovery, based on the input stream and the mime type. **ImageData** does not
 support loading an image directly from a URI. The loader should be used to map the
 URI to a stream that can be used to construct an **ImageData**. Once an **ImageData** is
 constructed, its only changeable properties are its embedded thumbnail, its metadata,
 15 and its pixel data. The other properties are considered immutable.

The pixel data of the **ImageData** can be changed, including in one of two ways:
 (1) obtaining a **Drawing Context** for the **ImageData** and issuing commands via the
Drawing Context to draw onto the image, or (2) using the **ImageData** as a destination
 (**RenderTarget**) for a **VisualManager** and issuing a command to render a visual tree (a
 15 scene) to the **ImageData**. In either case, the drawing is done to an image in memory,
 which means the pixel data first needs to be decoded and cached in memory. Only the
 cached memory image is changed – the image file itself is not affected (unless the
ImageData is later encoded to the image file using an **ImageEncoder**).

```
namespace System.Windows.Media
```

```
{
    /// ImageData can be constructed with or without a cache.
    /// If constructed without a cache, the image will be decoded every time it is
    /// used. The advantage of this approach is that the cache is not stored in
    /// memory and that the decoding and filtering can be optimal for the size that
    /// it is being drawn. The disadvantage is that the decoding has to be done with
    /// every redraw. The other approach is to cache the decoded image. There are
    /// several options for creating the decoded, cached image. It can be created at
    /// a specific size and with a specific source rect (area of interest from the
    /// source image). There are no options if the image is not cached – it is
    /// decoded with no source cropping. If the image is cached, the cached version
```

**/// of the image becomes the source image, and the original image source will
/// be discarded.**

public class ImageData : ImageSource

{

/// Construct an ImageData from a Stream.

/// The caller owns the stream and is responsible for closing it.

public ImageData(System.IO.Stream imageStream);

/// Construct an ImageData from a Stream.

/// The caller owns the stream and is responsible for closing it.

public ImageData(System.IO.Stream imageStream,

CodecInfo decoderInfo, // identifies the codec to be used (or null)

bool createCache, // if false, sourceRect and sizeOptions are ignored

IntegerRect sourceRect,

ImageSizeOptions sizeOptions

);

/// Construct an ImageData from an array of pixels.

unsafe public ImageData(

int pixelWidth,

int pixelHeight,

double dpiX,

double dpiY,

PixelFormat pixelFormat,

ImagePalette imagePalette,

byte[] pixels,

int stride,

IntegerRect sourceRect,

ImageSizeOptions sizeOptions

);

/// Construct an ImageData from an array of pixels in unmanaged memory

/// (e.g. a DibSection).

2

```
public ImageData(  
    int pixelWidth,  
    int pixelHeight,  
    double dpiX,  
    double dpiY,  
    PixelFormat pixelFormat,  
    ImagePalette imagePalette,  
    IntPtr pixels, // unmanaged array of pixels, e.g. a DibSection  
    int stride,  
    bool createCache, // if false, sourceRect and sizeOptions are ignored  
    IntegerRect sourceRect,  
    ImageSizeOptions sizeOptions  
);
```

/// Construct an ImageData from a HBITMAP.

```
public ImageData(  
    HBITMAP hbitmap,  
    HPALETTE hpalette,  
    IntegerRect sourceRect,  
    ImageSizeOptions sizeOptions  
);
```

/// Construct an ImageData from a HICON.

```
public ImageData(  
    HICON hicon,  
    IntegerRect sourceRect,  
    ImageSizeOptions sizeOptions  
);
```

/// Construct an ImageData from an ImageSource.

```
public ImageData(  
    ImageSource imageSource,
```

71

```

    bool createCache, // if false, sourceRect and sizeOptions are ignored
    IntegerRect sourceRect,
    ImageSizeOptions sizeOptions
);
/// Get the information about the specific codec that was used
/// to decode the image (if there a codec was required and we
/// have that information).
public CodecInfo DecoderInfo { get; }
/// Get a DrawingContext to draw on the ImageData.
public DrawingContext Open();
}
}

```

ImageDecoder is an abstract class that provides the base class for decoders. It provides a way to determine how many frames are in the image and to enumerate (or index) the frames. As mentioned above, image frames are each an ImageSource.

- 5 Built-in codecs create an ImageData object for each requested frame. Add-in codecs may use a different sub-class to return an ImageSource for each frame. ImageDecoder is not an ImageSource itself, but rather is a container for one or more ImageSources. Note that each frame of the image could potentially have different attributes (different size, different resolution, etc.).

```

namespace System.Windows.Media
{
    /// ImageDecoder is essentially a container for image frames.
    /// Each image frame is an ImageSource.
    public abstract class ImageDecoder : ICollection, IEnumerable
    {
        /// The number of image frames in this image.
        public int Count { get; }

        /// Whether access to the ICollection is synchronized (thread-safe).
        public bool IsSynchronized { get; }
    }
}

```

72

```
/// Gets an object that can be used to synchronize access to the ICollection.  
public object SyncRoot { get; }
```

```
/// Copies the frames to an Array, starting at a particular Array index.  
public void CopyTo(Array array, int index);
```

```
/// Returns an enumerator to iterate through the frames of the image.  
public IEnumerator GetEnumerator();
```

```
/// Returns a codec-specific object that identifies the properties  
/// that were used to encode this image (if supported by the codec).  
/// These codec-specific properties can be passed back to the associated  
/// encoder (if there is one) to get the same type of encoding again.  
public virtual object CustomEncoderProperties { get; }
```

```
/// The info that identifies this codec (including any associated encoder).  
public abstract CodecInfo Info { get; }
```

```
/// Get the pixel format for the first frame.  
public virtual System.Windows.Media.PixelFormat Format { get; }
```

```
/// Get the pixel width for the first frame.  
public virtual int PixelWidth { get; }
```

```
/// Get the pixel Height for the first frame.  
public virtual int PixelHeight { get; }
```

```
/// Get the horizontal dpi for the first frame.  
public virtual double DpiX { get; }
```

```
/// Get the vertical dpi for the first frame.
```

```
public virtual double DpiY { get; }
```

```
/// Get the width of the image in measure units (96ths of an inch).
```

```
public double Width { get; }
```

```
/// Get the height of the image in measure units (96ths of an inch).
```

```
public double Height { get; }
```

```
/// If there is an embedded thumbnail for the first frame, return it.
```

```
/// Otherwise, return null. This method does NOT create a
```

```
/// thumbnail for images that don't already have one.
```

```
public virtual ImageSource EmbeddedThumbnail { get; }
```

```
/// Return an image frame, cropped by the specified sourceRect.
```

```
/// An empty rect (one with a width and/or height of 0),
```

```
/// means to ignore the rect and get the entire frame.
```

```
public virtual ImageSource GetFrame(
```

```
    int frameIndex,
```

```
    bool createCache,
```

```
    IntegerRect sourceRect);
```

```
/// Return an image frame, with all the specified cropping and sizing applied.
```

```
/// An empty source rect (one with a width and/or height of 0),
```

```
/// means to ignore the rect and get the entire frame.
```

```
public virtual ImageSource GetFrame(
```

```
    int frameIndex,
```

```
    bool createCache,
```

```
    IntegerRect sourceRect,
```

```
    ImageSizeOptions sizeOptions);
```

```
/// Indexer for returning a specific frame of the image (at full size).
```

```
/// The index must be in the range: (NumFrames > index >= 0)
```

74

```

public abstract ImageSource this[int frameIndex] { get; }

    /// Conversion from ImageDecoder to ImageSource by returning the first frame.
    public static explicit operator ImageSource(ImageDecoder decoder);

    /// Provides read-only access to this image's metadata.
    public abstract ImageMetaData MetaData { get; }
}

```

A number of built-in decoders are provided with the MIL, including ImageDecoderBmp.cs, ImageDecoderGif.cs, ImageDecoderIcon.cs, ImageDecoderJpeg.cs, ImageDecoderPng.cs, ImageDecoderTiff.cs, and ImageDecoderWmp.cs. Each implement ImageDecoder and a single constructor which uses a System.IO.Stream to initialize the decoder, as in the following example:

10

```

    /// If this decoder cannot handle the image stream, it will
    /// throw an exception.
    public ImageDecoderJpeg(System.IO.Stream imageStream);

```

15

An ImageEncoder is a collection of ImageSources (image frames), each potentially with its own metadata and thumbnail. There can also be a global thumbnail and metadata associated with the entire set of frames. A codec may also choose to provide support for encoding properties, used to determine how to encode the image. The collection of frames can be saved (encoded) to any number of specified streams (one at a time). The collection can be cleared and then filled with a different collection and saved again.

```

namespace System.Windows.Media
{
    /// ImageEncoder collects a set of frames (ImageSource's) with their associated
    /// thumbnails and metadata and saves them to a specified stream. In addition
    /// to frame-specific thumbnails and metadata, there can also be an image-wide

```

704

25

```

/// (global) thumbnail and metadata, if the codec supports it.
public abstract class ImageEncoder : IList, IDisposable
{
    public void Dispose();

    /// Gets a value indicating whether the IList has a fixed size.
    public bool IsFixedSize { get; }

    /// Gets a value indicating whether the IList is read-only.
    public bool IsReadOnly { get { return false; } }

    /// Indexer for a specific frame of the image. The set method is
    /// a Replace operation for that frame. The object must be an ImageSource.
    public object IList.this[int frameIndex] { get; set; }
    public ImageSource this[int frameIndex] { get; set; }

    /// Add a frame to the image.
    public int IList.Add(object value);
    public int Add(ImageSource imageFrame);

    /// Reset the encoder to an empty state, with no frames,
    /// thumbnails, metadata, or encoding properties.
    public void Clear();

    /// Determines whether the Encoder contains a specific frame.
    public bool IList.Contains(object value);
    public bool Contains(ImageSource imageFrame);

    /// Determines the index of the specified frame.
    public int IList.IndexOf(object value);
    public int IndexOf(ImageSource imageFrame);

```


26

```

/// This method inserts the frame at specified frame location, with its
/// associated thumbnail and metadata (if any).
public void IList.Insert(int frameIndex, object value);
public void Insert(int frameIndex, ImageSource imageFrame);

/// Removes the first occurrence of the specified frame.
public void IList.Remove(object value);
public void Remove(ImageSource imageFrame);

/// Remove the specified frame from the image. Subsequent frames
/// are moved up in the list.
public void RemoveAt(int frameIndex);

/// The number of image frames in this image.
public int Count { get; }

/// Whether access to the ICollection is synchronized (thread-safe).
public bool IsSynchronized { get; }

/// Gets an object that can be used to synchronize access to the ICollection.
public object SyncRoot { get; }

/// Copies the frames to an Array, starting at a particular Array index.
public void CopyTo(Array array, int index);

/// Returns an enumerator to iterate through the frames of the image.
public IEnumerator GetEnumerator();

/// The info that identifies this codec (including any associated decoder).
public CodecInfo-Info { get; }

/// A codec-specific object that identifies the properties

```

26

```

/// used to encode this image.
public object CustomEncoderProperties { get; set; }

/// Set or get the image's global embedded thumbnail, if there is one.
public ImageSource EmbeddedThumbnail { get; set; }

/// Provides access to this image's global metadata.
public ImageMetadata Metadata { get; set; }

/// Whether this codec supports images with more than one frame.
public bool SupportsMultiFrame { get; }

/// Copy the image from the specified decoder, including all the frames,
/// all the thumbnails, and all the metadata, including the per-frame
/// and global data. This does an implicit clear of any data that's already
/// been set in the encoder.
public void CopyImage(ImageDecoder decoder);

/// Save (encode) the image to the specified stream.
public void Save(System.IO.Stream destStream);
}
}

```

A number of built-in encoders are provided with the MIL, including ImageEncoderBmp.cs, ImageEncoderGif.cs, ImageEncoderJpeg.cs, ImageEncoderPng.cs, ImageEncoderTiff.cs, and ImageEncoderWmp.cs.

5 ImageSizeOptions are used to specify the size of thumbnails and the size of cached images. The options include the width, height, whether to preserve the aspect ratio of the original image, and a rotation angle (multiple of 90 degrees).

10

```

namespace System.Windows.Media
{
/// Sizing options for an image. The resulting image

```

10

```

/// will be scaled based on these options.
public class ImageSizeOptions
{
    /// The rotation to be applied; only multiples of 90 degrees is supported.
5   public enum Rotation
    {
        /// Do not rotate
        Rotate0 = 0,

10        /// Rotate 90 degrees
        Rotate90 = 90,

        /// Rotate 180 degrees
        Rotate180 = 180,

15        /// Rotate 270 degrees
        Rotate270 = 270,
    }

20    /// Construct an ImageSizeOptions object.
    /// Still need to set the Width and Height Properties.
    public ImageSizeOptions();

    /// Construct an ImageSizeOptions object.
25    public ImageSizeOptions(
        bool preserveAspectRatio,
        int pixelWidth,
        int pixelHeight
    );

30    /// Whether or not to preserve the aspect ratio of the original
    /// image. If so, then the PixelWidth and PixelHeight are

```

```

5  /// maximum values for the image size. The resulting image
    /// is only guaranteed to have either its width or its height
    /// match the specified values. For example, if you want to
    /// specify the height, while preserving the aspect ratio for
    /// the width, then set the height to the desired value, and
    /// set the width to Int32.MaxValue.
    ///
    /// If we are not to preserve aspect ratio, then both the
    /// specified width and the specified height are used, and
10  /// the image will be stretched to fit both those values.
    public bool PreserveAspectRatio { get; set; }

    /// PixelWidth of the resulting image. See description of
    /// PreserveAspectRatio for how this value is used.
15  ///
    /// PixelWidth must be set to a value greater than zero to be valid.
    public int PixelWidth { get; set; }

    /// PixelHeight of the resulting image. See description of
    /// PreserveAspectRatio for how this value is used.
20  ///
    /// PixelHeight must be set to a value greater than zero to be valid.
    public int PixelHeight { get; set; }

    /// RotationAngle to rotate the image. Only multiples of 90 are supported.
25  public Rotation RotationAngle { get; set; }

    /// Whether the size options are valid. To be valid,
    /// both must be greater than zero, and at most one
30  /// can be set to Int32.MaxValue.
    public bool IsValid { get; }
}

```

80

```
}
```

The following provides a Pixel Format Definition for images and pixel-based surfaces:

```
namespace System.Windows.Media
{
    /// Pixel Format Definition for images and pixel-based surfaces
    public struct PixelFormat
    {
        /// Describes order of each channel of pixel data
        public enum ChannelDescription
        {
            /// Undefined channel description
            Undefined = 0,

            /// single channel indexed (palletized) formats
            Index = 1,

            /// single channel gray formats
            Gray = 2,

            /// red-green-blue
            RGB = 3,

            /// blue-green-red
            BGR = 4,

            /// alpha-red-green-blue
            ARGB = 5,

            -

            /// Cyan-magenta-yellow-black
        }
    }
}
```

490

```

    CMYK = 6
}

/// op_equality - returns whether or not the two pixel formats are equal
public static bool operator == (PixelFormat left, PixelFormat right);

/// Equals - Returns whether or not the two pixel formats are equal
public static bool Equals(PixelFormat left, PixelFormat right);

/// op_inequality - returns whether or not the two pixel formats are not equal
public static bool operator != (PixelFormat left, PixelFormat right);

/// Equals - Returns whether or not this is equal to the Object
public override bool Equals(Object obj);

/// GetHashCode - Returns a hash code
public override int GetHashCode();

/// The number of bits per pixel for this format.
public int BitsPerPixel { get; }

/// Whether this format is a palletized (indexed) format.
public bool IsPalletized { get; }

/// Whether this format is a set of gray values.
public bool IsGray { get; }

/// Whether this format is CMYK values (cyan, magenta, yellow, black)
public bool IsCMYK { get; }

/// Whether this format is SRGB (Gamma is approximately 2.2)
public bool IsSRGB { get; }

```

for

```

/// Whether this format is Linear (Gamma is 1.0)
public bool IsScRGB { get; }

/// Whether this format has premultiplied alpha.
public bool IsPremultiplied { get; }

/// Describes the data of this format and its order.
public ChannelDescription Channel { get; }

/// Get the mask to use for obtaining the red channel for this format.
/// Shift the mask by the leftShift amount to get the mask to use.
public void GetRedMask(out uint mask, out int leftShift);

/// Get the mask to use for obtaining the green channel for this format.
/// Shift the mask by the leftShift amount to get the mask to use.
public void GetGreenMask(out uint mask, out int leftShift);

/// Get the mask to use for obtaining the blue channel for this format.
/// Shift the mask by the leftShift amount to get the mask to use.
public void GetBlueMask(out uint mask, out int leftShift);

/// Get the mask to use for obtaining the alpha channel for this format.
/// Shift the mask by the leftShift amount to get the mask to use.
public void GetAlphaMask(out uint mask, out int leftShift);

/// Get the masks to use for obtaining each channel for this format.
/// Shift the masks by the leftShift amount to get the masks to use.
public void GetRGBMasks(out uint redMask, out int redLeftShift,
                        out uint greenMask, out int greenLeftShift,
                        out uint blueMask, out int blueLeftShift);

}

```

```
}

```

Each codec (ImageEncoder and ImageDecoder) is required to supply a CodecInfo that gives information about the codec, provides create instance methods for its associated decoder/encoder, and provides a method that determines if this codec matches a supplied CodecFilter.

```

namespace System.Windows.Media
{
    /// Information about a specific codec and a factory for creating
    /// the codec. This is returned from the codec enumerator.
10    public abstract class CodecInfo
    {
        /// The number of bytes needed from the image header to
        /// determine if the image is supported by this codec.
15        public abstract int RequiredHeaderSize { get; }

        /// Whether the codec supports this image, based on looking
        /// at the first RequiredHeaderSize bytes from the image. The header
        /// must contain at least the first RequiredHeaderSize bytes from the image.
20        public abstract bool IsImageSupported(byte[] header);

        /// The Friendly name of a codec
        public abstract string FriendlyName { get; }

        /// Which Mime Types the codec supports.
25        public abstract string[] MimeTypes { get; }

        /// Who authored the codec.
        public abstract string CodecAuthor { get; }

30        /// The version number of the codec.

```



```
public abstract System.Version CodecVersion { get; }

    /// Whether there is a decoder associated with this codec.
    public abstract bool HasDecoder { get; }

5    /// Whether there is an encoder associated with this codec.
    public abstract bool HasEncoder { get; }

    /// Whether this codec is one of the built-in ones.
10    public bool IsBuiltIn { get; }

    /// Determine if this codec matches the specified filter.
    /// Note: this does NOT check/sniff the bytes in the stream.
    public virtual bool MatchesFilter(CodecFilter filter);

15    /// Get an instance of the decoder associated with this codec (if there is one).
    public abstract ImageDecoder CreateDecoderInstance(System.IO.Stream
        imageStream);

    //Get an instance of the encoder associated with this codec (if there is one).
20    public abstract ImageEncoder CreateEncoderInstance();
}
}
```

25 Built-in CodecInfo objects are provided with the MIL, including
CodecInfoBmp.cs, CodecInfoGif.cs, CodecInfoIco.cs, CodecInfoJpeg.cs,
CodecInfoPng.cs, CodecInfoTiff.cs, and CodecInfoWmp.cs.

The CodecFilter is used by the codec enumerator to enumerate codecs based on
the specified criteria. Criteria that are not specified are ignored when looking for a
30 matching codec. For example, if the MimeType is not set, then codecs with any mime
type are considered.

namespace System.Windows.Media

77E

85

```

{
    /// Filter for enumerating codecs. Only those codecs that match the
    /// properties will be enumerated.
    public class CodecFilter
5      {
        /// Find a codec that can handle this image stream (by sniffing the stream).
        public System.IO.Stream ImageStream { get; set; }

        /// The Friendly name of a codec
10      public string FriendlyName { get; set; }

        /// Does the codec support this Mime Types.
        public string MimeType { get; set; }

15      /// Does the codec match this author.
        public string CodecAuthor { get; set; }

        /// Is the codec's version >= this version.
        public System.Version MinVersion { get; set; }
20      /// Is the codec's version <= this version.
        public System.Version MaxVersion { get; set; }

        /// Find those codecs that have a matching decoder. Setting this
        /// to false means that we don't filter on whether or not there is
        /// a decoder for that codec.
25      public bool HasDecoder { get; set; }

        /// Find those codecs that have a matching encoder. Setting this
        /// to false means that we don't filter on whether or not there is
        /// an encoder for that codec.
30      public bool HasEncoder { get; set; }
    }
}

```

Ab

```
    /// Find those codecs that are built-in (not add-ins). Setting this
    /// to false means that we don't filter on whether or not the
    /// codec is one of the built-in ones.
5    public bool IsBuiltIn { get; set; }
    }
}
```

When the enumerator is constructed (its ctors are internal), it is given a
10 **CodecFilter**. That filter is used to determine which codecs to enumerate. Only those
that match the filter (if any) are enumerated.

```
namespace System.Windows.Media
{
    /// The enumerator for Image frames.
15    public class ImageCodecEnumerator : IEnumerator
    {
        /// Reset - resets the position to before the first object in the collection.
        /// A call to MoveNext must precede any call to Current after a Reset.
        public void Reset();

20        /// MoveNext - Move to the next object in the collection. Returns false if the
        enumerator
        /// has passed the end of the collection
        public bool MoveNext();

25        /// Current - returns the current object in the collection
        public object Current { get; }
    }
}
}
```

30

82

ImageEffect is the base class for raster-based imaging effects. An ImageEffect can be viewed as a collection of 0 or more inputs and 0 or more outputs. The inputs and outputs to an ImageEffect are all of type ImageSource. An ImageEffect is typically initialized with its inputs and its properties and then its outputs are used either to draw part of a scene or as frames for an ImageEncoder. Built-in effects include (but are not limited to) the following: ImageEffectBlur, ImageEffectFlipRotate, ImageEffectGammaCorrect, ImageEffectGlow, ImageEffectGrayscale, ImageEffectNegate, ImageEffectSharpen, ImageEffectTint.

10 TRANSFORM

The Transform class of objects, represented in FIG. 7, can be used for scaling, rotating, translating and skewing vector and raster graphics. The derived Transform classes provide friendly usage and enumeration semantics. The transform class hierarchy differs from the Matrix struct by being a class and supporting animation and enumeration semantics:

TransformCollection (enumerating semantics, etc.)

TransformCollection.AddScale(...)

Animate MatrixTransform

Transform is the abstract base class:

```
[TypeConverter(typeof(TransformConverter))]  
public abstract class Transform : Animatable, IDisposable  
{  
    internal Transform();  
    public new Transform Copy();  
  
    public static MatrixTransform CreateMatrix(Matrix matrix);  
    public static TranslateTransform CreateTranslation(double x, double y);  
    public static RotateTransform CreateRotation(double angle);  
    public static RotateTransform CreateRotation(double angle, Point center);  
    public static ScaleTransform CreateScale(double scaleX, double scaleY);  
    public static ScaleTransform CreateScale(  
        double scaleX,
```

197

```
double scaleY,  
Point center);  
public static SkewTransform CreateSkew(double angleX, double angleY);  
public static SkewTransform CreateSkew(  
5 double angleX,  
double angleY,  
Point center);  
  
// Identity transformation  
10 public static Transform Identity { get; }  
  
public abstract Matrix Value { get; }  
}
```

15 A TransformCollection is a collection of Transform objects whose value is the matrix multiplication of the individual Transform values. The value is composed left to right, matching the first and last items of the collection:

```
public sealed class TransformCollection : Transform, IAddChild, IList  
{  
    public TransformCollection();  
    public TransformCollection(int capacity);  
    public TransformCollection(params Transform[] transforms);  
    public TransformCollection(ICollection transformCollection);  
    public new TransformCollection Copy();  
  
    // IEnumerable  
    public IEnumerator GetEnumerator();  
  
    // ICollection  
    public int Count { get; }  
    public bool IsSynchronized { get; }
```

public object SyncRoot(); public void CopyTo(Array array, int index); // IList public bool IsFixedSize { get; } bool IList.IsReadOnly { get; } object IList.this[int index] { get; set; } int IList.Add(object value); public void Clear(); bool IList.Contains(object value); int IList.IndexOf(object item); void IList.Insert(int index, object value); void IList.Remove(object value); public void RemoveAt(int index); // Extra functions
public int Capacity { get; set; }
public Transform this[int index] { get; set; }
public int Add(Transform transform);
public bool Contains(Transform transform); public int IndexOf(Transform transform); public int IndexOf(Transform transform, int startIndex); public int IndexOf(Transform transform, int startIndex, int count); public int LastIndexOf(Transform transform); public int LastIndexOf(Transform transform, int startIndex); public int LastIndexOf(Transform transform, int startIndex, int count); public void Insert(int index, Transform transform); public void Remove(Transform transform); public void AddRange(TransformCollection transformCollection); public void SetRange(int index, TransformCollection transformCollection); public void InsertRange(int index, TransformCollection transformCollection);

```

public void RemoveRange(int index, int count);
public TransformCollection GetRange(int index, int count);

// IAddChild
void IAddChild.AddChild(Object o);
void IAddChild.AddText(string s);

public override Matrix Value { get; }

public void AddMatrix(
    double m11,
    double m12,
    double m21,
    double m22,
    double offsetX,
    double offsetY);
public void AddMatrix(Matrix matrix);
public void AddTranslate(double translateX, double translateY);
public void AddRotate(double angle);
public void AddRotate(double angle, Point center);
public void AddScale(double scaleX, double scaleY);
public void AddScale(double scaleX, double scaleY, Point center);
public void AddSkew(double angleX, double angleY);
public void AddSkew(double angleX, double angleY, Point center);

public Transform GetOptimizedTransform();
}

```

A RotateTransform defines a rotation by an angle about a specific center point (the default being 0,0.)~The angle is specified in degrees. A static matrix representation for rotation angle about point x,y is as follows.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{bmatrix} * \begin{bmatrix} \cos(\text{angle}) & \sin(\text{angle}) & 0 \\ -\sin(\text{angle}) & \cos(\text{angle}) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{bmatrix}$$

```

1 public sealed class RotateTransform : Transform
2 {
3     public RotateTransform();
4     public RotateTransform(double angle);
5     public RotateTransform(
6         double angle,
7         Point center);
8     public RotateTransform(
9         double angle,
10        DoubleAnimationCollection angle,
11        Point center,
12        PointAnimationCollection center);
13     public new RotateTransform Copy();
14
15     [Animations("AngleAnimations")]
16     public double Angle { get; set; }
17     public DoubleAnimationCollection AngleAnimations { get; set; }
18
19     [Animations("CenterAnimations")]
20     public Point Center { get; set; }
21     public PointAnimationCollection CenterAnimations { get; set; }
22
23     public override Matrix Value { get; }
24 }

```

TranslateTransform defines an axis aligned translation in the x and y direction.
 The static matrix representation for translation by offset dx,dy is as follows.

net

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{bmatrix}$$

```

public sealed class System.Windows.Media.TranslateTransform : Transform
{
5   public TranslateTransform();
   public TranslateTransform(
       double offsetx,
       double offsety);
   public TranslateTransform(
10  double offsetx, DoubleAnimationCollection offsetx,
       double offsety, DoubleAnimationCollection offsety);
   public new TranslateTransform Copy();

   [Animations("XAnimations")]
15  public double X { get; set; }
   public DoubleAnimationCollection XAnimations { get; set; }

   [Animations("YAnimations")]
   public double Y { get; set; }
20  public DoubleAnimationCollection YAnimations { get; set; }

   public override Matrix Value { get; }
}

```

25 **ScaleTransform** defines a scale in the x and y direction about a center point (the default being 0,0.) The static matrix representation for scale **sx**, **sy** about the point **x,y** is as follows.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{bmatrix} * \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{bmatrix}$$

```

public sealed class ScaleTransform : Transform
{
5   public ScaleTransform();
    public ScaleTransform(
        double scaleX,
        double scaleY);
    public ScaleTransform(
10   double scaleX,
        double scaleY,
        Point center);
    public ScaleTransform(
        double scaleX,
15   DoubleAnimationCollection scaleXAnimations,
        double scaleY,
        DoubleAnimationCollection scaleYAnimations,
        Point center,
        PointAnimationCollection center);
20   public new ScaleTransform Copy();

    [Animations("XAnimations")]
    public double ScaleX { get; set; }
    public DoubleAnimationCollection ScaleXAnimations { get; set; }
25   [Animations("YAnimations")]
    public double ScaleY { get; set; }
    public DoubleAnimationCollection ScaleYAnimations { get; set; }

30   [Animations("CenterAnimations")]

```

94

```

    public Point Center { get; set; }
    public PointAnimationCollection CenterAnimations { get; set; }

    public override Matrix Value { get; }
5  }

```

SkewTransform defines a skew by angle degrees along the x and y direction. The skew angle is in degrees. The static matrix representation for skew by angle is as follows.

10

$$\begin{bmatrix} 1 & \tan(\text{angleY}) & 0 \\ \tan(\text{angleX}) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```

public sealed class SkewTransform : Transform
{
15  public SkewTransform();
    public SkewTransform(
        double angleX,
        double angleY);
    public SkewTransform(
20  double angleX,
        double angleY,
        Point center);
    public SkewTransform(
        double angleX,
25  DoubleAnimationCollection angleXAnimations,
        double angleY,
        DoubleAnimationCollection angleYAnimations,
        Point center,
        PointAnimationCollection pointAnimations);
30  public new SkewTransform Copy();

```

95

```

5  [Animations("AngleXAnimations")]
    public double AngleX { get; set; }
    public DoubleAnimationCollection AngleXAnimations { get; set; }

    [Animations("AngleYAnimations")]
    public double AngleY { get; set; }
    public DoubleAnimationCollection AngleYAnimations { get; set; }

10  [Animations("CenterAnimations")]
    public Point Center { get; set; }
    public PointAnimationCollection CenterAnimations { get; set; }

    public override Matrix Value { get; }

15  }

```

MatrixTransform defines a transform through its mathematical representation:

```

20  public sealed class MatrixTransform : Transform
    {
        public MatrixTransform();
        public MatrixTransform(
            double m11,
            double m12,
25  double m21,
            double m22,
            double offsetX,
            double offsetY);
        public MatrixTransform(Matrix matrix);
30  public new MatrixTransform Copy();

        public Matrix Matrix { get; set; }
    }

```

96

```

public override Matrix Value { get; }
}

```

- 5 When a Transform type property is specified in markup, the property system uses the Transform type converter to convert the string representation to the appropriate Transform derived object. There is currently no way to describe animated properties using this syntax.

10 The syntax is in vector graphics and the corresponding Transform construction is summarized as follows, wherein parameters denoted by "<>" represent optional parameters:

- matrix(m11 m12 m21 m22 offsetX offsetY)
 - AddMatrix(m11, m12, m21, m22, offsetX, offsetY)
- translate(tx <ty>)
 - 15 ◦ AddTranslate(tx, ty).
 - If ty is not specified it's assumed to be 0.
- scale(sx <sy>)
 - AddScale(sx, sy).
 - If sy is not specified it's assumed to be same as sx.
- 20 • rotate(angle <cx> <cy>)
 - AddRotate(angle, Point(cx,cy)).
 - If cx, cy is not specified it's assumed to be 0,0.
- skewX(angle)
 - AddSkew(angle, 0)
- 25 • skewY(angle)
 - AddSkew(0, angle)

transform-list:

wsp* transforms? wsp*

30

transforms:

transform
 | transform comma-wsp+ transforms

transform:
 5 matrix
 | translate
 | scale
 | rotate
 | skewX
 10 | skewY

matrix:
 "matrix" wsp* "(" wsp*
 number comma-wsp
 15 number comma-wsp
 number comma-wsp
 number comma-wsp
 number comma-wsp
 number wsp* ")"

20 translate:
 "translate" wsp* "(" wsp* number (comma-wsp number)? wsp* ")"

scale:
 25 "scale" wsp* "(" wsp* number (comma-wsp number)? wsp* ")"

rotate:
 "rotate" wsp* "(" wsp* number (comma-wsp number comma-wsp number)? wsp*
 ")"

30 skewX:
 "skewX" wsp* "(" wsp* number wsp* ")"

98

skewY:
"skewY" wsp* "(" wsp* number wsp* ")"

5 **EFFECTS**

Effects provide a means to alter the visual contents of a scene in a rendering-centric manner. For example, ImageEffects (raster-based bitmap effects) operate on the image-based, fully composited representation of a portion of a scene. Effects are broken down into various types including ImageEffects, BlendModes and
 10 VectorEffects.

ImageEffects can be used in a retained-mode scene by applying it to a sub-graph or an Element, or it can be used in the standalone image pipeline. In general, a ImageEffect has zero or more inputs and zero or more outputs, which are of type ImageSource. In the immediate mode image pipeline, there need not be an
 15 output, as the ImageEffect can surface other properties which describe attributes of its input. For example, a ImageEffect could emit color histogram information or face-detection information. In the retained mode scene, there is an additional meta-input which provides access to the rendered contents of the sub-graph to which the effect is applied.

/// The ImageEffect class is the base class for all imaging effects (blur, grayscale, etc)
 ///

/// It's possible for an effect to not have any inputs but an effect must always have
 /// at least one output. The default implementations of things assume this. If a derived
 /// effect is going to play with Output/Outputs be sure that at least one is there.

public abstract class System.Windows.Media.ImageEffect: Changeable
 {

 /// This constructor allows a derived class to specify the number of inputs and
 /// outputs for this effect, and this class will then handle the input and output
 /// arrays, including validation of indicies.

 /// This defaults to 1 and 1. If the effect wishes to have a variable
 /// number of inputs or outputs, it can pass -1 for either (or both) counts,
 /// and the input and output collections will allow this.

98

AA

```

/// Finally, these methods are all virtual, so derived classes may choose not
/// to delegate back to the base class, in which case no extra cost is incurred.
protected ImageEffect(int inputCount, int outputCount);

/// This constructor allows a derived class to specify the number of inputs and
/// outputs for this effect, and this class will then handle the input and output
/// arrays, including validation of indices.
/// This defaults to 1 and 1. If the effect wishes to have a variable
/// number of inputs or outputs, it can pass -1 for either (or both) counts,
/// and the input and output collections will allow this.
/// Finally, these methods are all virtual, so derived classes may choose not
/// to delegate back to the base class, in which case no extra cost is incurred.
protected ImageEffect(int inputCount, int outputCount, double scaleX, double
scaleY);

/// This is the first input, and is an alias for Inputs[0]
///
/// Performance Warning:
/// If the input of the effect IS NOT in a format that
/// the effect supports the effect will convert the
/// input to a workable format for you.
public virtual ImageSource Input { get; set; }

/// This is the collection of inputs
public virtual ImageSourceCollection Inputs { get; }

/// This is the first output, and is an alias for Outputs[0]
public virtual ImageSource Output { get; }

/// This is the collection of outputs
public virtual ImageEffectSourceCollection Outputs { get; }

```


protected internal virtual ImageSource GetOutput(int outputIndex);

/// These values contain the horizontal and vertical scale applied to this source.

/// There are occasions when an effect needs to operate at a different resolution

/// or a different coordinate space than the current, logical coordinate space.

/// Thus, these properties enable the consumer to map between local space and

/// ImageEffectSource space.

protected internal virtual double GetScaleX(int outputIndex);

protected internal virtual double GetScaleY(int outputIndex);

protected internal virtual PixelFormat GetFormat(int outputIndex);

/// Width, in pixels, of the image.

protected internal virtual int GetPixelWidth(int outputIndex);

/// Height, in pixels, of the image.

protected internal virtual int GetPixelHeight(int outputIndex);

/// Horizontal DPI of the image.

protected internal virtual double GetDpiX(int outputIndex);

/// Vertical DPI of the image.

protected internal virtual double GetDpiY(int outputIndex);

/// Get a palette for a particular output

protected internal virtual ImagePalette GetPalette(int outputIndex);

/// ImageSource abstract method implementation

///

/// PixelOffset actually doesn't do anything. If you don't want to start at (0,0)

/// in the input, then have your sourceRect start at the point you want.

```

protected internal abstract void Copy(int outputIndex,
    IntegerRect sourceRect, PixelFormat pixelFormat, byte[] pixels, int stride,
    int pixelOffset);
    }
}

```

BlendModes are a specific form of image-based effects. They can be applied to the retained mode scene in generally the same manner as ImageEffects. Thus there is an Element property ("BlendMode"), as well as a BlendMode property on Visual, a
5 PushBlendMode method on IDrawingContext and a BlendMode property on Brush. Blend modes perform a combination of the source and destination colors as the source is composited. Example of Blend modes include Multiply, Add, and so forth.

VectorEffects are another type of effects.

As described in the Overview, BlendModes describe an operation which controls the
10 way in which an image is composited into another image, or into the scene.

BlendModes can be applied to the scene and to Brushes. Each BlendMode describes a way to combine a source pixel and a destination pixel, and is applied to every pixel being composited. BlendModes are applied after a source is scaled or otherwise transformed, and after any effects are applied (including Opacity). Note that when the
15 BlendMode operations are applied, the source and destination are in pre-multiplied alpha format. To specify a BlendMode, the programmer can use one of the BlendModes specified in the BlendModes static class, or can explicitly set the source and destination multipliers. Because in one implementation the multipliers are not extensible and they have no parameters, they are represented by an enum:

```

20  /// The four values associated with each BlendModeMultiplier are
    /// multiplied by the corresponding channel in the color to which
    /// the BlendModeMultiplier is applied.
    /// The factors are multiplied by the Alpha, Red, Green and Blue
    /// channels, respectively
25  /// where in-gamut values range from [0..1].
    public enum BlendModeMultipliers
    {

```

```

5  /// Zero - (0,0,0,0)
    Zero,
    /// One - (1,1,1,1)
    One,
    /// SourceColor - (Alphasource, Redsource, Greensource, Bluesource)
    SourceColor,
    /// InverseSourceColor - (1-Alphasource, 1-Redsource, 1-Greensource,
    /// 1-Bluesource)
    InverseSourceColor,
10  /// SourceAlpha - (Alphasource, Alphasource, Alphasource, Alphasource)
    SourceAlpha,
    /// InverseSourceAlpha - (1-Alphasource, 1-Alphasource, 1-Alphasource,
    /// 1-Alphasource)
    InverseSourceAlpha,
15  /// DestinationColor - (Alphadestination, Reddestination, Greendestination,
    /// Bluedestination)
    DestinationColor,
    /// InverseDestinationColor -
    /// (1-Alphadestination, 1-Reddestination, 1-Greendestination,
20  /// 1-Bluedestination)
    InverseDestinationColor,
    /// DestinationAlpha - (Alphadestination, Alphadestination,
    /// Alphadestination, Alphadestination)
    DestinationAlpha,
25  /// InverseDestinationAlpha -
    /// (1-Alphadestination, 1-Alphadestination, 1-Alphadestination,
    /// 1-Alphadestination)
    InverseDestinationAlpha,
    /// SourceAlphaSaturate - f = min(Alphasource, 1 - Alphadestination),
30  /// (1, f, f, f)
    SourceAlphaSaturate
}

```

103

```

/// This class represents a blend operation between a source and
/// destination surface
public sealed class BlendMode: Changeable
5  {
    // Constructs a custom BlendMode from the provided
    // BlendModeMultipliers
    public BlendMode(BlendModeMultiplier sourceMultiplier,
    BlendModeMultiplier destinationMultiplier);
10
    public new BlendMode Copy();

    public BlendModeMultiplier SourceMultiplier { get; }
    public BlendModeMultiplier DestinationMultiplier { get; }
15 }

/// This is a collection of well-known BlendMode instances
public sealed class BlendModes
{
20 /// Normal – Source is One, Destination is InverseSourceAlpha
    public static Normal {get;}

    /// SourceCopy – Source is One, Destination is Zero
    public static SourceCopy {get;}
25 }

```

HIT TESTING

Hit testing is used to pick visuals in a scene. Some high-level scenarios

30 include lasso selection and rubber band selection, keyboard navigation (used to find the next element to switch focus), determining mouse focus in the element tree,

selecting overlapping elements with transparency (such as images), "thought bubble" hit testing and selecting a character hit in text.

In general, hit-testing provides consistency across Core, Framework and Controls, and operates by starting from the top of the control tree, and returning a control or set of controls by a point or geometry. A control can define whether it is hit or not with support services including rendered geometry, bounding box, out-of-band geometry (hit region), image opacity or mask, and its own logic. The control can return specific hit-related data on hit (e.g., line, character position, and so forth).

The hit test mechanism can filter hit test results in an efficient manner. Further, the hit test mechanism provides flexibility for extending to other types of Visuals and resolving down to sub-primitives within the Visual, e.g., a Retained3DVisual is one example of this.

The hit test walk is a deep right to left walk of the visual tree. There are three participants, the hit tester, the walker, and the control/visual. The hit tester implements two callbacks, one to steer the walker, and the other to end the walk early on certain hit visuals. The control implements a virtual method to define what is hit. The walker is a fixed part of the system and walks the visual tree based on callback behavior, essentially asking each control whether the control has been hit. Hits are reported through a callback in z-order, top-to-bottom fashion.

Internally, hit testing thus comprises a walk of the visual tree. When descending, the hit tester views the filtering in terms of element level relationships, for example, a canvas with shapes, or a dock panel with an inner canvas. When a hit occurs, the hit tester can either continue processing further hits (if any), or stop.

The control flow logic from the hit walker perspective is in the following pseudocode:

- For each visual, starting at root:
 - If hit point inside coalesced children hit bounds then (eg, `HitTestBounds.Contains(HitPoint)`)
 - Call hit tester delegate (e.g., `HitTestFilterDelegate`)
 - If null or returns Continue then (e.g., `HitTestFilterBehavior.Continue`)
 - For each child visual

- Transform hit point to its local space
- Hit test against child visual (eg, Visual.HitTest(...))
- If child returns Stop then return (e.g., HitTestResultBehavior.Stop)
- If hit point inside visual (eg, Visual.HitTestCore(...))
 - Call hit tester result delegate (HitTestResultDelegate)
 - Either continue or stop (e.g., HitTestResultBehavior.Continue)

The hit tester uses public methods to initiate the hit test and provides delegates for controlling behavior. The default behavior is to test against all visuals and return on the first hit. If no result delegate is given, an exception is thrown.

- 5 The control decides its hit test logic by overriding HitTestCore for point and geometry. When a hit test is initiated, internal visual tree walker calls HitTestCore, in effect asking the control whether it is hit. The HitTestBounds reflect tight bounds of the hit region and is used to optimize the walk. The default visual behavior is to test against the render content bounding box. The default hit bounds is the render content
- 10 bounding box.

```

public class Visual : DependencyObject, IDisposable, IVisual
{
    // Ignore non hit testing stuff

    // HitTestCore is implemented by visual author to hit against its
    // content only.
    protected virtual PointHitTestResult HitTestCore(
        PointHitTestParameters point);
    protected virtual GeometryHitTestResult HitTestCore(
        GeometryHitTestParameters geometry);

    protected virtual Rect HitTestBounds { get; }
  
```

106

```

}

public delegate HitTestFilterBehavior HitTestFilterDelegate(Visual visual);

5 public delegate HitTestResultBehavior HitTestResultDelegate(HitTestResult result)

public interface IVisual
{
    // Ignore non hit testing stuff

10    // Return top most visual hit
    public PointHitTestResult HitTest(Point point);

    // HitTest is called to initiates a hit test against a visual tree.
15    void HitTest(
        HitTestFilterDelegate filterHitDelegate,
        HitTestResultDelegate resultHitDelegate,
        HitTestParameters hitTestParams);
}

20

```

The hit tester initiates the hit test by passing in a hit point or geometry and additional parameters in the HitTestParameters. The class is provided primarily to simplify the design and allow for extensibility going forward. Special hit testing requests can derive from this class to pass additional information to interested

25 controls. Each control implements a specific HitTestCore against a point and geometry. It is expected that controls respect the hit testing parameters when implementing their HitTestCore logic.

```

public abstract class HitTestParameters
{
30    // Common hit testing parameters
    internal HitTestParameters();
}

```

216

107

```

public class PointHitTestParameters : HitTestParameters
{
    public PointHitTestParameters(Point point);

    Point HitPoint { get; }
}

// Returns detailed intersection information between
// two geometry objects.
enum IntersectionDetail
{
    // Intersection detail is empty or not requested.
    EmptyOrNotCalculated,

    // First geometry is fully inside second geometry.
    FullyInside,

    // Second geometry fully contains the first geometry.
    FullyContains,

    // First and second geometry edges intersect.
    Intersects
}

public class GeometryHitTestParameters : HitTestParameters
{
    Public GeometryHitTestParameters(
        Geometry geometry,
        bool computeIntersectionDetail);

    public Geometry HitGeometry { get; }

```



```

public bool ComputeIntersectionDetail { get; }
}

```

5 The control returns specific data by deriving from HitTestResult. For example, a text control may want to return the character position hit. The PointHitTestResult contains the local coordinate space point. The GeometryHitTestResult contains the local coordinate space geometry (of the original hit test.) The visual transform functions can map the hit location to an ancestor space.

10

```

public abstract class HitTestResult
{
    internal HitTestResult(Visual visual);

    public Visual Visual { get; }
}

public class PointHitTestResult : HitTestResult
{
    public PointHitTestResult(Visual visual, Point point);

    public Point Point { get; }
}

public class GeometryHitTestResult : HitTestResult

    public GeometryHitTestResult(
        Visual visual,
        Geometry geometry,
        IntersectionDetail intersectionDetail);

    public Geometry Geometry { get; }
}

```

109

```

// Make the specified Changeable be unmodifiable (IsChangeable == false) and
// returns true when the 'checking' parameter is 'false'. Otherwise, it checks
// to see if the changeable can be made unchangeable, returning true or false
// as appropriate. If null is passed in as the changeable, this operation
// doesn't do anything and returns 'true'.
static protected bool MakeUnchangeable(Changeable changeable, bool checking);
}

// UsageOverride to be used in ChangeableHelper.UseChangeable
public enum ChangeableUsageOverride
{
    NoOverride, // default, use as prescribed
    ForceUnchangeable, // force the "use" to be unchangeable
    PreferChangeableReference // as described above.
}

// This helper class exists to reduce OM clutter on
// the core Changeable type.
public class System.Windows.ChangeableHelper
{
    // HOSTING API. This is for use by systems other than the Property System
    // that are going to host these values. The Property System will use
    // these entry points for their own hosting in WriteLocal

    // remove handler from getting notified of changes to oldChangeable
    static public void UnsubscribeChangeable(Changeable changeable,
                                              EventHandler handlerToRemove);

    // return a Changeable, adding handler to getting notified of changes to it,
    // if any. Note that this is a "use" of the Changeable, and the returned value
    // will follow the "use" semantics. If "usageOverride" is NoOverride, all

```

```
// proceeds as expected, if it's "ForceUnchangeable", then
// always treat the Changeable as StatusOfNextUse==UseStatus.Unchangeable.
// See above for a description
static public Changeable UseChangeable(Changeable changeable,
                                     EventHandler handlerToAssociate,
                                     ChangeableUsageOverride usageOverride);
}
```

Note that the act of a qualified use of a Changeable, relying solely on its StatusOfNextUse, does not work exactly in every situation. In general, the problem is that when a changeable object (such as Brush, VideoData, and so forth) gets assigned into a Element property (like VideoSource), that changeable is "used" in a qualified use. In the case of animate Changeables (such as VideoData, but also any animation), the act of "using" creates a clone, which is the correct and expected behavior. Then, when the elements' OnRender() method is invoked, the OnRender() implementation typically pushes the value into a DrawingContext, e.g., through DrawingContext.DrawVideo(videoData, ...). This call into the DrawingContext also "uses" the Changeable (videoData in this case), resulting in another clone being made.

Both behaviors of when changeables are "used" in this manner are correct and make sense, when considered in isolation. However, the problem occurs when they combine, in that the implementer of the control does not expect a qualified use every time OnRender() is invoked, and there really is no benefit in doing so, since this use is not exposed to the application, and indeed, is pure overhead that should be eliminated. Moreover, when dependent animations and independent animations combine, OnRender() will be frequently called, and animations will be repeatedly copied, which is not correct behavior. A mechanism called ChangeableReference allows for a "use" to not actually copy, but to instead only obtain a reference to the value being used.

A solution is to have cooperation between an entity like the DrawingContext and the DependencyProperties on a DependencyObject. In particular, the DependencyProperty of a control, when having a value set into it, needs to indicate that it will "allow" the Changeable to be treated as a ChangeableReference if the

114
particular context in which it is subsequently used wants it to be such. Then, the DrawingContext operations say indicate that they would "prefer" for the Changeable to be treated as a ChangeableReference, provided the Changeable itself will allow it.

To this is, there is provided a Boolean property named

- 5 Changeable.AllowChangeableReferenceOverride and the enumeration ChangeableUsageOverride, used in ChangeableHelper.UseChangeable. In this implementation, UseChangeable works as before with true/false mapping to ForceUnchangeable/NoOverride. If UseChangeable is called with PreferChangeableReference and the changeable has IsChangeable==true, and the
10 changeable has AllowChangeableReferenceOverride==true, then the use of the Changeable will be as a ChangeableReference.

- This is used by having DependencyObject.SetValue() set the Changeable it has retained (when it is modifiable) to AllowChangeableReferenceOverride, and having DrawingContext methods invoke UseChangeable with
15 UsageOverridePreferChangeableReference.

- Note that when both conditions are not true, the correct behavior also occurs, in that Elt2.Prop = Elt1.Prop will use the property as expected in a qualified use, copying it if it is modifiable, unless it has been explicitly set to ChangeableReference, since UseChangeable will not be invoked with the PreferChangeableReference. Direct
20 usage of the DrawingContext will also function appropriately, since the Changeables being sent down to it will not have AllowChangeableReferenceOverride.

- Note that when there is a changeable where a subobject of it is ChangeableReference, a shallow clone and a deep clone may be made. A Shallow Clone should work, as the CloneCore method will create a new shallow "shell", and
25 assign over the children, not going deeper into them. With deep clones, the process is straightforward in the case of a tree of ChangeableCopys and Unchangeables, by cloning down to Unchangeables, making each clone along the way itself be Unchangeable (assuming CanMakeUnchangeable is true). This results in a deep clone where the top level is Changeable, and everything below it is Unchangeable. Note that
30 dotting down will make the sub-elements modifiable again.

However, when there is a ChangeableReference, the clone operation needs to be effectively performed, however a reference is maintained for a "Changeable" path

down to the ChangeableReference. This is needed so that when there is a notification from the ChangeableReference, the correct handlers get invoked for everywhere that it is hosted.

Consider the following example:

```

5  Brush b = new LinearGradientBrush();
   b.Stops = new GradientStopCollection();
   GradientStop gs = new GradientStop();
   gs.StatusOfNextUse = UseStatus.ChangeableReference;
   b.Stops.Add(gs);
10
   Button button1, button2;

   button1.Background = b;
   button2.Background = b; (or button2.Background = button1.Background)
15
   gs.Color = Colors.Purple;
```

Here, a LinearGradientBrush is created, as is its Stops collection and a single Stop, and is made a ChangeableReference. The brush may be used in multiple places, and a modification to the ChangeableReference GradientStop needs to affect both brushes.

EXAMPLES (VALID AN INVALID) OF USING CHANGEABLE OBJECTS

The following section provides a summary of using and manipulating objects that derive from the Changeable class, by which objects, such as brushes, pens, and animations, are mutable as controlled by a programmer. Classes that derive from Changeable simulate mutability by automatically building an immutable version of themselves when used in a qualified use. As described above, a Changeable is considered used in a qualified use when the object is set into a Property System property, used as a sub-object in a complex Changeable object, or used in a DrawingContext command.

113

When developing applications with such objects, graphics and media system objects are generally created, set, used, and then never modified. For example, to set the background of a Button a programmer might use a SolidColorBrush, which derives from Changeable; but the programmer might never modify the button's background again over the course of the application. The following is one example:

```
// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.Opacity = 0.5;

// A "use" of myBrush.
myButton.Background = myBrush;

' VB .NET
Dim myBrush As new _
    System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5

' A "use" of myBrush.
myButton.Background = myBrush
```

When used in this manner, a Changeable behaves like a value type, such as a Rect or Color. The Changeable is copied into its destination, and changes to the original do not affect changes to the used value. However, there are situations where the programmer might need to modify such an object after it has been used. For example, suppose that the programmer want to change the background of the button in the previous code after the user clicks it.

The Changeable pattern exists to meet the needs of a variety of situations such as the one above. In general, a Changeable is a value that may or may not be modifiable, denoted by the value of the IsChangeable property. Attempting to modify the value when IsChangeable is false results in an exception. Furthermore,

IsChangeable objects that can be modified raise their Changed events when they change or when any of their members is changed. When working with Changeables, it is thus important to understand when a Changeable is "used" in a qualified use.

- 5 By default, when a Changeable object is used in a qualified use, an immutable copy is created and that copy is actually used. The copy has an IsChangeable value of false. The following code causes an exception to be thrown because the code attempts to modify the unmodifiable copy of myBrush that was used to set the button's background:

```
// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.Opacity = 0.5;
myButton.Background = myBrush;

// Causes an exception to be thrown.
((SolidColorBrush)myButton.Background).Color = Colors.Blue;

' VB .NET
Dim myBrush As new _
    System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Causes an exception to be thrown.
CType(myButton.Background, System.Windows.Media.SolidColorBrush).Color = _
    System.Windows.Media.Colors.Blue
```

- 10 Modifying the original Changeable object does not update the copies:

```
// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.Opacity = 0.5;
myButton.Background = myBrush;
```

103

```
// Does not change the background of the button.
myBrush.Color = Colors.Blue;

' VB .NET
Dim myBrush As new _
    System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Does not change the background of the button.
myBrush.Color = System.Windows.Media.Colors.Blue
```

To change the background of the button in this example, the programmer reassigns the modified brush to the button's background property:

```
// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.Opacity = 0.5;
myButton.Background = myBrush;

// Does not change the background of the button.
myBrush.Color = Colors.Blue;

// Updates the background of the button.
myButton.Background = myBrush;

' VB .NET
Dim myBrush As new _
    System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Does not change the background of the button.
```

Test


```
myBrush.Color = System.Windows.Media.Colors.Blue
```

```
' Updates the background of the button.
```

```
    myButton.Background = myBrush
```

The programmer can also use the Copy method to retrieve a modifiable copy of a used Changeable object. The retrieved copy is still reassigned back to the property to have an effect:

```
// C#
```

```
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
```

```
myBrush.Opacity = 0.5;
```

```
myButton.Background = myBrush;
```

```
SolidColorBrush anotherBrush = (SolidColorBrush)myButton.Background.Copy();
```

```
anotherBrush.Color = Colors.Purple;
```

```
// Updates the background of the button.
```

```
myButton.Background = anotherBrush;
```

```
' VB .NET
```

```
Dim myBrush As new _
```

```
    System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
```

```
myBrush.Opacity = 0.5
```

```
myButton.Background = myBrush
```

```
Dim anotherBrush As new System.Windows.Media.SolidColorBrush
```

```
anotherBrush = _
```

```
    CType(myButton.Background.Copy, System.Windows.Media.SolidColorBrush)
```

```
anotherBrush.Color = System.Windows.Media.Colors.Purple
```

```
' Updates the background of the button.
```

```
    myButton.Background = anotherBrush
```

Because this is not an ideal behavior of a **Changeable** in all situations, e.g., consider the programmer wanting to modify the used version (the working copy) of the **Changeable**, the **Changeable** class enables the programmer to specify how it behaves when used by providing the **StatusOfNextUse** property.

The **StatusOfNextUse** provides three options of how a **Changeable** behaves when used:

Unchangeable: The default behavior, shown in the examples in the previous section. When the **Changeable** object is used, it creates an immutable copy of itself which is used in place of the original object. The programmer may continue to modify the original object; the used version (the copy that was made) is unaffected by modifications to the original object and cannot be modified. To modify the used version, the **Copy** method is used to obtain a modifiable version, that version is updated, and the new version replaces the used version.

ChangeableCopy: When the **Changeable** object is used, it creates a modifiable copy of itself which is used in place of the original object. The programmer may continue to modify the original object; the used version (the copy that was made) is unaffected by modifications to the original object, but is also modifiable. The used version has a **StatusOfNextUse** of **Unchangeable**.

ChangeableReference: When the **Changeable** object is used, it provides a reference to itself. The programmer may continue to modify the original object; changes to the original object affect the used version—they are the same object.

The **ChangeableCopy** alters the behavior of a **Changeable** so that when used it creates a modifiable copy of itself, not an unmodifiable copy (as is the case with the default setting of **Unchangeable**). The following code (shown previously) throws an exception because the **StatusOfNextUse** property of **myBrush** has a default setting of **Unchangeable**:

```
// C#  
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
```

106

```

myBrush.Opacity = 0.5;
myButton.Background = myBrush;

// Causes an exception to be thrown.
((SolidColorBrush)myButton.Background).Color = Colors.Blue;

' VB .NET
Dim myBrush As new _

System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Causes an exception to be thrown.
CType(myButton.Background, System.Windows.Media.SolidColorBrush).Color =
_

System.Windows.Media.Colors.Blue

```

However, if the `StatusOfNextUse` property of the brush is set to `ChangeableCopy`, the code works as intended:

```

// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.StatusOfNextUse = UseStatus.ChangeableCopy;
myBrush.Opacity = 0.5;
myButton.Background = myBrush;

// Works because the brush has a UseStatus of ChangeableCopy.
((SolidColorBrush)myButton.Background).Color = Colors.Blue;

' VB .NET
Dim myBrush As new _

```

707

```

System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.StatusOfNextUse = MS Avalon.Windows.UseStatus.ChangeableCopy
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Works because the brush has a UseStatus of ChangeableCopy.
CType(myButton.Background, System.Windows.Media.SolidColorBrush).Color = _
    System.Windows.Media.Colors.Blue

```

The ChangeableCopy setting also keeps any sub-objects of the main object modifiable. In the following example, a LinearGradientBrush is given a StatusOfNextUse of ChangeableCopy. As a result, the LinearGradientBrush and its sub-objects remain modifiable after they've been used; the programmer does not have to set the StatusOfNextUse property of any Changeable objects contained by the object, such as the GradientStop in this example:

```

// C#
LinearGradientBrush myBrush = new LinearGradientBrush();
myBrush.StatusOfNextUse = UseStatus.ChangeableCopy;

myBrush.GradientStops.Add(new GradientStop(Colors.Blue, 0));
myBrush.GradientStops.Add(new GradientStop(Colors.Green, 1));

myButton.Background = myBrush;

// Works because the brush has a UseStatus of ChangeableCopy.
((LinearGradientBrush)myButton.Background).GradientStops[0].Color =
Colors.LightBlue;

' VB .NET
Dim myBrush As new System.Windows.Media.LinearGradientBrush
myBrush.StatusOfNextUse = MS Avalon.Windows.UseStatus.ChangeableCopy

```

```

myBrush.GradientStops.Add( _
    new System.Windows.Media.GradientStop(System.Windows.Media.Colors.Blue,0))
myBrush.GradientStops.Add(new _
    System.Windows.Media.GradientStop(System.Windows.Media.Colors.Green, 1))

myButton.Background = myBrush

' Works because the brush has a UseStatus of ChangeableCopy.
CType(myButton.Background, _
    System.Windows.Media.LinearGradientBrush).GradientStops(0).Color = _
    System.Windows.Media.Colors.LightBlue

```

- When using Changeable objects with a StatusOfNextUse of ChangeableCopy, the programmer can also retain a handle to the used version of the Changeable and use that reference to modify the object. In the following example, a reference to the used
- 5 LinearGradientBrush is retrieved and used to modify the background of a button:

```

// C#
LinearGradientBrush myBrush = new LinearGradientBrush();
myBrush.StatusOfNextUse = UseStatus.ChangeableCopy;

myBrush.GradientStops.Add(new GradientStop(Colors.Blue, 0));
myBrush.GradientStops.Add(new GradientStop(Colors.Green, 1));

myButton.Background = myBrush;

LinearGradientBrush usedBrush = (LinearGradientBrush)myButton.Background;

// Works because the brush has a UseStatus of ChangeableCopy.
usedBrush.GradientStops[0].Color = Colors.LightBlue;

' VB .NET
Dim myBrush As new System.Windows.Media.LinearGradientBrush

```

```

myBrush.StatusOfNextUse = MS Avalon.Windows.UseStatus.ChangeableCopy
myBrush.GradientStops.Add( _
    new
    System.Windows.Media.GradientStop(System.Windows.Media.Colors.Blue,0))
myBrush.GradientStops.Add( _
    new
    System.Windows.Media.GradientStop(System.Windows.Media.Colors.Green,1))

myButton.Background = myBrush

Dim usedBrush As new System.Windows.Media.LinearGradientBrush
usedBrush = CType(myButton.Background, LinearGradientBrush)

' Works because the brush has a UseStatus of ChangeableCopy.
usedBrush.GradientStops(0).Color = System.Windows.Media.Colors.LightBlue

```

- The ChangeableReference setting alters the behavior of a Changeable so that it provides a reference to itself when used. The programmer may continue to modify the original object; changes to the original object affect the used version because they are the same object. The following is an example:
- 5

```

// C#
SolidColorBrush changeableReferenceBrush = new SolidColorBrush();
changeableReferenceBrush.Color = Colors.LimeGreen;

button1.Background = changeableReferenceBrush;
button2.Background = changeableReferenceBrush;
button3.Background = changeableReferenceBrush;

// Changes the color of the three buttons.
changeableReferenceBrush.Color = Colors.Purple;

```

```
// Also changes the color of all three buttons.
((SolidColorBrush)button1.Background).Color = Colors.Blue;

' VB .NET
Dim changeableReferenceBrush As new System.Windows.Media.SolidColorBrush
changeableReferenceBrush.Color = System.Windows.Media.Colors.LimeGreen

button1.Background = changeableReferenceBrush
button2.Background = changeableReferenceBrush
button3.Background = changeableReferenceBrush

' Changes the color of all three buttons.
changeableReferenceBrush.Color = System.Windows.Media.Colors.Purple

' Also changes the color of all three buttons.
CType(button1.Background, System.Windows.Media.SolidColorBrush).Color = _
    System.Windows.Media.Colors.Blue
```

BRUSH AND PEN

A brush is an object that represents a method to fill a plane. In addition to being able to fill a plane in an absolute way, brushes of the media integration layer are also able to adapt how they fill the plane relative to the size of the object that they are filling. Examples of types of brushes include SolidColorBrush, VisualBrush (which can reference a Visual), DrawingBrush (which can reference a vector graphics resource), LinearGradient, RadialGradient, ImageBrush and NineGridBrush. Default values for brush properties are specified below, and are generally values which result in no action. That is, colors default to transparent, and so on. Also, animation collections default to null.

As mentioned above, certain brush objects will have an idea of how they relate to the coordinate system when they are used, and an idea of how they relate to the

bounding box of the geometry with which they are used. This size is based on from the object that the brush is filling.

The Brush base class has a Transform, a general opacity, and a blend mode:

```
5 public abstract class System.Windows.Media.Brush : Changeable
{
    internal Brush();
    public new Brush Copy(); // hides Changeable.Copy()

    // Default is Transform.Identity
10 public Transform Transform { get; set; }

    [Animation("OpacityAnimations")]
    public double Opacity { get; set; } // Default is 1.0
    public DoubleAnimationCollection OpacityAnimations { get;set; }
15
    /// The BlendMode to apply to this Brush and its destination when drawn.
    /// Default is BlendModes.Normal
    public BlendMode BlendMode {get; set; }
20 }
```

Brush (and other object resources in Vector Graphics and the MIL API) objects are Changeables and are writable after they have been created, and follow the general Changeable pattern for how they behave after they are used in qualified use.

25 The brushes (except for VisualBrush and DrawingBrush) have a simple syntax for use in markup, however, this simple syntax does not allow access to all properties and behavior. If a programmer needs more access than the simple syntax provides, the programmer will need to use complex syntax. Note that only the simple syntax is documented here to avoid redundancy, because the complex syntax follows the same pattern as other CLR classes.

30 The following are the brush types with simple syntax for markup in a current implementation:

12A

```
brush:  
    solid-color-brush |  
    linear-gradient-brush |  
    radial-gradient-brush |  
5    image-brush |  
    video-brush |  
    nine-grid-brush
```

10 Many of the brush types use a coordinate system for specifying some of their parameters. This coordinate system can be defined as relative to the simple bounding box of the geometry with which the brush is used, or it can be absolute, and interpreted in the coordinate space that is active at the time that the brush is used. These are known, respectively, as `RelativeToBoundingBox` mode and `Absolute` mode.

```
15 public enum System.Windows.Media.BrushMappingMode  
    {  
        Absolute,  
        RelativeToBoundingBox,  
    }  
20
```

`SolidColorBrush` fills the plane with a solid color. If there is an alpha component of the color, it is combined in a multiplicative way with the corresponding opacity attribute in the `Brush`.

```

public sealed class System.Windows.Media.SolidColorBrush : Brush
{
    // Constructors
    public SolidColorBrush(); // initialize to transparent
5    public SolidColorBrush(Color color);
    public new SolidColorBrush Copy(); // hides Changeable.Copy()

    // Default is transparent
    [Animation("ColorAnimations")]
10    public Color Color { get; set; }
    public ColorAnimationCollection ColorAnimations { get; set; }
}

```

Because this is a simple type, (that is, none of its properties are Changeables),
15 the only protected method that needs to be implemented is CloneCore(). Also, since
there is no combination of values that make this object invalid, there is no need to
provide a ValidateObjectState() method. These methods and other related methods
are described in the attached Appendix.

The simple syntax for markup for SolidColorBrush:

```

20 solid-color-paint:
    color

```

The Brushes class contains static properties for the SolidColorBrush instances
that are exposed. Each is set to a color value of the same name. Note that because
25 these are standardized brushes, they have IsChangeable set to false, (e.g., the
implementation calls MakeUnchangeable() upon construction).

The following set forth some standard colors:

```

public sealed class Brushes
{
30    public static SolidColorBrush AliceBlue { get; }
    public static SolidColorBrush AntiqueWhite { get; }
}

```

TTT

```
public static SolidColorBrush Aqua { get; }
public static SolidColorBrush Aquamarine { get; }
public static SolidColorBrush Azure { get; }
public static SolidColorBrush Beige { get; }
5 public static SolidColorBrush Bisque { get; }
public static SolidColorBrush Black { get; }
public static SolidColorBrush BlanchedAlmond { get; }
public static SolidColorBrush Blue { get; }
public static SolidColorBrush BlueViolet { get; }
10 public static SolidColorBrush Brown { get; }
public static SolidColorBrush BurlyWood { get; }
public static SolidColorBrush CadetBlue { get; }
public static SolidColorBrush Chartreuse { get; }
public static SolidColorBrush Chocolate { get; }
15 public static SolidColorBrush Coral { get; }
public static SolidColorBrush CornflowerBlue { get; }
public static SolidColorBrush Cornsilk { get; }
public static SolidColorBrush Crimson { get; }
public static SolidColorBrush Cyan { get; }
20 public static SolidColorBrush DarkBlue { get; }
public static SolidColorBrush DarkCyan { get; }
public static SolidColorBrush DarkGoldenrod { get; }
public static SolidColorBrush DarkGray { get; }
public static SolidColorBrush DarkGreen { get; }
25 public static SolidColorBrush DarkKhaki { get; }
public static SolidColorBrush DarkMagenta { get; }
public static SolidColorBrush DarkOliveGreen { get; }
public static SolidColorBrush DarkOrange { get; }
public static SolidColorBrush DarkOrchid { get; }
30 public static SolidColorBrush DarkRed { get; }
public static SolidColorBrush DarkSalmon { get; }
public static SolidColorBrush DarkSeaGreen { get; }
```

```

5  public static SolidColorBrush DarkSlateBlue { get; }
   public static SolidColorBrush DarkSlateGray { get; }
   public static SolidColorBrush DarkTurquoise { get; }
   public static SolidColorBrush DarkViolet { get; }
   public static SolidColorBrush DeepPink { get; }
   public static SolidColorBrush DeepSkyBlue { get; }
   public static SolidColorBrush DimGray { get; }
   public static SolidColorBrush DodgerBlue { get; }
   public static SolidColorBrush Firebrick { get; }
10  public static SolidColorBrush FloralWhite { get; }
   public static SolidColorBrush ForestGreen { get; }
   public static SolidColorBrush Fuchsia { get; }
   public static SolidColorBrush Gainsboro { get; }
   public static SolidColorBrush GhostWhite { get; }
15  public static SolidColorBrush Gold { get; }
   public static SolidColorBrush Goldenrod { get; }
   public static SolidColorBrush Gray { get; }
   public static SolidColorBrush Green { get; }
   public static SolidColorBrush GreenYellow { get; }
20  public static SolidColorBrush Honeydew { get; }
   public static SolidColorBrush HotPink { get; }
   public static SolidColorBrush IndianRed { get; }
   public static SolidColorBrush Indigo { get; }
   public static SolidColorBrush Ivory { get; }
25  public static SolidColorBrush Khaki { get; }
   public static SolidColorBrush Lavender { get; }
   public static SolidColorBrush LavenderBlush { get; }
   public static SolidColorBrush LawnGreen { get; }
   public static SolidColorBrush LemonChiffon { get; }
30  public static SolidColorBrush LightBlue { get; }
   public static SolidColorBrush LightCoral { get; }
   public static SolidColorBrush LightCyan { get; }

```

128

```
public static SolidColorBrush LightGoldenrodYellow { get; }
public static SolidColorBrush LightGray { get; }
public static SolidColorBrush LightGreen { get; }
public static SolidColorBrush LightPink { get; }
5 public static SolidColorBrush LightSalmon { get; }
public static SolidColorBrush LightSeaGreen { get; }
public static SolidColorBrush LightSkyBlue { get; }
public static SolidColorBrush LightSlateGray { get; }
public static SolidColorBrush LightSteelBlue { get; }
10 public static SolidColorBrush LightYellow { get; }
public static SolidColorBrush Lime { get; }
public static SolidColorBrush LimeGreen { get; }
public static SolidColorBrush Linen { get; }
public static SolidColorBrush Magenta { get; }
15 public static SolidColorBrush Maroon { get; }
public static SolidColorBrush MediumAquaMarine { get; }
public static SolidColorBrush MediumBlue { get; }
public static SolidColorBrush MediumOrchid { get; }
public static SolidColorBrush MediumPurple { get; }
20 public static SolidColorBrush MediumSeaGreen { get; }
public static SolidColorBrush MediumSlateBlue { get; }
public static SolidColorBrush MediumSpringGreen { get; }
public static SolidColorBrush MediumTurquoise { get; }
public static SolidColorBrush MediumVioletRed { get; }
25 public static SolidColorBrush MidnightBlue { get; }
public static SolidColorBrush MintCream { get; }
public static SolidColorBrush MistyRose { get; }
public static SolidColorBrush Moccasin { get; }
public static SolidColorBrush NavajoWhite { get; }
30 public static SolidColorBrush Navy { get; }
public static SolidColorBrush OldLace { get; }
public static SolidColorBrush Olive { get; }
```

```

public static SolidColorBrush OliveDrab { get; }
public static SolidColorBrush Orange { get; }
public static SolidColorBrush OrangeRed { get; }
public static SolidColorBrush Orchid { get; }
5 public static SolidColorBrush PaleGoldenrod { get; }
public static SolidColorBrush PaleGreen { get; }
public static SolidColorBrush PaleTurquoise { get; }
public static SolidColorBrush PaleVioletRed { get; }
public static SolidColorBrush PapayaWhip { get; }
10 public static SolidColorBrush PeachPuff { get; }
public static SolidColorBrush Peru { get; }
public static SolidColorBrush Pink { get; }
public static SolidColorBrush Plum { get; }
public static SolidColorBrush PowderBlue { get; }
15 public static SolidColorBrush Purple { get; }
public static SolidColorBrush Red { get; }
public static SolidColorBrush RosyBrown { get; }
public static SolidColorBrush RoyalBlue { get; }
public static SolidColorBrush SaddleBrown { get; }
20 public static SolidColorBrush Salmon { get; }
public static SolidColorBrush SandyBrown { get; }
public static SolidColorBrush SeaGreen { get; }
public static SolidColorBrush SeaShell { get; }
public static SolidColorBrush Sienna { get; }
25 public static SolidColorBrush Silver { get; }
public static SolidColorBrush SkyBlue { get; }
public static SolidColorBrush SlateBlue { get; }
public static SolidColorBrush SlateGray { get; }
public static SolidColorBrush Snow { get; }
30 public static SolidColorBrush SpringGreen { get; }
public static SolidColorBrush SteelBlue { get; }
public static SolidColorBrush Tan { get; }

```

```

    public static SolidColorBrush Teal { get; }
    public static SolidColorBrush Thistle { get; }
    public static SolidColorBrush Tomato { get; }
    public static SolidColorBrush Transparent { get; }
5   public static SolidColorBrush Turquoise { get; }
    public static SolidColorBrush Violet { get; }
    public static SolidColorBrush Wheat { get; }
    public static SolidColorBrush White { get; }
    public static SolidColorBrush WhiteSmoke { get; }
10  public static SolidColorBrush Yellow { get; }
    public static SolidColorBrush YellowGreen { get; }
    }

```

Gradients are drawn by specifying a set of gradient stops. These gradient stops specify the colors along some sort of progression. There are two types of gradients presently supported, namely linear and radial gradients. The gradient is drawn by doing interpolations between the gradient stops in the specified color space.

Gradients are composed of a list of gradient stops. Each of these gradient stops contains a color (with the included alpha value) and an offset. If there are no gradient stops specified, the brush is drawn as transparent (as if there were no brush specified). If there is only one gradient stop specified, the brush is drawn as a solid color with the one color specified. Any gradient stops with offsets in the range of zero to one (0.0...1.0) are considered, along with the largest stop in the range $(-\infty...0.0]$ and the smallest stop in the range $[1.0...+\infty)$. If the set of stops being considered includes a stop which is outside of the range zero to one, an implicit stop is derived at zero (and/or one) which represents the interpolated color which would occur at this stop. Also, if two or more stops are set at the same offset, a hard transition (rather than interpolated) occurs at that offset. The order in which stops are added determines the behavior at this offset; the first stop to be added is the effective color before that offset, the last stop to be set is the effective color after this stop, and any additional stops at this offset are ignored.

This class is a `Changeable` like other resource classes:

```

public sealed class System.Windows.Media.GradientStop : Changeable
{
    public GradientStop(); public GradientStop(Color color, double offset);
5    public GradientStop(Color color, ColorAnimationCollection colorAnimations,
        double offset, DoubleAnimationCollection offsetAnimations);
    public new GradientStop Copy(); // hides Changeable.Copy()

    // Default is transparent
10    [Animation("ColorAnimations")]
    public Color Color { get; set; }
    public ColorAnimationCollection ColorAnimations { get; set; }

    // Default is 0
15    [Animation("OffsetAnimations")]
    public double Offset { get; set; }
    public DoubleAnimationCollection OffsetAnimations { get; set; }
}

```

20 Like SolidColorBrush, this has nested Changeables in the animation collections.

The GradientSpreadMethod enum specifies how the gradient should be drawn outside of the specified vector or space. There are three possible values, including Pad, in which the end colors (first and last) are used to fill the remaining space,

25 Reflect, in which the stops are replayed in reverse order repeatedly to fill the space, and Repeat, in which the stops are repeated in order until the space is filled. The default value for properties of this type is Pad:

```

public enum System.Windows.Media.GradientSpreadMethod
{
30    Pad,
    Reflect,
    Repeat
}

```



```
}

```

FIG. 24 provides some GradientSpreadMethod examples, (albeit in grayscale rather than in color). Each shape has a linear gradient going from white to grey. The solid line represents the gradient vector.

The ColorInterpolationMode enum defines the interpolation mode for colors within a gradient. The two options are PhysicallyLinearGamma and PerceptuallyLinearGamma.

```
10 public enum ColorInterpolationMode
    {
        // Colors are interpolated in Gamma 1.0 space
        PhysicallyLinearGamma,
        // Colors are interpolated in Gamma 2.2 space
15    PerceptuallyLinearGamma
    }
```

This is an abstract base class.

```
20 public abstract class System.Windows.Media.GradientBrush : Brush
    {
        internal GradientBrush();
        public new GradientBrush Copy(); // hides Changeable.Copy()

25    // Default is "PerceptuallyLinearGamma"
        public ColorInterpolationMode ColorInterpolationMode { get; set; }

        // Default is RelativeToBoundingBox
        public BrushMappingMode MappingMode { get; set; }

30    // Default is Pad
    }
```

5

10

```

public GradientSpreadMethod SpreadMethod { get; set; }

// Gradient Stops
public void AddStop(Color color, double offset);
public GradientStopCollection GradientStops { get; set; }

// ColorInterpolationMode
public ColorInterpolationMode ColorInterpolationMode { get; set; }
}

```

The LinearGradient specifies a linear gradient brush along a vector. The individual stops specify colors stops along that vector.

```

public sealed class System.Windows.Media.LinearGradient : GradientBrush
{
    public LinearGradient(); // initializes to transparent

    // Sets up a gradient with two colors and a gradient vector
    // specified to fill the object the gradient is applied to.
    // This implies RelativeToBoundingBox for the GradientUnits
    // property
    public LinearGradient(Color color1, Color color2, double angle);
    public LinearGradient(Color color1, Color color2,
                          Point vectorStart, Point vectorEnd);
    public new LinearGradient Copy(); // hides Changeable.Copy()

    // Gradient Vector Start Point
    // Default is 0,0
    [Animation("StartPointAnimations")]
    public Point StartPoint { get; set; }
    public PointAnimationCollection StartPointAnimations { get; set; }
}

```

```

// Default is 1,1
[Animation("EndPointAnimations")]
public Point EndPoint { get; set; }
public PointAnimationCollection EndPointAnimations { get; set; }
}

```

The simple syntax for markup for LinearGradientBrush:

linear-gradient-brush:

"HorizontalGradient" comma-wsp color comma-wsp color |

"VerticalGradient" comma-wsp color comma-wsp color |

"LinearGradient" comma-wsp coordinate-pair comma-wsp color comma-wsp color

5 The markup for LinearGradient allows specification of a LinearGradient with two color stops, at offsets zero and one. If the "LinearGradient" version is used, the start point and end point are specified, respectively. If "HorizontalGradient" is used, the start point is set to 0,0 and the end point is set to 1,0. If "VerticalGradient" is used, the start point is set to 0,0 and the end point is set to 0,1. In these cases, the default MappingMode is used, which is RelativeToBoundingBox.

10 The RadialGradient is similar in programming model to the linear gradient. However, whereas the linear gradient has a start and end point to define the gradient vector, the radial gradient has a circle along with a focal point to define the gradient behavior. The circle defines the end point of the gradient – in other words, a gradient stop at 1.0 defines the color at the circle's circumference. The focal point defines
15 center of the gradient. A gradient stop at 0.0 defines the color at the focal point. FIG. 25 represents a RadialGradient that (in grayscale) goes from white to grey. The outside circle represents the gradient circle while the solid dot denotes the focal point. This gradient has SpreadMethod set to Pad.

```

public sealed class System.Windows.Media.RadialGradient : GradientBrush
{
    public RadialGradient(); // initialize to transparent

    // Sets up a gradient with two colors.
}

```

```

// This implies RelativeToBoundingBox for the GradientUnits
// property along with a center at (0.5,0.5)
// a radius of 0.5 and a focal point at (0.5,0.5)
public RadialGradient(Color color1, Color color2);

public new RadialGradient Copy(); // hides Changeable.Copy()

// Default is 0.5,0.5
[Animation("CenterAnimations")]
public Point Center { get; set; }
public PointAnimationCollection CenterAnimations { get; set; }

// Default is 0.5
[Animation("RadiusXAnimations")]
public double RadiusX { get; set; }
public DoubleAnimationCollection RadiusXAnimations { get; set; }

// Default is 0.5
[Animation("RadiusYAnimations")]
public double RadiusY { get; set; }
public DoubleAnimationCollection RadiusYAnimations { get; set; }

// Default is 0.5,0.5
[Animation("FocusAnimations")]
public Point Focus { get; set; }
public PointAnimationCollection FocusAnimations { get; set; }
}

```

The markup for RadialGradient allows specification of a RadialGradient with two color stops, at offsets 0 and 1 respectively. The default MappingMode is used, which is RelativeToBoundingBox, as are the default radii, 0.5:

radial-gradient-brush:

"RadialGradient" comma-wsp color comma-wsp color

The **TileBrush** is an abstract base class which contains logic to describe a tile and a means by which that tile should fill an area. Subclasses of **TileBrush** contain content, and logically define a way to fill an infinite plane.

- 5 The **Stretch** enum is used to describe how a **ViewBox** (source coordinate space) is mapped to a **ViewPort** (destination coordinate space). This is used in **TileBrush**:

```
public enum System.Windows.Stretch
{
    // Preserve original size
    None,
    // Aspect ratio is not preserved, ViewBox fills ViewPort
    Fill,
    // Aspect ratio is preserved, ViewBox is uniformly scaled as large as
    // possible such that both width and height fit within ViewPort
    Uniform,
    // Aspect ratio is preserved, ViewBox is uniformly scaled as small as
    // possible such that the entire ViewPort is filled by the ViewBox
    UniformToFill
}
```

- 10 FIG. 26 provides stretch examples. In these examples, the contents are top/left aligned.

The **TileMode** enum is used to describe if and how a space is filled by Tiles. A **TileBrush** defines where the base Tile is (specified by the **ViewPort**). The rest of the space is filled based on the **TileMode** value.

```
public enum System.Windows.Media.TileMode
{
    // Do not tile – only the base tile is drawn, the remaining area is
```

136

```

// left as transparent
None,
// The basic tile mode – the base tile is drawn and the remaining area
// is filled by repeating the base tile such that the right edge of one
// tile butts the left edge of the next, and similarly for bottom and top
Tile,
// The same as tile, but alternate columns of tiles are flipped
// horizontally. The base tile is drawn untransformed.
FlipX,
// The same as tile, but alternate rows of tiles are flipped vertically
// The base tile is drawn untransformed.
FlipY,
// The combination of FlipX and FlipY. The base tile is drawn
// untransformed
FlipXY
}

```

FIG. 27 provides TileMode examples. The top left-most tile in each example is the base tile. These examples represent None, Tile, FlipX, FlipY and FlipXY.

5 The VerticalAlignment enum is used to describe how content is positioned within a container vertically:

```

public enum System.Windows.VerticalAlignment
{
    // Align contents towards the top of a space
    Top,
    // Center contents vertically
    Center,
    // Align contents towards the bottom of a space
    Bottom,
}

```

The **HorizontalAlignment** enum is used to describe how content is positioned within a container horizontally.

```
public enum System.Windows.HorizontalAlignment
{
    // Align contents towards the left of a space
    Left,
    // Center contents horizontally
    Center,
    // Align contents towards the right of a space
    Right,
}
```

- 5 The **TileBrush** properties select a rectangular portion of the infinite plane to be a tile (the **ViewBox**) and describe a destination rectangle (**ViewPort**) which will be the base Tile in the area being filled. The remaining destination area will be filled based on the **TileMode** property, which controls if and how the original tile is replicated to fill the remaining space:

```
public abstract class System.Windows.Media.TileBrush : Brush
{
    public new TileBrush Copy(); // hides Brush.Copy()

    // Default is RelativeToBoundingBox
    public BrushMappingMode ViewPortUnits { get; set; }

    // Default is RelativeToBoundingBox
    public BrushMappingMode ContentUnits { get; set; }

    // Default is Rect.Empty
    [Animation("ViewBoxAnimations")]
    public Rect ViewBox { get; set; }
    public RectAnimationCollection ViewBoxAnimations { get; set; }
}
```

```

// Default is Fill
public Stretch Stretch { get; set; }

// Default is None
public TileMode TileMode { get; set; }

// Default is Center
public HorizontalAlignment HorizontalAlignment { get; set; }

// Default is Center
public VerticalAlignment VerticalAlignment { get; set; }

// Default is 0,0,1,1
[Animation("ViewportAnimations")]
public Rect ViewPort { get; set; }
public RectAnimationCollection ViewPortAnimations { get; set; }
}

```

A TileBrush's contents have no intrinsic bounds, and effectively describe an infinite plane. These contents exist in their own coordinate space, and the space which is being filled by the TileBrush is the local coordinate space at the time of application.

- 5 The content space is mapped into the local space based on the ViewBox, ViewPort, Alignments and Stretch properties. The ViewBox is specified in content space, and this rectangle is mapped into the ViewPort rectangle.

- The ViewPort defines the location where the contents will eventually be drawn, creating the base tile for this Brush. If the value of ViewPortUnits is Absolute, the value of ViewPort is considered to be in local space at the time of application. If, instead, the value of ViewPortUnits is RelativeToBoundingBox, then the value of ViewPort is considered to be in the coordinate space where 0,0 is the top/left corner of the bounding box of the object being painted and 1,1 is the bottom/right corner of the same box. For example, consider a RectangleGeometry being filled which is drawn from 100,100 to 200,200. Then, if the ViewPortUnits is Absolute, a ViewPort of
- 10
- 15

TLC

(100,100,100,100) would describe the entire content area. If the ViewPortUnits is RelativeToBoundingBox, a ViewPort of (0,0,1,1) would describe the entire content area. If the ViewPort's Size is empty and the Stretch is not None, this Brush renders nothing.

5 The ViewBox is specified in content space. This rectangle is transformed to fit within the ViewPort as determined by the Alignment properties and the Stretch property. If the Stretch is None, then no scaling is applied to the contents. If the Stretch is Fill, then the ViewBox is scaled independently in both X and Y to be the same size as the ViewPort. If the Stretch is Uniform or UniformToFill, the logic is
10 similar but the X and Y dimensions are scaled uniformly, preserving the aspect ratio of the contents. If the Stretch is Uniform, the ViewBox is scaled to have the more constrained dimension equal to the ViewPort's size. If the Stretch is UniformToFill, the ViewBox is scaled to have the less constrained dimension equal to the ViewPort's size. Another way to think of this is that both Uniform and UniformToFill preserve
15 aspect ratio, but Uniform ensures that the entire ViewBox is within the ViewPort (potentially leaving portions of the ViewPort uncovered by the ViewBox), and UniformToFill ensures that the entire ViewPort is filled by the ViewBox (potentially causing portions of the ViewBox to be outside the ViewPort). If the ViewBox's area is empty, then no Stretch will apply. Alignment will still occur, and it will position the
20 "point" ViewBox.

Once the ViewPort is determined (based on ViewPortUnits) and the ViewBox's destination size is determined (based on Stretch), the ViewBox needs to be positioned within the ViewPort. If the ViewBox is the same size as the ViewPort (if Stretch is Fill, or if it just happens to occur with one of the other three Stretch values),
25 then the ViewBox is positioned at the Origin so as to be identical to the ViewPort. If not, then HorizontalAlignment and VerticalAlignment are considered. Based on these properties, the ViewBox is aligned in both X and Y dimensions. If the HorizontalAlignment is Left, then the left edge of the ViewBox will be positioned at the Left edge of the ViewPort. If it is Center, then the center of the ViewBox will be
30 positioned at the center of the ViewPort, and if Right, then the right edges will meet. The process is repeated for the Y dimension.

If the ViewBox is Empty it is considered unset. If it is unset, then ContentUnits are considered. If the ContentUnits are Absolute, no scaling or offset occurs, and the contents are drawn into the ViewPort with no transform. If the ContentUnits are RelativeToBoundingBox, then the content origin is aligned with the ViewPort Origin, and the contents are scaled by the object's bounding box's width and height.

When filling a space with a TileBrush, the contents are mapped into the ViewPort as above, and clipped to the ViewPort. This forms the base tile for the fill, and the remainder of the space is filled based on the Brush's TileMode. If set, the Brush's transform is applied, which occurs after the other mapping, scaling, offsetting, and so forth.

A VisualBrush is a TileBrush whose contents are specified by a Visual. This Brush can be used to create complex patterns, or it can be used to draw additional copies of the contents of other parts of the scene.

```
public sealed class System.Windows.Media.VisualBrush : TileBrush
{
    public VisualBrush(); // initializes to transparent
    public VisualBrush(Visual v);

    public new VisualBrush Copy(); // hides TileBrush.Copy()

    // Visual - Default is null (transparent Brush)
    public Visual Visual { get; set; }
}
```

As mentioned, a VisualBrush has no simple syntax for markup, though it can be described via complex syntax.

A DrawingBrush is a TileBrush whose contents are specified by a Drawing. This Brush can be used to create complex patterns which have been created via a DrawingContext.

```
public sealed class System.Windows.Media.DrawingBrush : TileBrush
{
```

```

public DrawingBrush(); // initializes to transparent
public DrawingBrush(Drawing drawing);

public new DrawingBrush Copy(); // hides TileBrush.Copy()

// Drawing - Default is null (transparent Brush)
public Drawing Drawing { get; set; }
}

```

As mentioned, a DrawingBrush has no simple syntax for markup, though it can be described via complex syntax.

ImageBrush is a TileBrush having contents specified by an ImageSource. This
5 Brush can be used to fill a space with an image.

```

public sealed class System.Windows.Media.ImageBrush : TileBrush
{
    public ImageBrush(); // Initializes to transparent contents

    // Sets the image, sets ViewBox to (0,0,Width,Height)
    // and Stretch to Fill
    public ImageBrush(ImageData image);

    public new ImageBrush Copy(); // hides TileBrush.Copy()

    // Default is null
    public ImageSource ImageSource { get; set; }

    // Default is true
    // If this is true, the ViewBox property will be overridden
    // and effectively set to the native size of the Image
    public bool SizeViewBoxToContent { get; set; }
}

```

The simple syntax for markup for ImageBrush:

image-brush:

"Image" image-uri

VideoBrush is a TileBrush having contents specified by a VideoData. This Brush can be used to fill a space with a Video.

```
public sealed class System.Windows.Media.VideoBrush : TileBrush
{
    public VideoBrush(); // Initializes to transparent contents

    // Sets the image, sets ViewBox to (0,0,Width,Height) and the
    // Stretch to Fill
    public VideoBrush(VideoData video);

    public new VideoBrush Copy(); // hides TileBrush.Copy()

    // Default is null
    public VideoData VideoData { get; set; }

    // Default is true
    // If this is true, the ViewBox property will be overridden
    // and effectively set to the native size of the Video
    public bool SizeViewBoxToContent { get; set; }
}
```

5

The simple syntax for markup for VideoBrush:

video-brush:

"Video" video-uri

NineGridBrush is a Brush which always fills the object bounding box with its content image, and the image stretch isn't accomplished purely via a visual scale. The image source is divided into nine rectangles by four borders (hence the name

10

132

NineGrid). The contents of the image in each of those nine regions are scaled in 0, 1 or 2 dimensions until they fill the object bounding box. The dimensions in which each section is scaled can be seen in this diagram: FIG. 28 represents the concept of a NineGrid, showing the nine grids which are defined by the Top, Left, Bottom and
 5 Right borders. The arrows in each grid square show the dimension(s) in which those contents will be stretched to meet the ViewPort size.

In addition to the nine grid regions pictured above, there is an optional "tenth" grid. This takes the form of an additional image which is centered in the ViewPort and which is not scaled. This can be used to place a shape in the center of a button,
 10 etc. This "tenth grid" is called a glyph, and is exposed by the GlyphImageSource property:

```
public sealed class System.Windows.Media.NineGridBrush : Brush
{
    public NineGridBrush(ImageSource imageSource,
        int LeftBorder,
        int RightBorder,
        int TopBorder,
        int BottomBorder);

    public NineGridBrush(ImageSource imageSource,
        int LeftBorder,
        int RightBorder,
        int TopBorder,
        int BottomBorder,
        ImageSource glyphImage);

    public new NineGridBrush Copy(); // hides Brush.Copy()

    // Default is null
    public ImageSource ImageSource { get; set; }

    // Default is 0
```

```

public int LeftBorder { get; set; }

// Default is 0
public int RightBorder { get; set; }

// Default is 0
public int TopBorder { get; set; }

// Default is 0
public int BottomBorder { get; set; }

// Default is null
public ImageSource GlyphImageSource{ get; set; }
}

```

Note that the border members count in from the edge of the image in image pixels

The simple syntax for markup for NineGridBrush:
 nine-grid-brush:

5 "NineGrid" image-uri int int int int [glyph-image-uri]

The four integers are the values for LeftBorder, RightBorder, TopBorder and BottomBorder respectively. The final URI for the tenth-grid, or glyph, is optional.

10 The Pen is an object that takes a Brush and other parameters that describe how to stroke a space/Geometry. Conceptually, a Pen describes how to create a stroke area from a Geometry. A new region is created which is based on the edges of the Geometry, the Pen's Thickness, the PenLineJoin, PenLineCap, and so forth. Once this region is created, it is filled with the Brush.

```

public sealed class System.Windows.Media.Pen : Changeable
{
    // Constructors
    Public Pen();
    public Pen(Brush brush, double thickness);
}

```

```

public new Pen Copy(); // hides Changeable.Copy()

// Properties
// Default is DashArrays.Solid (no dashes)
public DoubleCollection DashArray { get; set;}

// Default is 0
[Animations(DashOffsetAnimations)]
public double DashOffset { get; set;}
public Double.AnimationCollection DashOffsetAnimations { get; set;}

// Default is Flat
public PenLineCap StartLineCap { get; set;}

// Default is Flat
public PenLineCap EndLineCap { get; set;}

// Default is Flat
public PenDashCap DashCap { get; set;}

// Default is Miter
public PenLineJoin LineJoin { get; set;}

// Default is 10
public double MiterLimit { get; set;}

// Default is null
public Brush Brush { get; set;}

// Default is 1.0
[Animations(ThicknessAnimations)]

```

```

public double Thickness { get; set;}
public DoubleAnimationCollection ThicknessAnimations { get; set;}
}

```

The PenLineCap determines how the ends of a stroked line are drawn:

```

public enum System.Windows.Media.PenLineCap
{
    // This is effectively no line cap – the line is squared off
    // at the last point in the line
    Flat,
    // The line is capped by a hemi-circle of diameter equal to
    // the line width
    Round,
    // The dash is capped by a triangle
    Triangle,
    // The line is capped with a square of side with equal to the
    // line width, centered on the end point
    Square
}

```

- 5 The PenDashCap determines how the ends of each dash in a dashed, stroked line are drawn:

```

public enum System.Windows.Media.PenDashCap
{
    // This is effectively no dash cap – the line is squared off
    // at the last point in the line
    Flat,
    // The dash is capped by a hemi-circle of diameter equal to
    // the line width
    Round,
    // The dash is capped by a triangle
}

```


Triangle

}

The PenLineJoin determines how joints are draw when stroking a line:

```
public enum System.Windows.Media.PenLineJoin
{
    // A sharp corner is created at the intersection of the outer
    // edges of the intersecting line segments
    Miter,
    // Similar to Miter, but the corner is rounded
    Round,
    // A beveled join, this produces a diagonal corner
    Bevel
}
```

- 5 The DashArrays class comprises static properties which provide access to common, well-known dash styles:

```
public sealed System.Windows.Media.DashArrays
{
    // A solid Dash array (no dashes)
    public static DoubleCollection Solid { get; }
    // Dash - 3 on, 1 off
    public static DoubleCollection Dash { get; }
    // Dot - 1 on, 1 off
    public static DoubleCollection Dot { get; }
    // DashDot - 3 on, 1 off, 1 on, 1 off
    public static DoubleCollection DashDot { get; }
    // DashDotDot - 3 on, 1 off, 1 on, 1 off, 1 on, 1 off
    public static DoubleCollection DashDotDot { get; }
}
```

COLOR

The color architecture is built upon some general principles, including that Color requires a context; therefore color values will have an explicitly assigned context or an implicitly assumed default to minimize color mismatches in workflows.

- 5 Further, core platform design requires minimal code paths and APIs with long life spans for security, reliability, maintainability, future expansion and performance; therefore the rendering core may be limited to primarily an scRGB code path to which incoming content streams will be converted to and outgoing streams will be converted from (an additional sRGB legacy path that will have lower quality is also allowed).
- 10 Note that "scRGB" refers to the internal vector graphics default representation, based upon the IEC 61966-2-2 international standard; (although no official definition has been provided for what "sc" means, "standard compositing" will be used herein to help clarify that this is the optimum space for composition processing).

- 15 Performance requires that complex processing be performed as close to the color object definition/specifications stage as possible, rather than at the real time rendering stage; this requires that color parameters to APIs be converted to scRGB (essentially immediately) upon specification and that scRGB color values be maintained and synchronized for non-scRGB defined objects. Ease of use requires a layered API in which the most common developer cases are exposed first, while the
- 20 most advanced case have clean, but minimal APIs; therefore sRGB APIs are provided (but immediately converted to scRGB internally), scRGB APIs are provided, and a minimal context associated API is provided to support advanced CMYK (Cyan-Magenta-Yellow-Black), monochrome, human visual system and multichannel solutions. Since scRGB is essentially an "infinite" color gamut, additional device
- 25 characterization and gamut mapping solutions are required to "hook" scRGB workflows with real world devices.

- 30 Color is a psychological perception most often typically caused by an external physical sensation. This means that computer-based color requires a physical context in order to effectively communicate the perceived colors across devices and between users. Historically various technologies have been inconsistent in providing reasonable contextual meaning for color architectural implementations, e.g., this

TSD

150
resulted in "red" meaning "orange" for one device or user and "pink" for another, with little viable methods to resolve the mismatch.

The present architecture provides a combination of implicit (using defaults) and explicit color contexts for any color object. This means that there will be no color object without contextual meaning. This is an imperfect technology, and thus an aspect of the architecture is to provide consistent color contexts in a manner that can be evolved as technology progresses. Note that most computer users (and most developers) do not want to deal with color management, and prefer that color should simply work correctly.

10 In general, the architecture attempts to minimize internal code paths, which is accomplished to an extent by enabling two basic code paths internally, one for quality and the future hardware, and the other for legacy and memory/performance constraints. The MIL core internal rendering and composition engine supports 32bpp sRGB and 128bpp floating point scRGB (although 64bpp scRGB is also under
15 consideration and, some instances of 64bpp support are best implemented with fixed point, some with float and some with integer).

The architecture provides 128bpp scRGB path from capture to display to editing to storage to print (the display will be 128bpp back buffer and 10bpc or more front buffer), and allows legacy 32bpp sRGB path that sacrifices quality for
20 performance, memory, and/or bandwidth.

Color management of the present invention improves on prior drawbacks by providing devices and applications with additional flexibility and provides a profile-based color management solution. The most common scenarios are based on scRGB and sRGB color objects to support getting and setting the color values for common UI
25 elements, and supports most content creation for the web, multimedia and computer graphics. Less common scenarios include using RGB color contexts with specific working space profiles for professional photography workflows, using CMYK color values to edit color object for prepress and graphics design work, and monochrome and multichannel color workflows, which support niche printing and press scenarios
30 as well as provide the flexibility to support future workflows that have not been defined. HVSV (human visual system-based spaces) workflows support some niche professional photography editing scenarios.

To match capture sensor technology, which continues to progress in quality and bit depth, imaging will support at least the 64bpp formats for every feature/API in order to support contemporary digital negative initiatives. As the present invention implements a new architecture for vector graphics, vector graphics will be
5 implemented in 32bit floating point per channel bit precision. This implementation is actually "hidden" in order to provide traditional 8bpc color access as well as grayscale and HSV interfaces.

Another type of color data is named color data, such as "CornflowerBlue" or "Pantone" colors. By providing a color context that is based upon extending
10 traditional color management profiles, a very generic and powerful color naming interface is provided. To maintain some legacy compatibility with previous APIs and common practice, the default constructors will be biased towards sRGB input.

Color space support for vector graphics is divided into native scRGB support, property support for sRGB and similar spaces that require no explicit color context,
15 method support for color spaces closely related to sRGB or scRGB such as HSV (hue, saturation and value) that also require no explicitly associated color context, named colors and related color spaces such as palette and indexed color spaces that are based on either the implicitly or the explicitly associated color context, and color spaces that require additional color channels as well as explicitly associated color contexts, such
20 as CMYK, hi-fi color (CMYK plus orange and green), CcMmYK inkjet color spaces and potentially spectral color support in the future.

While these color spaces are converted to scRGB or sRGB for rendering in the MIL core or compositing engine, they can be stored or sent to printers (such as CMYK) using the vector graphic markup language as a program design language. The
25 color markup syntax includes four basic specification mechanisms: hexadecimal, known colors, sRGB properties and advanced color context. The first three assume an sRGB color space color context.

The example below creates a gradient using these four mechanisms:

Example:

```
<Canvas Width="100px" Height="100px"  
Background="VerticalGradient #ee7711 CornFlowerBlue"
```

739

```

xmlns="using:System.Windows;System.Windows.Controls;System.Windows.Docum
ents;System.Windows.Shapes;System.Windows.Media;System.Windows.Presenters"
>

<Rectangle Top="12.5%" Left="0" Width="100%" Height="12.5%">
  <Rectangle.Fill>
    <RadialGradient>
      <GradientStop Offset="0" Color="sGray.icc 0.5"/>
      <GradientStop Offset="0.5">
        <GradientStop.Color>
          <Color A="0.8" R="0.2" G="1" B="0.2"/>
        </GradientStop.Color>
      </GradientStop>
      <GradientStop Offset="1" Color="mswopintent8.icm 0.9 0.2 0.1 0.3"/>
    </RadialGradient>
  </Rectangle.Fill>
</Rectangle>
</Canvas>

```

The first background color is specified as hexadecimal (#ee7711). This hexadecimal representation is identical to how .NET framework V1 and WinForms specified colors. It is flexible to allow four different variations; #RGB, #ARGB, 5 #RRGGBB, or #AARRGGBB.

The second background color is specified as a known color (CornFlowerBlue). This representation is identical to how .NET framework V1 and WinForms specified colors. It is based upon named color values. The named color solutions will enable Pantone, Trumatch and other named colors to be supported with color contexts. This 10 will also support alpha channel settings.

A first gradient-stop is specified using a color context ("sGray.icc 0.5"). The text string specifies the color context file name, which may be a URL or URI. This

sample is to illustrate clean support for monochrome printing without requiring RGB values to be translated first into monochrome values at print rendering time.

The second gradient stop is specified using sRGB properties (A="0.8" R="0.2" G="1" B="0.2"). This representation is identical to how .NET framework V1 and WinForms specified colors.

The third gradient stop is using a color context (= "mswopintent8.icm 0.9 0.2 0.1 0.3"). The text string specifies the color context file name, which may be a URL or URL, and may support alpha channel settings. This sample shows CMYK support, such as required for Publisher and other such applications.

Taken together, these examples provide a very robust and clean syntax to support color requirements. Note that a color context may be globally specified and internal color references required to conform to this global context, e.g., for performance optimization.

As described above, a color context is required for color objects, whether they are vector or raster based. At a coarse level, a color context can be thought of as a profile, providing a relationship between the color space of the color object and the human visual system. The color context provides the information on the relationship between the user color space and the sRGB color space (or human visual system color). This allows for "round-tripping" CMYK and other color information that has not been previously feasible in an effective manner.

In practice today, there exist literally hundreds of different color spaces such as sRGB, scRGB, AdobeRGB, BruceRGB, AppleRGB, TomRGB, CorbisRGB, JoeRGB, HSV, HSB, XYZ, LAB, LUV, YUV, YCC, CMYK, CcMmYK, CMYKOG, lightness grayscale, luminance grayscale and many, many others. Many of these individual color spaces can be broken down into classes of color spaces such as RGB spaces that are primarily defined by three channels containing approximates to the visual perception of red, green and blue with defining semantics including gamma, primaries, white point and bit precision. Bit precision is necessary since lower bit precisions (such as 8 bits per channel) typically require extensive, nonlinear compensations to achieve reasonable encoding efficiencies.

Providing a color context class reduces the set of possible color space classes to a much smaller set, such as grayscale, RGB, HSV, CMYK, LAB, LCH, CcMmYK,

CMYKOG, spectral colors and special effects spaces such as duotone, tritone and quadtone spaces used for artistic effects. A further reduction is possible by combining spaces which share the same underlying meaning but provide different coordinate systems (similar to rectilinear and polar geometries). This would make HSV a method
 5 on top of the RGB color space class and LCH a method on top of the LAB color space class.

It is also possible to combine special effects color spaces with spectral color and include support for CcMmYK and CMYKOG and have the color value in just this context be a dynamic array since only advanced users will use this feature. A further
 10 reduction is possible to reduce the color spaces to scRGB, which will support sRGB and other RGB spaces, and a multichannel color space with a color context. This leaves a reasonable number of basic color space classes to support, including just scRGB and multi-channel spaces.

The ColorContext is associated with either a vector graphics Color class or an
 15 ImageData object. Another alternative is to limit Visuals to a single ColorContext. This would help optimize the amount of color conversions and validations in many circumstances, and may be more natural for application developers, e.g., an individual control is not likely to use colors from multiple color systems. Note that the color profile is still allowed to be changed for advanced applications that explicitly use
 20 multiple color types via a mechanism that allows dynamic changes. A ColorContext also allows for the rendering intent or color gamut mapping between two device gamuts to be specified and thus associated with the color object. Since the ColorContext only deals with a single color object, the destination gamut can be a virtual device. This allows the ColorContext to contain both the objective description
 25 of the color space as well as the subjective rendering intent for the color.

Color names are simple lookup tables embedded in the profile that is associated with the color context that provide a link between a color value based on the type of ColorContext and the actual color name. This allows different color naming dictionaries for each color object. For example, it is possible to associate one
 30 type of naming system; such as Trumatch with process color objects and another type of naming system, such as Pantone with spot color objects.

The public vector graphics Color type matches the lower level system to optimize performance by minimizing conversions when transmitting data to that lower level system. A separate "native" (i.e. CMYK or such) set of floating point values is stored in the color context and is synchronized when any changes occur. The native

5 ColorContext colorValue is a value type (structure) based on an array of floats in order to transparently support grayscale, RGB and even CMYK color spaces. The native ColorContext ColorValue array should be dynamic and not limited to 1, 3, 4 or even 8 color channels. This allows for spectral or abridged spectral color process solutions with this same architecture. Note that while the cost of the allocation is fairly

10 significant relative to the cost of a five-element array that is sometimes only partially used, this insures a consistent, coherent, and flexible solution for the future, and there is no cost is null when scRGB workflows are used. The Alpha value is separate from the ColorContext ColorValue, because it is a different concept and is treated differently in most uses.

Name	Type	Description	Other Info
InternalColor	Float Red, Float Green, Float Blue, Float Alpha	Identical structure to internal rendering color structure to optimize performance	Internal struct, this second internal struct is used to support efficient marshaling of data.
context	ColorContext	Color context providing methods relating to using color context information	Carries native color values that are synchronized to InternalColor values

15

Name	Arguments	Description	Other Info
FromProfile(...)	String	ICC or other	Public static

		profile filename-based constructor	
FromProfileAndRenderingIntent(...)	String, String	ICC or other profile filename-based constructor	Public static
FromAValues(...)	Float, float[], filename	General constructor based on alpha channel value, array of floating point values and an ICC or other profile filename	Public static
FromValues(...)	Float[], filename	Same as FromAValues(...) but alpha is assumed to be 1.0f	Public static
FromARGB(...)	byte, byte, byte, byte	Legacy sRGB constructor based on alpha, red, green and blue sRGB values	Public static, sRGB values are internally converted to scRGB for processing

Handwritten signature

FromRGB(...)	byte, byte, byte	Legacy sRGB constructor based on red, green and blue sRGB values (alpha is assumed to be 1.0f)	Public static, sRGB values are internally converted to scRGB for processing
FromScRGB(...)	Float, float, float, float	scRGB constructor based on alpha, red, green and blue scRGB values	Public static

Note that obtaining the color value of system color UI objects may be dependent on other context, as are other more advanced types of theming, and should be collected with other system metrics with the application model/theming APIs.

157

Name	Return Type	Description	Other Info
R	byte	The red sRGB value of red scRGB component of the current color	R/W
G	byte	The green sRGB value of red scRGB component of the current color	R/W
B	byte	The blue sRGB value of red scRGB component of the current color	R/W
A	byte	The alpha sRGB value of red scRGB component of the current color	R/W
ScR	float	The red scRGB value of red scRGB component of the current color	R/W
ScG	float	The green scRGB value of red scRGB component of the current color	R/W
ScB	float	The blue scRGB value of red scRGB component of the current color	R/W
ScA	float	The alpha scRGB value of red scRGB component of the current color	R/W

Note that scRGB values can range below 0.0 and above 1.0 to support extended dynamic ranges and gamuts.

Operator overrides for Color objects are context specific, because mixing and adding colors is color space dependent. For example, luminance RGB spaces are additive and linear, so typical mathematical operations are fairly intuitive, but lightness RGB spaces as well as CMYK spaces are not both linear and additive, whereby this operations result in different visual effects. Additionally, most Color operations can result in values beyond the desired gamut and thus requires gamut mapping compensation. This can be as simple as low quality clamping or it can be significantly more sophisticated.

Some animation operator overloads can be provided if they are limited specifically to the scRGB ColorContext since scRGB is based on physical light and mixes linearly and additively. CMYK, sRGB and other color spaces have very different mixing models.

Name	Return Type	Arguments	Description	Other Info
+	Color	Color, Color	Context dependent color addition	RGB context is linear wrt luminance (how photons mix) which is correct for scRGB and computer graphics
Add	Color	Color, Color	Context dependent color addition	Public
-	Color	Color, Color	Context dependent color subtraction	Public
Subtract	Color	Color, Color	Context dependent color subtraction	Public
*	Color	Color, float	Context dependent color multiply a color by a floating point value	Public
Multiply	Color	Color, float	Context dependent color multiply a color by a floating point value	Public
Equals	Bool	Color, color	Returns true if two color values are equal	Public
Equals	Bool	Object	Returns true if color	Public

147

			object is equal to current color	
=	Bool	Color, color	Returns true if two color values are equal	Public
IsEqual	Bool	Color, color	Returns true if two color values are equal	Public
!=	Bool	Color, color	Returns true if two color values are not equal	public
IsNotEqual	Bool	Color, color	Returns true if two color values are not equal	public

Similar methods as used in Multichannel colors can also be used to support HSB, YCC and YUV and similar color spaces that are closely related to sRGB or scRGB.

Name	Return Type	Arguments	Description	Other Info
Clamp	Color	Void	Clamps color values in range [0.0...1.0]	Public
GetHashCode	int	Void	Return Color hash code	Public
Name	String	Returns color name	Redundant with colorcontext call	
ToRgbColor	None	Color	Returns scRGB equivalent color of current color	Public
ToString	String	Void	Returns formatted string value of color	public
AreClose	Bool	Color, Color	Returns true if color values are close using FloatUtil function	Static

740

IsClose	Bool	Color	Returns true if color is close to current color using FloatUtil function	Public
SetRenderingIntent	Bool	String	Returns true if rendering intent for ColorContext is successful	Public

5 The float values in the rgb and argb forms based on sRGB are specified on a scale from 0.0 to 1.0. By definition, these values will never be outside of this range and so should be clamped. Conversely, scRGB based values are valid well below 0.0 and above 1.0. These values should only be clipped if the destination device cannot support the extended color gamut. This can be determined by querying the profile associated with the destination gamut. Ideally, for displays, the graphic hardware can take care of this issue using DX's gamut management functionality.

10 If the string being parsed in is invalid, the color will be initialized to ARGB=(0.0, 0.0, 0.0, 0.0)

 When persisting, the value will be written as a known color name if the color was created as a known color. If not, then the rgb(float, float, float) form will be used if alpha is 1.0. If alpha is not 1.0 then the argb(float, float, float, float) form will be used.

15 Raster graphics or imaging or pixel formats are different compared with the vector graphics solutions above. Simply stated, imaging input can be nearly anything from 1 bpp up to 128bpp with a variety of supported colors spaces from black and white to sRGB to scRGB to CMYK. Therefore, the color solution for ImageData requires a ColorContext for ImageData or pixel formats. This can be generated from
20 embedded profiles or embedded gamma and chromaticity information within standard file formats. This eliminates the need for providing gamma or other redundant properties or fields in the ImageData class.

 The MILRender code understands 32-bpc color specifications (scRGB). Input color conversion should happen above the rendering code (but not necessarily outside
25 of unmanaged code).

TTT

Animation of color values should primarily occur in a linear space. It could be a linearized HSV space or sRGB, but it is to be linear for "bounce" and other per-channel animations to make sense.

Three color fidelity modes are described, including:

5 Full - 32bpc through the system; 128bpp back-buffer / 10-bpc+ front-buffer, full 32bpc compositing.

 Hybrid - 32bpc color specifications / interpolation; dither or clamp colors to 32bpp pre-composite.

10 Legacy - 32bpc color specification - converted immediately to 32bpp; 32bpp compositing / output

 These modes will be supported with two back buffer formats, namely 128-bpp 1.0 gamma (sRGB) and 32-bpp 2.2 gamma (sRGB). Support is also provided for handling lower-bit-depth (16 and 8 bpp displays) front-buffer scenarios. The back-buffer is dithered on Present to prevent loss in the composition.

15 The Geometry class of objects can be used for clipping, hit-testing, and rendering of 2D vector-based data with the Pen and Brush. The derived Geometry classes provide more specific building and enumeration semantics. A number of shape-specific Geometry types are provided, as well as a generalized PathGeometry
20 that allows for explicit definition of more complex shaped Geometry. For those familiar with GDI+ it is most similar to GraphicsPath.

Textbook	GDI+	Present Invention
Path	GraphicsPath	PathGeometry
SubPath	GraphicsPath	PathFigure

Geometry is the abstract base class.

25

```
[TypeConverter(typeof(PathGeometryConverter))]  
public abstract class Geometry : Animatable, IDisposable  
{  
    internal Geometry();  
}
```

TSO

```

public new Geometry Copy();

// If animated, Bounds returns the "current" bounds
// This does not take into account any pen
5 public virtual Rect Bounds { get; }

// This takes the pen into account. When animations are present
// this takes the "current" values of both the geometry and pen.
10 public virtual Rect GetBounds(Pen pen);

// Returns identity if no transformation.
public Transform Transform { get; set; }

// Release resources cached by Geometry (tessellations, path data, etc)
15 public virtual void Dispose();
}

```

To set a transform that is applied to the geometry, set the Transform property.

A GeometryCollection is a collection of multiple Geometry objects that have
 20 been combined using particular boolean operations on their defined area. This object allows easier building visual combinations of Geometry objects than is possible using strictly PathFigure objects within a PathGeometry.

The combine mode enumeration directs the combination of geometry area in the collection. Boolean operations Union, XOR, Intersect are commutative and
 25 therefore apply order independent to the geometries. Complement and Exclude are not commutative and therefore are defined between the first geometry and the individual remaining geometry. In other words, an exclude combination of { g1, g2, g3 } would be applied as ((g1 exclude g2) and (g1 exclude g3)). Complement specifies that the existing region is replaced by the result of the existing region being removed
 30 from the new region. In other words, the existing region is excluded from the new region. Exclude specifies that the existing region is replaced by the result of the new region being removed from the existing region. In other words, the new region is

187

excluded from the existing region. Intersect refers to combining regions by taking their intersection, union refers to combining regions by taking the union of both, and Xor refers to combining regions by taking only the areas enclosed by one or the other region, but not both:

5

```
public enum System.Windows.CombineMode
{
    Complement,
    Exclude,
    Intersect,
    Union,
    Xor
}
```

10

15

```
public sealed class GeometryCollection : Geometry, IAddChild, IList
{
    public GeometryCollection();
    public GeometryCollection(int capacity);
    public GeometryCollection(
        CombineMode combineMode,
        params Geometry[] collection);
    public GeometryCollection(
        CombineMode combineMode,
        ICollection collection);
    public GeometryCollection(
        CombineMode combineMode,
        ICollection collection,
        Transform transform);
    public new GeometryCollection Copy();

    // Specify how to combine geometry
    [DefaultValue(CombineMode.Union)]
```

20

25

30

752

```

public CombineMode CombineMode { get; set; }

// IEnumerable
public IEnumerator GetEnumerator();

5

// ICollection
public int Count { get; }
public bool IsSynchronized { get; }
public object SyncRoot();
10 public void CopyTo(Array array, int index);

// IList
public bool IsFixedSize { get; }
bool IList.IsReadOnly { get; }
15 object IList.this[int index] { get; set; }
int IList.Add(object value);
public void Clear();
bool IList.Contains(object value);
int IList.IndexOf(object item);
20 void IList.Insert(int index, object value);
void IList.Remove(object value);
public void RemoveAt(int index);

// Extra functions
25 public int Capacity { get; set; }
public Geometry this[int index] { get; set; }
public int Add(Geometry geometry);
public bool Contains(Geometry geometry);
public int IndexOf(Geometry geometry);
30 public int IndexOf(Geometry geometry, int startIndex);
public int IndexOf(Geometry geometry, int startIndex, int count);
public int LastIndexOf(Geometry geometry);

```

```
public int LastIndexOf(Geometry geometry, int startIndex);
public int LastIndexOf(Geometry geometry, int startIndex, int count);
public void Insert(int index, Geometry geometry);
public void Remove(Geometry geometry);
5 public void AddRange(GeometryCollection geometryCollection);
public void SetRange(int index, GeometryCollection geometryCollection);
public void InsertRange(int index, GeometryCollection geometryCollection);
public void RemoveRange(int index, int count);
10 public GeometryCollection GetRange(int index, int count);

public Geometry GetOptimizedGeometry();
}
```

The `GetOptimizedGeometry()` collapses the collection of geometry where possible, the result not necessarily being a `GeometryCollection`. This might include optimizations such as combining adjacent rectangle geometry into a single rectangle geometry, performing a boolean operation between adjacent path geometry to create a new path geometry, or flattening a `GeometryCollection` with the same combination mode. In situations where the geometry is used in many different contexts this provides a performance improvement in processing and storage.

The following sample demonstrates markup that uses a `GeometryCollection`:

```
<Path Fill="Red">
  <Path.Data>
    <GeometryCollection Transform="translate(425 25)" CombineMode="Xor">
      <RectangleGeometry Rect="0 0 100 100" />
      <RectangleGeometry Rect="50 50 100 100" />
    </GeometryCollection>
  </Path.Data>
30 </Path>
```

184

The PathGeometry is a collection of PathFigure objects. Each of the PathFigure objects is composed of one or more PathSegment objects which actually define their shape. The filled area of the PathGeometry is defined by taking the contained PathFigures that have their Filled property set to true and applying the FillRule to determine the enclosed area. FIG. 13 represents PathGeometry Object Relationships.

The FillRule enumeration specifies how the intersecting areas of Figure objects contained in a Geometry are combined to form the resulting area of the Geometry:

```

10 public enum System.Windows.Media.FillRule
    {
        EvenOdd,
        NonZero
    }
15 {
    public PathGeometry();
    public PathGeometry(params PathFigure[] figures);
    public PathGeometry(ICollection figureCollection);
    public PathGeometry(
        ICollection figureCollection,
20     FillRule FillRule,
        Transform transform);
    public new PathGeometry Copy();

    [DefaultValue(FillRule.EvenOdd)]
25     public FillRule FillRule { get; set; }

    // This method will add the Figures from the specific
    // Geometry to this PathGeometry, but will use the
    // current FillRule instead of the geometry's FillRule.
30     public void AddGeometry(Geometry geometry);
    public void AddFigure(PathFigure figure);
    public void AddPointAndTypes(

```

795

5

```
Point[] points,
byte[] types);

public PathFigureCollection Figures { get; set; }

public PathGeometry GetWidenedPathGeometry(Pen pen);
public PathGeometry GetOutlinedPathGeometry();
}
```

10

The EvenOdd rule determines the "insideness" of a point on the canvas by drawing a ray from that point to infinity in any direction and then examining the places where a segment of the shape crosses the ray. Starting with a count of zero, add one each time a Segment crosses the ray from left to right and subtract one each time a path segment crosses the ray from right to left. After counting the crossings, if the result is zero then the point is outside the path. Otherwise, it is inside.

15

The NonZero rule determines the "insideness" of a point on the canvas by drawing a ray from that point to infinity in any direction and counting the number of path Segments from the given shape that the ray crosses. If this number is odd, the point is inside; if even, the point is outside.

20

To convert other geometry types into a path geometry for inclusion with other figures, AddGeometry method is used. This adds a figure which geometrically matches the input geometry. For non-animated geometry the match is exact, while animated geometry may be lossy in conversion. The reason for lossy conversion is the animated parameters of the input geometry do not match the form that fits a segment.

25

Geometry	Lossless/Lossy	Figures
LineGeometry	Lossy w/animation	StartSegment and LineSegment
RectangleGeometry	Lossy w/animation	PolyLineSegment
EllipseGeometry	Lossy w/animation	ArcSegment and ArcSegment
GeometryCollection	Lossy w/animation	Many kinds of segments.
PathGeometry	Lossy w/animation	All except Arc and Quadratic

156

		segments.

Enumeration and structure of resultant PathGeometry is not guaranteed to match the input geometry exactly.

A figure collection, is a collection of PathFigure objects and the primary contents defining a PathGeometry:

```

5  public sealed class PathFigureCollection : Animatable, IAddChild, IList
    {
        public PathFigureCollection();
        public PathFigureCollection(int capacity);
10  public PathFigureCollection(params PathFigure[] figures);
        public PathFigureCollection(ICollection figureCollection);
        public new PathFigureCollection Copy();

        // IEnumerable
15  public IEnumerator GetEnumerator();

        // ICollection
        public int Count { get; }
        public bool IsSynchronized { get; }
20  public object SyncRoot();
        public void CopyTo(Array array, int index);

        // IList
        public bool IsFixedSize { get; }
25  bool IList.IsReadOnly { get; }
        object IList.this[int index] { get; set; }
        int IList.Add(object value);
        public void Clear();
        bool IList.Contains(object value);
30  int IList.IndexOf(object item);
        void IList.Insert(int index, object value);
    }

```

24

1270
makes deeper copies as necessary. Such dotting-down provides significant performance improvements.

In accordance with an aspect of the present invention, another property on the changeable (generally invisible to applications) is that a changeable has a changed event (of type EventHandler). When a property of a Changeable changes, a Changed delegate on that changeable gets invoked, with the changing changeable as the sender. The act of making a shallow clone through dotting-down pushes the changed event handlers down into the shallow clone. This allows subsequent changes to occur on deeper elements and have the proper event handlers set up. Note that the Changed event also exists so that clients other than the property system can use this system and register for notifications.

Modifications to a changed event handler propagate down to sub-objects. In addition, modifications to a changeable itself involving other changeables (e.g., adding a changeable sub-object to a changeable object, removing one, and so forth) results in the containing changeable's event handlers being removed from the old ones and pushed into the new ones, recursively.

FIGS. 18-23 demonstrate how shallow-cloning and dotting down works, based on the following code for this example:

```
20 LinearGradientBrush lgb = new LinearGradientBrush();  
   lgb.Stops.Add(new GradientStop(Colors.Red,0.3));  
   lgb.Stops.Add (new GradientStop(Colors.Blue,0.2));  
   lgb.Stops.Add (new GradientStop(Colors.Green,0.1));  
  
25 Btn1.Background = lgb;  
   Btn2.Background = lgb;
```

As shown in FIG. 18, both Button1 and Button2 point to the same linear gradient brush 1802, which has a stops node 1804 and the color and position properties for the specified stops arranged hierarchically below. Consider the code:

```
Btn1.Background = Btn1.Background.Copy();
```

171

Executing this code results in a copy 1902 of the linear gradient brush 1802 being made and pointed to by Button1, as shown in FIG. 19.

Executing the code:

```
5 LinearGradientBrush lgb = ((LinearGradientBrush)Btn1.Background);  
lgb.Stops[1].Color = Colors.Orange;
```

provides access to a Changeable-valued property of a Changeable object that has IsChangeable==true, meaning that what is retrieved is guaranteed to be writable. As
10 generally represented in FIGS. 20-22, executing this code results in (1) another stops node 2004 inserted into the hierarchy that points to each of the individual stops (FIG. 20); (2) a copy 2110 (FIG. 21) of the second stop node (stop node [1], labeled 2010 in FIGS. 20 and 21, that has the "blue" property below) to be made, such that the parent of this copy, the previously copied stops node 2004, has as its child this copy 2110
15 (instead of the original stop node 2010 for the blue property); and (3) changes the blue property of this node 2110 to orange, as represented in FIG. 20. Note that Orange is a value type denoted by the diamond shapes in the figures, and subsequent changes result in no allocation, such as the change to the color Red in FIG. 23.

When in an unchangeable state, a changeable object may be read from and
20 written to from any context. If in a changeable state, then a UiContext, determined at construction time, may be used to associate with the changeable object so as to only allow access from that context. If MakeUnchangeable is later called, the context goes to null. Also, anytime a Copy() of a changeable is made, the new copy gets the UiContext from the caller, not from the context of the source changeable. The API
25 provides a UiContext get-only property on changeable which is null when immutable. This property is public so that applications can tell whether a given object may be accessed.

Changeable objects constructed with null passed into the constructor will be defined with a null UiContext. If a changeable object has a null UiContext and the
30 IsChangeable property is set to true, then the application needs to manage any possible thread contention issues that may arise. In this case, the system does not prevent concurrent access from multiple contexts.

26

A situation can arise whereby a changeable object is attempting to embed another changeable object into it (e.g., setting a gradient stop in a linear gradient brush), and the UIContexts do not match. For example, consider a LinearGradientBrush lgb having a UIContext of A, while a GradientStop gs has both a
 5 UIContext of B, and a StatusOfNextUse equal to ChangeableReference. Attempting to set gs into lgb will result in an exception being raised, as this is an attempt to mix UIContexts, which is disallowed.

When a change is made to a changeable object, the Changed event is raised, and that changeable is provided as the sender object for the event handler. However,
 10 there are situations when sending the object that actually changed is not desirable, and wherein having a different object as the sender would be more helpful. An example of this is with animated objects, where an animation (itself a changeable) holds onto a timeline (sometimes referred to as a clock) that describes its animation behavior. Events such as Pause() occur on the timeline, and not the animation, but generally
 15 applications want to know that the animation has paused.

Various solutions are possible, such as to fire changed events up the chain of changeables. This presents a number of problems, including deciding where to stop, causing notification storms with many more events fired even when nothing is receiving and/or using the events, and that changeables, by design, do not know their
 20 parents, but rather generally only know what to notify in the event of a change. Having a scheme in which changeables tracked their parents would require additional storage and bookkeeping. Notwithstanding, such a solution may be implemented.

Another solution is to implement a changeable such that the sender is the changeable object, and not the inner changeable that actually changed. The
 25 PropagateEventHandler is implemented to not push down the handler it receives, but rather store that handler off, create a new, local handler that, when invoked, calls the stored handler, but with the changeable as the sender argument. The new local handler is pushed down to PropagateEventHandler on the changeable children. Note that this technique intercepts every event handler, requiring that PropagateEventHandler is
 30 correctly handled when called with 'false' (when handlers are to be removed), thus requiring that bookkeeping be performed.

Note that this solution does not have an explicit BeginChange/EndChange scoping mechanism, making it more straightforward and robust in the face of exceptions (since there is no modality involved, and no EndChange() that can get skipped passed by an unexpected exception). However, Begin/EndChange existed to allow “getters” of objects to not have to make shallow clones of the values they are getting, when those clones were frozen and the system was not in write mode. Otherwise, those frozen values, in write mode, get a shallow clone made of them. As a result, the tree is exploded out more frequently than with Begin/EndChange, and can do so when there is absolutely no setting involved, just getting. Notwithstanding, if a getter is being invoked on an unchangeable value to begin with, it will not make a shallow clone (note this is distinct from the getter being invoked on a changeable value, and the value that is being obtained via “get” is unchangeable, which is where the clone operation happens).

By way of example, if accessing Btn.Background.Opacity, and Btn.Background is not changeable (e.g., as by default), then a copying is not made. Instead, the copying will occur when “Btn.Background = Btn.Background.ChangeableValue” or the like takes place, meaning that the copying expense only occurs if used. In other words, if an intent to change the value is not expressed, then arbitrary “getting” does not incur the copy cost. Note that if values maintain a notion of their “last created clone” then that clone can be handed out on use of the object, as long as the object has not been changed since the clone was made (such changes just result in letting go of that cached clone). This allows more sharing. Further, note that the control implementers are not as unduly burdened by participating in this pattern as it is for the pattern to be useful to users. Similarly, if type extensibility is provided, writing MediaTypes should not be overly complex.

The control implementer is presented with the same model for dealing with a Changeable as with any other value. For instance, the following code provides a Grid control with an AlternateBrush property of type Brush:

```
30 public static readonly DependencyProperty AlternateBrushID = ...;
    private Brush _alternateBrush;
    public Brush AlternateBrush
```

628

174

```

{
    get
    {
        return ComputeValue(AlternateBrushID,
5         ref _alternateBrush);
    }

    set
    {
10     WriteLocal(AlternateBrushID, value);
    }
}

```

Note that this is identical to a general property participating in the property system. That is because WriteLocal will do special handling for deep properties that derive from the Changeable class.

The changeable type implementer needs a one line preamble and a one line postscript on anything that modifies a changeable (e.g., properties). Also, a simple, one line preamble is needed on objects that access state of a changeable (e.g., property getters). Needed are implementations of CloneCore(), MakeUnchangeableCore(), PropagateEventHandlerCore(), PropagateEventHandlers() (note that the last three are only needed for types that have other Changeables as properties), and also needed is a typesafe-wrapper for Copy().

The following examples come from the referenced prototype, including an (artificial) example of a GradientStop which is a simple changeable type (which is simplified in that none of its subtypes are changeables). Note that in practice, very few changeables will be this simplified, in that anything that contains an animation collection (itself a changeable) will be more complex:

```

30 public class GradientStop : Changeable
{
    public GradientStop()

```

```

{
}

public GradientStop(Color color, float location)
5 {
    _color = color;
    _location = location;
}

10 public Color Color
{
    get
    {
        ReadPreamble();
15     return _color;
    }
    set
    {
        // This is the pattern for setting simple value types.
20     WritePreamble();
        if (_color != value)
        {
            _color = value;
            WritePostscript();
25     }
        }
    }

    public float Location
30 {
    get
    {

```

```

        ReadPreamble();
        return _location;
    }
    set
5    {
        // This is the pattern for setting simple value types.
        WritePreamble();
        if (_location != value)
        {
10            _location = value;
            WritePostscript();
        }
    }
}
15
// Create a new, typesafe version of Copy().
public new GradientStop Copy()
{
    return (GradientStop)base.Copy();
20 }

protected override Changeable CloneCore(bool shallowClone)
{
    // Shallow and deep are treated the same here when
25    // all members are value types
    return new GradientStop(_color, _location);
}

private Color _color;
30 private float _location;
}

```

The following is an example of a `LinearGradientBrush` which is a more complex changeable type (because some of its subtypes, namely `GradientStops`, are themselves changeable):

```

5  public class LinearGradientBrush : Brush
   {
       public LinearGradientBrush()
       {
       }

10     public LinearGradientBrush(GradientStop stop1, GradientStop stop2,
                                   double opacity)
        : base(opacity)
        {
15         // Assigning through the properties results in "using" the
           // parameters.
           Stop1 = stop1;
           Stop2 = stop2;
        }

20     public GradientStop Stop1
        {
            get
            {
25                 _stop1 = (GradientStop)EmbeddedChangeableGetter(_stop1);
                return _stop1;
            }
            set
            {
30                 if (_stop1 != value)
                    {
                        _stop1 = (GradientStop)EmbeddedChangeableSetter(_stop1, value);
                    }
            }
        }
    }

```

```

        WritePostscript();
    }
}

5
public GradientStop Stop2
{
    get
    {
10        _stop2 = (GradientStop)EmbeddedChangeableReader(_stop2);
        return _stop2;
    }
    set
    {
15        if (_stop2 != value)
        {
            _stop2 = (GradientStop)EmbeddedChangeableWriter(_stop2, value);
            WritePostscript();
        }
20    }
}

// In place modification to r/o (or check whether it can be done).
// Helpers check if arguments are null.
25 protected override bool MakeUnchangeableCore(bool checking)
{
    return Changeable.MakeUnchangeable(_stop1, checking);
    && Changeable.MakeUnchangeable(_stop2, checking);
}

30
// Propagation of event changes
protected override void PropagateEventHandlerCore(EventHandler handler,

```

```

                                bool adding)
    {
        Changeable.ModifyHandlerIfChangeable(_stop1, handler, adding);
        Changeable.ModifyHandlerIfChangeable(_stop2, handler, adding);
5    }

    protected override Changeable CloneCore(bool shallowClone)
    {
        // Always use member fields, not properties, for cloning.
        // Otherwise you inadvertently cause extra shallow copying.
10    if (shallowClone)
        {
            return new LinearGradientBrush(_stop1, _stop2, _opacity);
        }
15    else
        {
            return new LinearGradientBrush(
                (GradientStop)Changeable.CloneDownToUnchangeable(_stop1),
                (GradientStop)Changeable.CloneDownToUnchangeable(_stop2),
20                _opacity);
        }
    }

    private GradientStop _stop1;
25    private GradientStop _stop2;
}

```

30 The changeable object model is divided into a public part, and a part which is what extenders and hosters see. Note again, though, that things are straightforward for the component writer that uses these types.

```

// PUBLIC API. Everyone expected to use this portion of the API.
// Anything outside of this section will be hidden

```


180

```

// from the editors.
public abstractclass System.Windows.Changeable
{
    public bool IsChangeable { get; } // defaults to true
5    public Changeable Copy();
    public void MakeUnchangeable();
    public bool CanMakeUnchangeable { get; }
    public UseStatus StatusOfNextUse { get; set; } // defaults to false
    public event EventHandler Changed { add; remove; }
10    public UIContext UIContext { get; } // null if IsChangeable==false

    public bool AllowChangeableReferenceOverride { get; set; }
        // defaults to false

15    // protected stuff shown below.
}

Public enum System.Windows.UseStatus
{
20    Unchangeable,
    ChangeableCopy,

    [EditorState.Advanced]
    ChangeableReference
25 }

```

```

public class System.Windows.Changeable
{
    // COMPONENT WRITER API. This is for use by Component Writers

```

187

```

// (Note: there is nothing that component writers need to know about)

// EXTENDER APL. This is for use by those who write new Changeables

protected Changeable(); // use UIContext.CurrentContext
protected Changeable(UIContext context);

// Subtypes implement this to provide cloning as described above.

protected abstract Changeable CloneCore(bool shallowClone);

// Subclasses implement to freeze their complex subtypes (when
// 'checking' is false, or to check whether they can (when
// 'checking' is true. Note that when checking is false, the return
// boolean doesn't really make sense, but implementers are to return
// 'true' in this case, so that calling code (like the above example)
// will work correctly. If a call is made with 'true' that cannot
// actually make the value unchangeable, then the implementation should
// raise an exception.
// The default implementation returns true in these cases.
protected virtual bool MakeUnchangeableCore(bool checking);

// Subclasses implement to push down changes to event handlers
// that happen up above (default impl is empty). If adding is true
// then the handler should be added to sub-changeables, else it
// should be removed from them as described above
protected virtual void PropagateEventHandler(EventHandler handler,
                                             bool adding);

// Subclasses implement to validate that their in a good state. They

```

af

182

```

// throw a meaningful exception if not. This will be called on
// WritePostscript(), EmbeddedChangeableWriter().
// Default impl is empty.
protected virtual void ValidateObjectState();

// Called when there's a change to the value. Default implementation
// invokes registered Changed handlers.
protected virtual void OnChanged();

// Call from anything that reads an embedded changeable out of
// a changeable (Reader) or writes an embedded changeable into
// a changeable (Writer). The primary uses of these are for property
// sets and gets. See examples above.
protected Changeable EmbeddedChangeableReader(Changeable property);
protected Changeable EmbeddedChangeableWriter(Changeable originalChangeable,
                                                Changeable newChangeable);

// Call from simple property setters and getters, or anything that
// access non-Changeable state on a Changeable. See examples above.
protected void ReadPreamble();
protected void WritePreamble();

// Call from both simple and complex property setters and anything else
// that sets state on a Changeable. See examples above.
protected void WritePostscript();

//// Static helper methods

// Create a deep, unchangeable clone of the provided Changeable, but stop at nodes
// that are already unchangeable (since those can just be "cloned" by handing
// out references to them). If null is passed in, null is returned
static protected Changeable CloneDownToUnchangeable(Changeable changeable);

```

97

123

(e.g., of an operating system). One such multiple-level graphics processing system comprises two components, including a tick-on-demand or slow-tick high-level component, and a fast-tick (e.g., at the graphics hardware frame refresh rate) low-level component. In general, the high-level, less frequent component performs

5 computationally intensive aspects of updating animation parameters and traversing scene data structures, in order to pass simplified data structures to the low-level component. The low-level component operates at a higher frequency, such as the frame refresh rate of the graphics subsystem, to process the data structures into constant output data for the graphics subsystem. The low-level processing includes
10 interpolating any parameter intervals as necessary to obtain instantaneous values to render the scene for each frame of animation.

Top level MIL objects include a visual tree, which is an object that contains the main content to be drawn. Controls will derive from visuals of the tree directly. Visuals are device and parent context independent. A render target is the device to
15 which the visual is drawn. This object (e.g., screen) may have its own dirty or invalidation mechanism. Various render targets include a screen in a window, a Printer, a Metafile, a Surface, a streaming media file (for example, DVD) and a "Sub-window" which is a part of the scene that is drawn separately from the rest of the scene. Other drawing related objects include a Visual Renderer, comprising an object
20 that is configured to draw a visual tree onto a render target, and a Display Scheduler object that knows when to draw the visual tree on to the render target. A Time Manager is a context object for a set of timing nodes, and is the object that the scheduler calls tick on.

A Visual API is provided, which is essentially a starting point for drawing via
25 the media integration layer, and comprises multiple types of objects, including a VisualManager object, which connects a Visual Tree to a medium. The different types of VisualManagers (e.g., Screen, Printer, and Surface) are responsible for rendering a Visual Tree to their particular medium. A visual is where the programmer does the drawing; it is a node in the visual tree and provides a place for a program to draw.

30 The DrawingContext APIs presents a context-based programming model for how to construct visual content that populates a Visual or are rendered to an ImageData. DrawingContext classes are provided, as well as the classes and

entrypoints necessary to acquire a DrawingContext and enumerate the visual content in a RetainedVisual/DrawingVisual.

To enable mutability, there is provided a single set of types that derive from a common Changeable base class. Any type for which mutability is desired change may
5 derive from the Changeable class. For example, in a graphics programming, the object model includes Brushes, Pens, Geometries, FloatAnimations, GradientStops, Segments, and so forth. An IsChangeable property specifies whether the changeable object can be modified or not, depending on its current value, which defines a state.

A brush is an object that represents a method to fill a plane. In addition to
10 being able to fill a plane in an absolute way, brushes of the media integration layer are also able to adapt how they fill the plane relative to the size of the object that they are filling. Examples of types of brushes include SolidColorBrush, VisualBrush (which can reference a vector graphics resource/Visual), DrawingBrush, LinearGradient, RadialGradient, ImageBrush and NineGridBrush. Certain brush objects will have an
15 idea of how they relate to the coordinate system when they are used, and an idea of how they relate to the bounding box of the geometry with which they are used. This size is based on from the object that the brush is filling. Certain type of brushes (for example VisualBrush) also may be tiled to create programmer defined patterns. The Brush base class has a Transform, a general opacity, and a blend mode. Brush (and
20 other object resources in Vector Graphics and the MIL API) objects are Changeables and are writable after they have been created, and follow the general Changeable pattern for how they behave after they are used in qualified use.

A Geometry class of objects can be used for clipping, hit-testing, and rendering of 2D vector-based data with the Pen and Brush. The derived Geometry classes
25 provide more specific building and enumeration semantics. A number of shape-specific Geometry types are provided, as well as a generalized PathGeometry that allows for explicit definition of more complex shaped Geometry. Geometry is an abstract base class. A GeometryCollection is a collection of multiple Geometry objects that have been combined using particular CombineMode operations on their
30 defined area. This object allows easier building visual combinations of Geometry objects than is possible using strictly PathFigure objects within a PathGeometry.

ImageSource is an abstract class, comprising a basic building block for imaging. An ImageSource conceptually represents a single, constant set of pixels at a certain size and resolution. For example, an ImageSource may be a single frame in an image file that a Decoder could provide, or it may be the results of a transform that
5 operates on a certain ImageSource of its own. An ImageSource is changeable, not because its own properties can be changed, but because the properties of its sub-classes can potentially be changed.

A Transform class of objects is provided for scaling, rotating, translating and skewing vector and raster graphics. The derived Transform classes provide friendly
10 usage and enumeration semantics.

Effects provide a means to alter the visual contents of a scene in a rendering-centric manner. For example, ImageEffectss (raster-based bitmap effects) operate on the image-based, fully composited representation of a portion of a scene. Effects are broken down into various types including ImageEffectss, BlendModes and
15 VectorEffects. ImageEffectss can be used in a retained-mode scene by applying it to a sub-graph or an Element, or it can be used in the standalone image pipeline. BlendModes are a specific form of image-based effects, and can be applied to the retained mode scene in generally the same manner as ImageEffectss. Blend modes perform a combination of the source and destination colors as the source is
20 composited, e.g., multiply or add.

Hit testing is used to pick visuals in a scene, and operates by starting from the top of the control tree, and returning a control or set of controls by a point or geometry. A control can define whether it is hit or not with support services including rendered geometry, bounding box, out-of-band geometry (hit region), image opacity or
25 mask, and its own logic. The control can return specific hit-related data on hit. The hit test mechanism can filter hit test results in an efficient manner. The hit test walk is a deep right to left walk of the visual tree, with hits reported through a callback in z-order, top-to-bottom fashion. When descending, the hit tester views the filtering in terms of element level relationships, for example, a canvas with shapes, or a dock
30 panel with an inner canvas. When a hit occurs, the hit tester can either continue processing further hits (if any), or stop.

An animation system is provided, comprised of a timing control engine and a set of animation objects. The timing engine is a service that can be used by any objects that exhibit time-varying behaviors, e.g., animations and audio or video media objects. Animation objects implement a set of functions that map time spans into other data types, which are then used as inputs into other higher-level objects. Graphical animation is achieved by associating an animation collection with a rendering operation. Each animation used in a rendering operation may be run on a separate clock, referred to as a called a "timeline." Multiple timelines can be organized in a timing tree to support hierarchical timing. Once an animated primitive is drawn and animation parameters specified, the low-level rendering system takes care of redrawing the scene at regular intervals. Each time a frame is rendered the current value of the animations involved in the scene is computed, based on the elapsed time (in most cases measured by the system clock), and then the animated primitives are redrawn.

Various primitive types, color features and media support is also provided via the MIL. MediaData can be used to play any audio/video content.

Other benefits and advantages will become apparent from the following detailed description when taken in conjunction with the drawings, in which:

20

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a block diagram representing an exemplary computer system into which the present invention may be incorporated;

FIG. 2 is a block diagram generally representing a graphics layer architecture into which the present invention may be incorporated;

25

FIG. 3 is a representation of a scene graph of visuals and associated components for processing the scene graph such as by traversing the scene graph to provide graphics commands and other data in accordance with an aspect of the present invention;

30

FIG. 4 is a representation of a scene graph of validation visuals, drawing visuals and associated Instruction Lists constructed in accordance with an aspect of the present invention;

5/6

FIG. 5 is a representation of a visual class, of an object model, in accordance with an aspect of the present invention;

FIG. 6 is a representation of various other objects of the object model, in accordance with an aspect of the present invention;

5 FIG. 7 is a representation of a transform class hierarchy, in accordance with an aspect of the present invention;

FIGS. 8 and 9 are representations of transformations of a visual's data in a geometry scale and a non-uniform scale, respectively, in accordance with an aspect of the present invention;

10 FIG. 10 is a representation of geometry classes of the object model, in accordance with an aspect of the present invention;

FIG. 11 is a representation of a PathGeometry structure, in accordance with an aspect of the present invention;

15 FIG. 12 is a representation of a scene graph of visuals and Instruction Lists showing example graphics produced by the primitives, in accordance with an aspect of the present invention;

FIG. 13 is a representation of brush classes of the object model, in accordance with an aspect of the present invention;

20 FIG. 14 is a block diagram generally representing a changeable architecture in which requests are processed by a state machine to control a type's mutability, in accordance with an aspect of the present invention;

FIGS. 15-17 are state diagrams representing how property states control the behavior of mutable types, in accordance with an aspect of the present invention;

25 FIGS. 18-23 are hierarchical representations of objects in an example scene graph showing how the properties control state transitions and cloning behaviors for example code, in accordance with an aspect of the present invention;

FIG. 24 is a representation of rendered graphics resulting from data in a linear gradient brush object, in accordance with an aspect of the present invention;

30 FIG. 25 is a representation of rendered graphics resulting from data in a radial gradient brush object, in accordance with an aspect of the present invention;

FIG. 26 is a representation of rendered graphics resulting from having various stretch values, in accordance with an aspect of the present invention;

FIG. 27 is a representation of rendered graphics resulting from having various tile values, in accordance with an aspect of the present invention;

FIG. 28 is a representation of a rendered nine grid brush object in accordance with an aspect of the present invention;

5 FIGS. 29-40 are graphical representations of example timelines used in animation, in accordance with an aspect of the present invention;

FIG. 41 is a representation of an example three-dimensional image constructed via a three-dimensional visual in accordance with an aspect of the present invention; and

10 FIG. 42 is a representation of three-dimensional concepts for providing three-dimensional support in accordance with an aspect of the present invention.

DETAILED DESCRIPTION

EXEMPLARY OPERATING ENVIRONMENT

15 FIGURE 1 illustrates an example of a suitable computing system environment 100 on which the invention may be implemented. The computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither
20 should the computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100.

The invention is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use
25 with the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, tablet devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

30 The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures,

109

and so forth, which perform particular tasks or implement particular abstract data types. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules
5 may be located in both local and remote computer storage media including memory storage devices.

With reference to FIG. 1, an exemplary system for implementing the invention includes a general purpose computing device in the form of a computer 110. Components of the computer 110 may include, but are not limited to, a processing unit
10 120, a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The system bus 121 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard
15 Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, Accelerated Graphics Port (AGP) bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus.

The computer 110 typically includes a variety of computer-readable media.
20 Computer-readable media can be any available media that can be accessed by the computer 110 and includes both volatile and nonvolatile media, and removable and non-removable media. By way of example, and not limitation, computer-readable media may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable
25 media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or
30 other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by the computer 110. Communication media typically embodies computer-readable instructions, data structures, program

209

modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection; and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of the any of the above should also be included within the scope of computer-readable media.

The system memory 130 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 131 and random access memory (RAM) 132. A basic input/output system 133 (BIOS), containing the basic routines that help to transfer information between elements within computer 110, such as during start-up, is typically stored in ROM 131. RAM 132 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 120. By way of example, and not limitation, FIG. 1 illustrates operating system 134, application programs 135, other program modules 136 and program data 137.

The computer 110 may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, FIG. 1 illustrates a hard disk drive 141 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156 such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150.

38

191

The drives and their associated computer storage media, discussed above and illustrated in FIG. 1, provide storage of computer-readable instructions, data structures, program modules and other data for the computer 110. In FIG. 1, for example, hard disk drive 141 is illustrated as storing operating system 144, application programs 145, other program modules 146 and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136, and program data 137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given different numbers herein to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 110 through input devices such as a tablet (electronic digitizer) 164, a microphone 163, a keyboard 162 and pointing device 161, commonly referred to as mouse, trackball or touch pad. Other input devices (not shown) may include a joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. The monitor 191 may also be integrated with a touch-screen panel 193 or the like that can input digitized input such as handwriting into the computer system 110 via an interface, such as a touch-screen interface 192. Note that the monitor and/or touch screen panel can be physically coupled to a housing in which the computing device 110 is incorporated, such as in a tablet-type personal computer, wherein the touch screen panel 193 essentially serves as the tablet 164. In addition, computers such as the computing device 110 may also include other peripheral output devices such as speakers 195 and printer 196, which may be connected through an output peripheral interface 194 or the like.

The computer 110 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 110, although only a memory

ST

142

storage device 181 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local area network (LAN) 171 and a wide area network (WAN) 173, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

5 When used in a LAN networking environment, the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet. The
10 modem 172, which may be internal or external, may be connected to the system bus 121 via the user input interface 160 or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 1 illustrates remote application programs 185 as residing on
15 memory device 181. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

MEDIA INTEGRATION LAYER (MIL)

20 One aspect of the present invention is generally directed to an architecture referred to as the media integration layer (MIL), that includes an immediate mode graphics application programming interface (API), a screen-partitioning data structure and API that also plays a role in caching drawing instructions, a set of control level objects, and a markup language. In general, the architecture allows program code,
25 such as an application or operating system component, to communicate drawing instructions and other information (e.g., image bitmaps) to graphics components in order to render graphical output on the system display. An aspect of the present invention provides a number of defined functions and methods, e.g., in the form of APIs to an object model, that enable programs to populate a scene graph with data
30 structures, instruction lists (drawing primitives / commands), and other graphics-related data. When processed, the scene graph results in graphics being displayed on the screen.

In accordance with an aspect of the present invention, the MIL is a composited system which works in layers. Each item to be composited is conceptually drawn into a bitmap that has a full alpha channel. That alpha channel is then used when compositing that bitmap to the back buffer. The individual objects are composited from back to front. Note that while this is a conceptual model, in actuality, the system understands when a full intermediary bitmap surface is not needed and composites directly to the back buffer or other backing surface. The system also can understand incremental changes and do minimal repaint.

FIG. 2 represents a general, layered architecture (the MIL) 200 into which the present invention may be implemented. As represented in FIG. 2, program code 202 (e.g., an application program or operating system component or the like) may be developed to output graphics data in one or more various ways, including via imaging 204, via vector graphic elements 206, and/or via function / method calls placed directly to a visual application programming interface (API) layer 212. In general, imaging 204 provides the program code 202 with a mechanism for loading, editing and saving images, e.g., bitmaps. As described below, these images may be used by other parts of the system, and there is also a way to use the primitive drawing code to draw to an image directly. Vector graphics elements 206 provide another way to draw graphics, consistent with the rest of the object model (described below). Vector graphic elements 206 may be created via a markup language, which an element / property system 208 and layout system 210 interprets to make appropriate calls to the visual API layer 212. Vector graphic elements 206, along with the element / property system 208 and layout system 210, are described in the aforementioned copending patent application Serial No. 10/401,717.

Thus, the MIL 200 includes different levels that a programmer can program to, including imaging 204, which is the pipeline for loading, editing and saving images. These images can be used in the rest of the system as desired. Furthermore, there is a way to use the primitive drawing code to draw to an image directly.

Another level comprises the Visual API 212, which is an API that primarily provides access to the data structure 216 for organizing items to be drawn. Each of those items can be loaded with drawing instructions that can be cached by the system.

There is a variety of ways to specify this data structure and what is drawn; inside of a typical MIL-aware application, this API may be used from within a layout system 210.

A third level for programming comprises the vector graphics elements level 206, which is a markup language to describe and draw graphics in a way that is consistent with the rest of the control/Element object model. Vector Graphics Elements expose the graphics system through the element system. This includes a set of elements for rendering and a set of properties that work on any arbitrary element. In one implementation, there are two subsets, including Element level Vector Graphics, which gets parsed into and creates elements, and Resource level vector graphics, which gets parsed and stored in an efficient manner. Element level object model refers to operating at the higher level control world that brings in the element tree, the property system and the layout system 210. With respect to parsing, many dynamic properties at the Element level are of MIL types. In general, the markup is resolved to objects, in which an XML scheme for the XAML markup is usually declared at top of a markup file as follows:

```
<Canvas xmlns="http://schemas.microsoft.com/2003/xaml"
xmlns:def="Definition"
def:Class="Surfin.ChannelBrowser"
def:CodeBehind="ChannelBrowser.xaml.cs"
ID="mainCanvas" Background="Black" Width="100%"
Height="100%"
Loaded="PageLoaded">
```

When <Path > tag is used for example, the parser uses the schema to look up the relevant namespace (for example, System.Windows.Shapes) to resolve and build the object. As generally described in the aforementioned copending patent application Serial No. 10/401,717, the parser relies on type converters to convert from a string to an instance of a MIL object. These types which require complex syntax have each writeable CLR (common language runtime) property exposed as an optional XML attribute, which is parsed in the same manner as a dynamic property. Some types (notably Brushes) can be parsed in either simple or complex form.

304

It should be noted that the function calls directed to any of these layers or to any of these classes may be handled directly or indirectly. For example, a request handler may comprise middleware code that converts the requests received in one operating system to API calls handled by another operating system. Thus, as used
5 herein, functions are called by programs that "cause" the requested behavior to occur, regardless of where the actual processing occurs or where the data structures and classes are provided.

As will be understood, and also represented in FIG. 2, an animation system 220 pervades the entire API. As described herein, animate values can be passed essentially
10 anywhere, including at the element property level 208, inside of the Visual API 212 and in any of the other resources. The timing system is also exposed at both the element and visual levels.

In one implementation, the graphics layer architecture 200 includes a high-level composition and animation engine 214, which includes or is otherwise associated
15 with a caching data structure 216. The caching data structure 216 contains a scene graph comprising hierarchically-arranged objects that are managed according to a defined object model, as described below. In general, the visual API layer 212 provides the program code 202 (and the layout system 210) with an interface to the caching data structure 216, including the ability to create objects, open and close
20 objects to provide data to them, and so forth. In other words, the high-level composition and animation engine 214 exposes a unified media API layer 212 by which developers may express intentions about graphics and media to display graphics information, and provide an underlying platform with enough information such that the platform can optimize the use of the hardware for the program code. For example,
25 the underlying platform will be responsible for caching, resource negotiation and media integration.

In one implementation, the high-level composition and animation engine 214 passes an instruction stream and possibly other data (e.g., pointers to bitmaps) to a fast, low-level compositing and animation engine 218. In general, the low-level
30 composition and animation engine / renderer 218, provides a set of systems that manages the actual drawing and compositing onto the screen. Note that as used herein, the terms "high-level" and "low-level" are similar to those used in other



computing scenarios, wherein in general, the lower a software component is relative to higher components, the closer that component is to the hardware. Thus, for example, graphics information sent from the high-level composition and animation engine 214 may be received at the low-level compositing and animation engine 218, where the
 5 information is used to send graphics data to the graphics subsystem including the hardware 222. Note that the present invention may be extended to multi-tiered composition beyond two layers.

Further, note that in order to facilitate resolution and device independent user interfaces, the concept of a pixel is not exposed as a fundamental unit in the main
 10 APIs. Instead, the initial coordinate system is set so that one unit in that coordinate system is equal to 1/96th of an inch. This may be referred to as a dip, or alternatively a px, even though on some systems (such as a high resolution monitor or a printer) it does not map to a pixel. In Length, the dip unit is translated directly to one user unit. The multiplier between other physical units (in, cm, pt, and the like) is locked at
 15 1/96th of an inch. This means that if a scale transform is used it will affect everything that is drawn, even if it is specified in physical issues. The value of 1/96th of an inch was chosen so that a default user unit is equivalent to a pixel on contemporary displays with default settings. Note that a mechanism may be provided to provide hints to the layout system and other user code so that they can optimize for the output resolution of
 20 the device to which they are rendering.

The high-level composition and animation engine 214 in conjunction with the program code 202 builds a scene graph to represent a graphics scene provided by the program code 202. For example, each item to be drawn may be loaded with drawing instructions, which the system can cache in the scene graph data structure 216. As
 25 will be described below, there are a number of various ways to specify this data structure 216, and what is drawn. Further, the high-level composition and animation engine 214 integrates with timing and animation systems 220 to provide declarative (or other) animation control (e.g., animation intervals) and timing control. Note that the animation system allows animate values to be passed essentially anywhere in the system, including, for example, at the element property level 208, inside of the visual
 30 API layer 212, and in any of the other resources. The timing system is exposed at the element and visual levels.

36

The low-level compositing and animation engine 218 manages the composing, animating and rendering of the scene, which is then provided to the graphics subsystem 222. The low-level engine 218 composes the renderings for the scenes of multiple threads (e.g., from one or more applications), and with rendering components, implements the actual rendering of graphics to the screen. Note, however, that at times it may be necessary and/or advantageous for some of the rendering to happen at higher levels. For example, while the lower layers service requests from multiple threads, the higher layers are instantiated on a per-thread basis, whereby is possible via the imaging mechanisms 204 to perform time-consuming or thread-specific rendering at higher levels, and pass references to a bitmap to the lower layers.

The MIL 200 provides a number of concepts which are integrated to provide improvements in graphics and audiovisual programming, such as a set of resources and classes that are shared through this entire layered stack, including Pens, Brushes, Geometry, Transforms and Effects. Also, simple primitive types are provided, including Points, Rectangles, and so forth. Pen and Brush are complex types that describe how to affect rendering at these various levels. Also provided is a special type of Brush, referred to as VisualBrush, that allows programmers to use an arbitrary graphics "metafile" to fill an area (via a VisualBrush explicitly or a Pen that references a VisualBrush). Since this is a compressed form for storing and using arbitrary graphics, it plays the role of a graphics resource. There is a specific profile of the vector graphics markup syntax used to create these objects directly. Although generally similar to VisualBrush, the DrawingBrush is more compressed and digested, and is essentially a metafile brush whereas the VisualBrush is a scene graph brush.

Other classes include Geometry, which is a complex type that is used to define an area for filling, stroking or clipping. Transform is another complex type hierarchy for defining how to transform coordinate spaces. Effects describe a system for putting arbitrary filter effects on a section of content, e.g., a blur. Note that this also includes an add-in extensibility model.

A Visual API is provided that uses the above types to get bits to the screen or other targets. This includes base level hookup to the rest of the system (via an hWnd or other mechanism) along with the screen partitioning data structure introduced.

above. Imaging allows the programmer to get images into, and out of the MIL-based system. Media provides the ability to use other forms of media, including audio and video. In general, the Visual API refers to the API set that operates below the element system and the layout system, and refers to programming with and to Visuals directly, instead of at a higher level. Note that a visual is a basic object in the caching data structure 216, which comprises a retained data structure for visual things on the screen, and also caches the Instruction Lists and device specific resources for performance reasons.

FIGS. 3 and 4 show example scene graphs 300 and 400, respectively, including a base object referred to as a Visual. In general, a Visual comprises an object that represents a virtual surface to the user and has a Visual representation on the display. As represented in FIG. 5, a base class Visual provides the base functionality for other Visual types, that is, the Visual class is an abstract base class from which Visual types derive.

As represented in FIG. 3, a top-level (or root) Visual 302 is connected to a Visual manager object 304, which also has a relationship (e.g., via a handle) with a window (HWND) 306 or similar unit in which graphic data is output for the program code. The VisualManager 304 manages the drawing of the top-level Visual (and any children of that Visual) to that window 306. FIG. 6 shows the VisualManager as one of a set of other objects 620 in the object model of the graphics system described herein.

To draw, the VisualManager 304 processes (e.g., traverses or transmits) the scene graph as scheduled by a dispatcher 308, and provides graphics instructions and other data to the low level component 218 (FIG. 2) for its corresponding window 306, such as generally described in the aforementioned U.S. Patent Applications. The scene graph processing will ordinarily be scheduled by the dispatcher 308 at a rate that is relatively slower than the refresh rate of the lower-level component 218 and/or graphics subsystem 222. FIG. 3 shows a number of child Visuals 310-314 arranged hierarchically below the top-level (root) Visual 302, some of which are represented as having been populated via drawing contexts 316, 317 (shown as dashed boxes to represent their temporary nature) with associated instruction lists 318 and 319, respectively, e.g., containing Instruction Lists and other Visuals. The Visuals may

also contain other property information. In general, most access on the base visual class comes via an IVisual interface, and visual derives from DependencyObject, as represented in FIG. 5. Visuals (further described in the Appendix) may also contain other property information, as shown in the following example Visual class:

1

200

```

public class Visual : IVisual, DependencyObject
{
    protected Visual();
    protected Visual(UIContext context);

    bool IVisual.HasChildren { get; }
    VisualCollection IVisual.Children { get; }

    Visual IVisual.Parent { get; }

    Geometry IVisual.Clip { get; set; }
    Bool IVisual.Show { get; set; }
    double IVisual.Opacity { get; set; }
    BlendMode IVisual.BlendMode {get; set;}

    Matrix IVisual.TransformToAncestor(Visual ancestor);
    Matrix IVisual.TransformToDescendant(Visual descendant);
    Matrix IVisual.TransformFromAncestor(Visual ancestor);
    Matrix IVisual.TransformFromDescendant(Visual descendant);
    Matrix IVisual.TransformToVisual(Visual visual);
    Matrix IVisual.TransformFromVisual(Visual visual);

    bool IVisual.IsAncestorOf(Visual descendant);
    bool IVisual.IsDescendantOf(Visual ancestor);

    Visual IVisual.FindCommonVisualAncestor(Visual otherVisual);

    PointHitTestResult IVisual.HitTest(Point point);
    void IVisual.HitTest(
        HitTestFilterDelegate filterHitDelegate,
        HitTestResultDelegate resultHitDelegate,
        HitTestParameters hitTestParams);

```

```

Rect IVisual.VisualContentBounds { get; }
Rect IVisual.VisualDescendantBounds { get; }

protected virtual PointHitTestResult HitTestCore(
    PointHitTestParameters point);
protected virtual GeometryHitTestResult HitTestCore(
    GeometryHitTestParameters geometry);

protected virtual Rect HitTestBounds { get; }
}

```

5 A Visual is a container for graphical content and a set of children. Various properties on a Visual can be used to control the rendering behavior of a Visual. For example by setting a clip on a Visual the content of a Visual is clipped to the shape specified. Other properties are transform, blend mode, opacity, show, etc. All those properties can be controlled via get and set properties.

10 A Show property is used to show/hide the visual, e.g., when false the visual is invisible, otherwise the visual is visible. Furthermore, MIL objects (whether Visuals at the Visual API layer or elements at the element layer) exist in a hierarchy. A coordinate system is inherited down through this hierarchy. In this way, a parent can push a coordinate transform that modifies the rendering pass and gets applied to that parent's children.

15 The transform for a visual is on the connection to that visual. In other words, it is set via the [Get/Set]ChildTransform on the parent's VisualCollection Children property. See also the VisualCollection described later on.

20 Note that the coordinate transforms may be applied in a uniform way to everything, as if it were in a bitmap. Note that this does not mean that transformations always apply to bitmaps, but that what gets rendered is affected by transforms equally. By way of example, if the user draws a circle with a round pen that is one inch wide and then applies a scale in the X direction of two to that circle, the pen will be two

inches wide at the left and right and only one inch wide at the top and bottom. This is sometimes referred to as a compositing or bitmap transform (as opposed to a skeleton or geometry scale that affects the geometry only). FIG. 8 is a representation of scaling transformation, with a non-transformed image 800 appearing on the left and a
 5 transformed image 802 with a non-uniform scale appearing on the right. FIG. 9 is a representation of scaling transformation, with the non-transformed image 800 appearing on the left and a transformed image 904 with geometry scaling appearing on the right.

With respect to coordinate transformation of a visual, TransformToDescendant
 10 returns a transform that reflects the coordinate space change going from the reference visual to the descendant visual. The transform can then be used to for example transform a point from the coordinate space of the reference visual to the coordinate space of the descendant visual. TransformFromDescendant returns similarly a
 15 transform that describes the coordinate space change going from the descendant Visual to the reference Visual. The transformation can then be used to transform a point from coordinate space of the descendant visual to coordinate space of the reference visual. For convenience the Visual API also provides the TransformToAncestor, TransformFromAncestor, TransformFromVisual, and TransformToVisual which also
 20 return transforms for the respective coordinate space changes. Note that in the latter two APIs, the relationship between the Visuals is not specified. They can even be peers in the Visual tree as long as they share a common ancestors. The implementation will find the common ancestor and then calculate the coordinate transformation from the reference Visual to the common ancestor, and then from the common ancestor to the target Visual. The resulting transform can be used to for example transform a point
 25 between the specified Visuals.

Two get properties are available that may be used to determine the bounding box of the content of the Visual, namely VisualDescendantBounds, which is the bounding box of all the graphical content of the descendants, and
 VisualContentBounds which is the bounds of the content. Applying a Union to these
 30 provides the total bounds of the Visual.

The clip property sets (and gets) the clipping region of a visual. Any Geometry (the geometry class is shown in FIG. 10 and described in the Geometry section below)

can be used as a clipping region. In one implementation, a default setting for the clipping region is null, i.e., no clipping, which can be thought of as an infinite big clipping rectangle from $(-\infty, -\infty)$ to $(+\infty, +\infty)$.

5 The Opacity property gets/sets the opacity value of the visual, such that the content of the visual is blended on the drawing surface based on the opacity value and the selected blending mode. The BlendMode property can be used to set (or get) the blending mode that is used. For example, an opacity (alpha) value may be set between 0.0 and 1.0, with linear alpha blending set as the mode, e.g., $\text{Color} = \alpha * \text{foreground color} + (1.0 - \alpha) * \text{background color}$. Other services, such as special
10 effects properties, may be included in a visual, e.g., blur, monochrome, and so on.

The Visual has also a Children property for managing the set of children. It also provides a HasChildren property to check if the Visual has any children at all. The Children property returns a VisualCollection that lets the user perform operations like add, remove, insert, etc. on the set of children. The following shows an example of a
15 VisualCollection:

209

```

public class Visual : ICollection, IEnumerable
{
    public Int32 Add(Visual visual)
    public Void Clear()
    public Boolean Contains(Visual visual)
    public virtual sealed Void CopyTo(Array array, Int32 index)
    public Int32 get_Capacity()
    public virtual sealed Int32 get_Count()
    public virtual sealed Boolean get_IsSynchronized()
    public Visual get_Item(Int32 index)
    public virtual sealed Object get_SyncRoot()
    public virtual sealed IEnumerator GetEnumerator()
    public Transform GetTransform(Int32 index)
    public Transform GetTransform(Visual child)
    public Int32 IndexOf(Visual visual)
    public Void Insert(Int32 index, Visual visual)
    public Void Remove(Visual visual)
    public Void RemoveAt(Int32 index)
    public Void RemoveRange(Int32 index, Int32 count)
    public Void set_Capacity(Int32 value)
    public Void set_Item(Int32 index, Visual value)
    public Void SetTransform(Int32 index, Transform transform)
    public Void SetTransform(Visual child, Transform transform)
    public Void TrimToSize()
}

```

The order of the Visuals in the VisualCollection determines in which order the Visuals are rendered, i.e. Visuals are rendered from the lowest index to the highest index from back to front (painting order).

- 5 A ProxyVisual is a visual that may be added more than once into the scene graph, e.g., below a container visual. Since any visual referred to by a ProxyVisual

may be reached by multiple paths from the root, read services (TransformToDescendent, TransformFromDescendent and HitTest) do not work through a ProxyVisual. In essence, there is one canonical path from any visual to the root of the visual tree and that path does not include any ProxyVisuals.

5 FIG. 4 shows an example scene graph 400 in which ContainerVisuals and DrawingVisuals are related in a scene graph, and have associated data in the form of Instruction Lists, (e.g., in corresponding drawing contexts). The ContainerVisual is a Visual that has only structural content that is derived from the Visual base class. Visuals can be arbitrarily nested into each other. In particular it is legal to nest
10 ContainerVisuals. The main purpose of the ContainerVisual is to provide a container for Visuals that can be conveniently accessed without going through the IVisual interface. The ContainerVisual therefore reimplements all the IVisual methods as public methods again. The children of a ContainerVisual can be manipulated can be manipulated with methods on the VisualCollection Children property of the
15 ContainerVisual.

Returning to FIG. 5, yet another visual is an HwndVisual 505, which positions a Win32 child Hwnd in the scene graph. More particularly, legacy programs will still operate via the WM_PAINT method (or the like) that draws to a child Hwnd (or the like) based on prior graphics technology. To support such programs in the new
20 graphics processing model, the HwndVisual allows the Hwnd to be contained in a scene graph and moved as the parent visual is repositioned. Other types of visuals 506 are also feasible, such as three-dimensional (3D) visuals which enable a connection between two-dimensional and three dimensional worlds, e.g., a camera-like view is possible via a two-dimensional visual having a view into a three-dimensional world.
25 Such a 3D visual is described below.

As described above, visuals can be drawn on by populating their drawing contexts with various drawing primitives, including Geometry, ImageSource and MediaData. Furthermore, there are a set of resources and classes that are shared through this entire stack. This includes Pens, Brushes, Geometry, Transforms and
30 Effects. The DrawingContext abstract class exposes a set of drawing and context state operations that can be used to populate a DrawingVisual, RetainedVisual, ImageData, etc. In other words, the drawing context abstract class exposes a set of drawing

operations and push/pop operations; for each drawing and push operation there are two methods, one that takes constants as arguments, and one that takes animators as arguments. Examples for push/pop operations are PushTransform, PopTransform, PushClip, PopClip, PushOpacity, PopOpacity, etc.

- 5 The various services (including transform, opacity, and clip) can be pushed and popped on a drawing context, and push/pop operations can be nested, as long as there is an appropriate pop call for each push call.

10 The PushTransform method pushes a transformation. Subsequent drawing operations are executed with respect to the pushed transformation. The pop call pops the transformation pushed by the matching PushTransform call:

```
void PushTransform(Transform transform);
void PushTransform(Matrix matrix);
void Pop();
```

15

Similarly, the PushOpacity method pushes an opacity value. Subsequent drawing operations are rendered on a temporary surface with the specified opacity value and then composite into the scene. Pop() pops the opacity pushed by the matching PushOpacity call:

20

```
void PushOpacity(float opacity);
void PushOpacity(FloatAnimation opacity);
void Pop();
```

25

The PushClip method pushes a clipping geometry. Subsequent drawing operations are clipped to the geometry. The clipping is applied in post transformation space. Pop() pops the clipping region pushed by the matching PushClip call:

```
void PushClip(Geometry clip);
void Pop();
```

30

Note that push operations can be arbitrarily nested as long as the pop operations are matched with a push. For example, the following is valid:

PushTransform(...);
 DrawLine(...);
 PushClip(...);
 DrawLine(...);
 Pop();
 PushTransform(...);
 DrawRect(...);
 Pop();
 Pop();

Geometry is a type of class (FIG. 10) that defines a vector graphics skeleton, without stroke or fill. Each geometry object is a simple shape (LineGeometry, EllipseGeometry, RectangleGeometry), a complex single shape (PathGeometry) or a list of such shapes GeometryCollection with a combine operation (e.g., union, intersection, and so forth) specified. These objects form a class hierarchy as represented in FIG. 10.

As represented in FIG. 11, the PathGeometry is a collection of Figure objects. In turn, each of the Figure objects is composed of one or more Segment objects which actually define the figure's shape. A Figure is a sub-section of a Geometry that defines a segment collection. This segment collection is a single connected series of two-dimensional Segment objects. The Figure can be either a closed shape with a defined area, or just a connected series of Segments that define a curve, but no enclosed area.

As represented in FIG. 12, when geometry (e.g., a rectangle) is drawn, a brush or pen can be specified, as described below. Furthermore, the pen object also has a brush object. A brush object defines how to graphically fill a plane, and there is a class hierarchy of brush objects. This is represented in FIG. 12 by the filled rectangle 1202 that results when the visual including the rectangle and brush instructions and parameters is processed. A Pen object holds onto a Brush along with properties for Thickness, LineJoin, LineCap, EndCap, MiterLimit, DashArray and DashOffset, as described below. As also described below, some types of Brushes (such as gradients

and nine grids) size themselves. When used, the size for these brushes is obtained from the bounding box, e.g., when the GradientUnits/DestinationUnits for the Brush is set to RelativeToBoundingBox, the bounding box of the primitive that is being drawn is used. If those properties are set to Absolute, then the coordinate space is used.

5 As mentioned above and as further described below, the graphics object model of the present invention includes a Brush object model, which is generally directed towards the concept of covering a plane with pixels. Examples of types of brushes are represented in the hierarchy of FIG. 13, and, under a Brush base class, include Gradient Brush, NineGridBrush, SolidColorBrush and TileBrush. GradientBrush
10 includes LinearGradient and RadialGradient objects. DrawingBrush and ImageBrush derive from TileBrush. Alternative arrangements of the classes are feasible, e.g., deriving from TileBrush may be ImageBrush, VisualBrush, VideoBrush, NineGridBrush and Drawing Brush. Note that Brush objects may recognize how they relate to the coordinate system when they are used, and/or how they relate to the
15 bounding box of the shape on which they are used. In general, information such as size may be inferred from the object on which the brush is drawn. More particularly, many of the brush types use a coordinate system for specifying some of their parameters. This coordinate system can either be defined as relative to the simple bounding box of the shape to which the brush is applied, or it can be relative to the
20 coordinate space that is active at the time that the brush is used. These are known, respectively, as RelativeToBoundingBox mode and Absolute mode.

VISUAL API

The Visual API is a starting point for drawing via the media integration layer,
25 and comprises multiple types of objects, including a VisualManager object, which connects a Visual Tree to a medium. The different types of VisualManagers (e.g., Screen, Printer, Surface) manage the rendering process of a Visual Tree to their particular medium. The VisualManager is further described herein in the section entitled "Top Level MIL Objects."

30 A visual is where the user does drawing. It is a node in the visual tree (a container object that is the structure for a scene, as described below) and provides a

209

place for a program to draw. There are various types of visuals, each tailored for different usages. A visual is similar to the visual/output side of a Win32 hWnd.

Visuals provide a number of capabilities, including parent visual access, child visual collection, clipping based on an arbitrary geometry, opacity, blendmode, transform that affects that visual and its children, hit testing, coordinate transformation services, bounding box services, and effects (raster and vector).

To render a visual scene, the visual tree is traversed, e.g., top-down, left-right, first rendering the content and then traversing the Visual's children from left to right. Any of a visual's children are drawn before the content of that visual itself. If the content requires a callback to the user, it happens synchronously during render time. Here is some pseudo-code for what the system does (via a VisualRenderer) to render a Visual tree to a device:

```
Visual.RenderToDevice(device)
{
15   this.RenderContents(device);
   foreach (Visual child in this.Children)
   {
       Child.Walk(ctx);
   }
20 }
```

A derived visual may call back to the user during the RenderContents call.

FIG. 13 represents a Visual Class Hierarchy in one implementation. During callbacks that happen as part of the render pass, (comprising a callback to IRetainedVisual.Render or a callback to PaintingVisual.RenderCore) the Visual tree is "locked" for performance reasons. This locking happens on a context-wide basis, meaning that no context can modify the tree when locked, no matter which visual tree a visual belongs to. When the tree is locked, the children of a Visual cannot be changed, and another visual's contents may not be changed in any way (e.g., Open, set root 3D model, and so forth), Transform, Opacity, Clip, BlendMode or Effect on a Visual cannot be set, hit testing will not operate, nor will a Get bounding information operation work.

The capabilities on a *Visual* are exposed via the *IVisual* interface, thereby making the capabilities public, while protecting the object model. The following is the *IVisual* interface in one example implementation:

```

5  public interface IVisual
   {
       bool HasChildren { get; }
       VisualCollection Children { get; }

       Visual Parent { get; }
10  Geometry Clip { get; set; }
       bool Show { get; set; }
       double Opacity { get; set; }
       Matrix TransformToAncestor(Visual ancestor);
       Matrix TransformToDescendant(Visual descendant);
15  Matrix TransformFromAncestor(Visual ancestor);
       Matrix TransformFromDescendant(Visual descendant);
       Matrix TransformToVisual(Visual visual);
       Matrix TransformFromVisual(Visual visual);
       bool IsAncestorOf(Visual descendant);
20  bool IsDescendantOf(Visual ancestor);
       Visual FindCommonVisualAncestor(Visual otherVisual);
       PointHitTestResult HitTest(Point point);
       void HitTest(
           HitTestFilterDelegate filterHitDelegate,
25  HitTestResultDelegate resultHitDelegate,
           HitTestParameters hitTestParams);
       Rect VisualContentBounds { get; }
       Rect VisualDescendantBounds { get; }
   }
30

```

A *Visual* comprises render content and a collection of children. A number of properties, including transform, clip, opacity, blend mode, and so on can be used to

control the actual rendering of the Visual tree. Note that a Visual does not have to have content and children at the same time. In an implementation, rendering content and child collection could be created on demand to optimize for memory usage. The Visual API allows the user of it to derive from a Visual and specialize it.

- 5 Transformations on children are performed via a transform on the property Children of type VisualCollection.

```
public void SetChildTransform(Visual visual, Transform transform);  
public Transform GetChildTransform(Visual visual);
```

- 10 Transformation Read Services provide methods that allow users to obtain a Matrix representing the aggregate transform from one coordinate frame to another:

```
Matrix TransformToDescendant(Visual descendant);  
Matrix TransformFromDescendant(Visual descendant);  
Matrix TransformToAncestor(Visual ancestor);  
15    Matrix TransformFromAncestor(Visual ancestor);  
Matrix TransformToVisual(Visual visual);  
Matrix TransformFromVisual(Visual visual);
```

- 20 The TransformToAncestor and TransformToDescendant methods are more efficient but require the caller to know the relationship between two Visuals. The more general TransformTo/FromVisual methods find a common ancestor and compute the transform to that visual. Note that these may cause caches to be updated and OnRender to be called on arbitrary visuals. If the visuals are not connected or a degenerate transform is encountered, an exception is thrown.

- 25 Bounds calculation is also provided:

```
protected Rect VisualDescendantBounds { get; }  
protected Rect VisualContentBounds { get; }
```

- 30 VisualDescendantBounds returns the union of the content bounding boxes for the descendants of the current visual, but not including the contents of the current visual. VisualContentBounds returns the bounding box for the contents of the current visual.

212.

A **Opacity** property (e.g., a double) specifies an optional **Opacity** value to apply to a **Visual** when it is composited into its parent. By default, this value is 1.0, which will cause the contents to appear with their full **Opacity**. Note that since this value is multiplied with any other **Opacity** data in the sub-graph, a **Opacity** of 1.0 alters
5 nothing. A **Opacity** of 0.0 causes the entire contents to be transparent, a value of 0.25 causes **Opacity** to be twenty-five percent of its nominal values, and so forth. **Opacity** is applied before the **BlendMode**.

The **BlendMode** is a property that specifies an optional **BlendMode** to apply to the contents of a sub-graph and the destination when this **Visual** is composited. By
10 default, this value is **BlendMode.Normal**, which performs an Alpha-channel aware composition into the destination. Setting this property to some other value will perform the composition with the **Visual**'s contents as the source and the contents of the render target as the destination. This is applied after the **Opacity** property is applied.

15 The base visual class presents interfaces for the features that **Visuals** have in common:

```
public class System.Windows.Media.Visuals.Visual :
    DependencyObject, IVisual
{
20    // Constructors
    protected Visual();
    protected Visual(UiContext Context);

    // Finalizer
25    ~Visual();

    protected virtual PointHitTestResult HitTestCore(
        PointHitTestParameters point);
    protected virtual GeometryHitTestResult HitTestCore(
30    GeometryHitTestParameters geometry);

    protected virtual Rect HitTestBounds { get; }
```

```

// All methods for IVisual are implemented. These are implemented
// explicitly only for the interface. Here is an example:
bool IVisual.HasChildren { get; }
}

```

5

The ContainerVisual derives directly from the visual class and promotes the protected properties to be public. This is to enable a user to create visual containership without having to derive a new class.

```

10 public class System.Windows.Media.Visuals.ContainerVisual :
    Visual
    {
        // Constructors
        public Visual();
15     public Visual(UiContext Context);

        // All IVisual methods are "reexposed" so that
        // they are public. This class is merely a convenience
        // above and beyond Visual
20     }

```

A RetainedVisual is a visual that introduces a "retained instruction stream" that can be used for drawing:

```

25 public class System.Windows.Media.Visuals.RetainedVisual : Visual
    {
        protected RetainedVisual();
        protected RetainedVisual(UiContext Context);

        // Open/Close semantics
30     protected DrawingContext RenderOpen();
        protected DrawingContext RenderAppend();

```

```

// Invalidation
protected bool IsVisualInvalid { get; }
protected bool InvalidateVisual();
Rect RenderBounds { set; get; }
5 }

public class System.Windows.Media.Visuals.IRetainedRender
{
    void Render(DrawingContext drawingContext);
10 }

```

The instruction stream can be used in an OnDemand mode, in which the user is called back to render as needed. The user is required to implement IRetainedRender. The instruction stream can be used in an Imperative mode, in which the user can directly call RenderOpen and get a drawing context. Generally a user will use one of these modes at a time, but it is possible for them to be used in a mixed way.

RenderOpen and RenderAppend will affect the current stream and are available in various scenarios. They will throw if this Visual is currently in a Render callback. RenderOpen clears all previous content that was in the RetainedVisual, while RenderAppend will tack new content on to the end of the stream. If the user has implemented IRetainedRender on the visual, then the user signals the system that the OnDemand mode also should be used. The system will use the value set in to the RenderBounds property as the bounds for the content to be provided by the Render call. The system may decide to optimize the scene and throw content away at any time, when IRetainedRender is implemented. RenderBounds will default to an empty rect, although Rect.Infinite or an unset value are feasible alternatives. To get the virtualization performance gains implied with the callback, the user will have to set in a reasonable value.

When rendering the scene, the system will examine each visual in concept (note that in reality the system will be able to ignore most visuals most of the time). If that visual has IsVisualInvalid set to true, and, based on RenderBounds, that visual's contents will be needed, then the system will call IRetainedVisual.Render to fill in the

215

contents of that visual. This will replace any content that is already there. The user can manually tell the system to throw the content stream away by calling Invalidate.

If IRetainedRender is not implemented then IsVisualInvalid will always return false. Invalidate will do nothing. Note that IRetainedRender is so named (e.g., instead of IRender), because it is not general enough to be used in all rendering callback cases. For instance, the PaintingVisual calls back with an invalid rectangle.

A DrawingVisual is very similar to RetainedVisual but is designed to be used without derivation. The protected methods are "promoted" to public. Furthermore, there is no Render callback or need to implement the IRetainedRender interface.

10 Because of this, the content is always retained, similar to when IRetainedRender is not implemented on a RetainedVisual.

```
public class System.Windows.Media.Visuals.DrawingVisual :
    ContainerVisual
{
    15 // Constructors
    public DrawingVisual();
    public DrawingVisual(UiContext Context);

    // Tree interfaces
    20 public bool HasVisualChildren { get; }
    public Visual VisualParent { get; }
    public bool IsVisualAncestorOf(Visual descendant);
    public bool IsVisualDescendantOf(Visual ancestor);
    public Visual FindCommonVisualAncestor(Visual otherVisual);

    25 // Standard compositing properties
    public new Geometry VisualClip { get; set; }

    // Default value is 1.0
    30 public new double VisualOpacity { get; set; }
    public new DoubleAnimationCollection VisualOpacityAnimations { get; set; }
```

```

// Default value is BlendModes.Normal
public new BlendMode VisualBlendMode { get; set; }

// Transformations
5 public Transform TransformToDescendant(Visual descendant);
  public Transform TransformFromDescendant(Visual descendant);
  public Transform TransformToAncestor(Visual ancestor);
  public Transform TransformFromAncestor(Visual ancestor);
  public Transform TransformToVisual(Visual visual);
10 public Transform TransformFromVisual(Visual visual);

// Bounds calculation
  public new Rect VisualDescendantBounds { get; }
  public new Rect VisualContentBounds { get; }
15

// Open/Close operations
  public new DrawingContext RenderOpen();
  public new DrawingContext RenderAppend();
  }
20

```

A **PaintingVisual** is also provided:

```

public class System.Windows.Media.Visuals.PaintingVisual : Visual
{
  protected SurfaceVisual();
25 protected SurfaceVisual(UiContext Context);

// Surface properties
  protected Rect PaintingBounds { get; set; }
  protected Color FillColor { get; set; } // defaults to transparent – more efficient if
30     opaque

// Open/Close semantics

```

```

protected StaticDrawingContext PaintingOpen();
protected StaticDrawingContext PaintingAppend();

// RenderCore semantics
5 protected abstract void RenderCore(Rect invalidRect);
protected void InvalidateRender();
protected void InvalidateRender(Rect invalidationRect);
}

```

10 While the RetainedVisual adds a retained instruction stream, the PaintingVisual is essentially backed by a surface. The system may virtualize the surface and still keep rendering instructions as long as the performance requirements are met. Because of this, the surface is not accessible by the user.

One difference of a PaintingVisual relative to a RetainedVisual is that the methods provide a StaticDrawingContext which allows no animation. If an animated argument is used anywhere an exception will be thrown. Another difference is that "Append" will grow the instruction stream much cheaper in terms of memory cost. Further, PaintingBounds, which essentially sets a hard clip on the user, is required and is necessary. Note that this is different from the clip property, as this bounds the content of this visual whereas Clip will clip the content of this visual and all of its children. RenderCore (analogous to IRetainedVisual.Render) is also implemented, wherein if the resolution changes or if the system needs to re-render the content for some reason, the user provides this mechanism.

25 A PaintingVisual is potentially much more lightweight than a SurfaceVisual, as there is no explicit backing of this visual by a surface. Instead, this is a node in a display tree that is at some lower point backed by a surface.

To achieve a SurfaceVisual, the user should create a RetainedVisual, call DrawImage and then change the image behind the scenes. In that case the user is explicitly controlling the rasterization instead of letting the system call back. There will be an immediate mode API for working on the image directly. This API will allow the user to get a StaticDrawingContext that works on that image. Note that the API for a SurfaceVisual is similar to an hWnd, DUser Gadget or a Trident display tree

node. "Appending" content (actually -- making a small delta that composites over what is already there) is a cheap operation. There is, by and large, no memory penalty for appending content, unlike with a RetainedVisual whereby RenderAppend will cause the instruction stream to get longer and longer, potentially leading to exponential growth. Since the surface that may be backing this visual may come and go, the visual is required to implement an "on-demand" RenderCore virtual.

A main usage scenario for PaintingVisual is for porting of application code that is largely structured around a WM_PAINT painting model. This is also useful for static content that is rarely changing and dense. Note that the PaintingVisual may be backed by a metafile or by a true surface. The system can decide at runtime which is more appropriate, e.g., based on memory and performance concerns. However, it is guaranteed that, past a certain point, appending new content will not cost more memory. Note that the system can switch between metafile and surface storage as needed.

15

TOP LEVEL MIL OBJECTS

As can be readily appreciated, various objects are provided for operating in a typical windowed scenario. Note that these are not necessarily formal classes (e.g., there is not an explicit scheduler interface or object).

20

One such object comprises a visual tree, which is an object that contains the main content to be drawn. Controls will derive from visuals of the tree directly. Visuals are device and context independent.

A render target is the device to which the visual is drawn. This object (e.g., screen) may have its own dirty or invalidation mechanism, which is necessary to back the visual system by a legacy hWnd. Various render targets include a screen in a window, a Printer, a Metafile, a Surface, and a "Sub-window" which is a part of the scene that is drawn separately from the rest of the scene. This is the mechanism to enable cross thread drawing, and is equivalent to a lower-level engine's compositable objects.

30

Other drawing related objects include a Visual Renderer, comprising an object that is configured to draw a visual tree onto a render target, and a Display Scheduler object that knows when to draw the visual tree on to the render target. A Time

56

29

Manager is a context object for a set of timing nodes, and is the object that the scheduler calls tick on.

The following is an example flow of control for drawing to the screen:

- 5 1. The user acquires the UiContext in some way and starts modifying the Visual Tree. This may be during application start up or perhaps in response to a UI input event.
2. The dirty notification propagates up the Visual tree. The root visual knows which visual renderer it is associated with and forwards the dirty notification. This notification is private.
- 10 3. The visual renderer, via a public event, reports that it has been changed and is out of sync with its render target.
4. The scheduler decides when and where to actually rectify this situation and cause drawing to happen. Typically this is done by posting a work item to the dispatcher. However, the user can do something else.
- 15 5. The user yields the UiContext and allows the Dispatcher to run.
6. The Dispatcher runs and calls the scheduler deferred work item. (Most probably any deferred work items in a single context will be coalesced. Also, this may run lockstep with the layout system to reduce thrashing.)
7. The scheduler runs its main update loop:
 - 20 a. Ticks the TimeManager
 - b. Runs layout, among other things
 - c. Tells the visual renderer to render new changes to the render target. The renderer then:
 - 25 i. Walks the dirty parts Visual Tree and updates internal cached bounding boxes.
 - ii. Calls all necessary "on demand" visuals to render. (By default "on demand" visuals will have an empty rect as their bounds and so they will not be called until layout runs and sets them up.)
 - iii. Walks the dirty parts again and sends the necessary rendering updates
30 to the lower level graphics systems.

59

Note that the visual system knows nothing about the dispatcher. It is up to the scheduler object to take care of those details. The scheduler can do any appropriate control flow.

Furthermore, there is the idea of an incremental visual renderer and a snap-shot visual renderer. It may be desirable for a visual to belong to one and only one incremental visual renderer at a time. This restriction is necessary for efficient caching of data on the visual itself. However, it is reasonable to also have a way to "snap-shot" an entire visual tree to a render target. In this case there are no persistent connections between the visual tree and the renderer. This might be use for getting a high resolution screen grab or for sending the visual tree (as it is on the screen) directly to a printer.

The Window plays the role of the render target, above. It is also a managed replacement for hWnd.

```

class System.Windows.Media.WindowContext
15  : IDisposable
{
    public WindowContext();

    // IDispose
20  public void Dispose();
    public bool IsDisposed { get; set; }

    // Window size
    public Size Size { get; }
25  public event EventHandler SizeChanged { add; remove; }

    // Window validation
    public bool IsInvalid { get; }
    public event EventHandler IsInvalidChanged { add; remove; }
30

    // Resolution information
    public ResolutionInformation ResolutionInformation { get; }

```

60

```

public event EventHandler ResolutionInformationChanged { add; remove; }
}

```

Window manager control is outside of this object, however it can be integrated with the WindowContext by making properties (e.g., such as Size) read/write values, and providing additional properties such as location, window title, and so forth. Note that Size represents the size of the Window in physical (1/96th of an inch) units. This is not a pixel size. Note that there may be situations where the data rendered to the window is lost for some reason, such as a video mode switch or a switch from a local console to a remote terminal server session.

VisualRenderers and VisualManagers are other objects, and responsible for rendering a tree of Visuals to a render target. A VisualRenderer provides a simple "one shot" model that will render to a medium while the VisualManager establishes a retained connection between the tree of visuals and the target they are being rendered to. It supports an "incremental" render to the medium.

The following is an example of what the base VisualRenderer looks like in one implementation:

```

public class System.Windows.Media.Visuals.VisualRenderer :
    System.Threading.ContextAffinityObject,
    IDisposable
{
    // Constructors
    internal VisualRenderer();
    internal VisualRenderer(System.Threading.UiContext context);

    // ContextAffinityObject
    public UiContext Context { get; }
    public bool CheckContext();
    public void VerifyContext();
    public void SetContext(System.Threading.UiContext context);

    // IDisposable+
}

```

```

public void Dispose();
public bool IsDisposed { get; }

// Properties
5 public Color BackgroundColor { get; set; }
  public Visual RootVisual { get; set; }

// Device information
public Size RootSize { get; }
10 public ResolutionInformation ResolutionInformation { get; set; }

// Render control
public void RenderNow();
}

```

15

The class cannot be publicly instantiated because as there is no "default" medium. The `VisualRenderer` is also a `ContextAffinity` object.

A `BackgroundColor` property is provided:

```

20 public Color BackgroundColor { get; set; }

```

20

This is the default background color of the visual manager, which may default this to transparent for `VisualManagers`. However, some mediums (such as rendering to legacy HWnds) cannot support per pixel transparency, and thus each `VisualManager` can define its own default for this property. Most applications would ignore this property, and for example may be set to the system window background color or transparent.

25

The `RootVisual` property identifies the root visual for rendering:

```

public Visual RootVisual { get; set; }

```

30

This defaults to null. When the `RootVisual` property is null the `VisualManager` draws the background color onto the medium.

The RootSize property returns, in virtual units, the size of the render target. For example, for a VisualManager backed by a window, this will be the client size of the window:

```
public abstract Size RootSize { get; }
```

5 Resolution Information is also provided:

```
public ResolutionInformation ResolutionInformation { get; set; }
```

10 Every medium is required to have a device resolution, even if it is not backed by pixels. For instance, when printing, even if capturing to a metafile, there needs to be resolution information made available through the VisualManager so that content can be optimized for that resolution. Note that in the case of a metafile capture, a relatively high default resolution may be used, while allowing the user to configure the resolution directly.

15 The initial "world to device" transform set up for the RootVisual makes it so that one unit in that visual is equal to 1/96th of an inch on the device. For instance, if there is a ScreenVisualManager that is backed by a device that is 192 dpi, then the initial transform must be set up so that one unit in the coordinate frame for the RootVisual must be equal to two units on the device. In this case,
20 ResolutionInformation.PixelSize will return (0.5, 0.5) to signify that each pixel is 1/48th of an inch on a side.

A VisualManager establishes a long term connection to the root visual and the render target and tracks differences:

```
25 public System.Windows.Media.Visuals.VisualManager :
    System.Windows.Media.Visuals.VisualRenderer
    {
        // Constructors
        internal VisualManager();
        30 internal VisualManager(System.Threading.UiContext context);
```

63

```

// Visual tree dirty
public bool IsVisualTreeDirty { get; }
public event EventHandler VisualTreeDirtyChanged { add; remove; }
}

```

5

The WindowVisualManager is a primary way to draw on the screen. It manages rendering a Visual Tree to a WindowContext:

```

public sealed class System.Windows.Media.Visuals.ScreenVisualManager :
    VisualManager
{
    // Constructors
    public ScreenVisualManager(WindowContext windowContext);
    public ScreenVisualManager(WindowContext windowContext,
        System.Threading.UiContext context);

    // WindowContext
    public WindowContext WindowContext { get; set; }
}

```

10

15

20 DRAWING CONTEXT

The DrawingContext APIs present a familiar, "context-based" programming model to those skilled in the art, for how to construct visual content that populates a Visual or are rendered to an ImageData. This section describes the DrawingContext classes, as well as the classes and entrypoints necessary to acquire a DrawingContext and enumerate the visual content in a RetainedVisual/DrawingVisual.

25

Applications do not directly construct a DrawingContext, and the exposed versions of DrawingContext are abstract classes. There are a number of ways to acquire a DrawingContext to put visual content into. These include RetainedVisual.RenderOpen() or RetainedVisual.RenderAppend(), which each return a DrawingContext to issue instructions into. Other ways include

30

IRetainedRender.Render(), DrawingVisual.RenderOpen(),

DrawingVisual.RenderAppend(), and PaintingVisual.PaintingOpen() or PaintingVisual.PaintingAppend() (although PaintingVisuals do not process animations). ImageData (or a subclass of ImageData) has a mechanism to return a DrawingContext for rendering onto a fixed-resolution bitmap surface. ImageData also
5 does not process animations.

The following sets forth the DrawingContext API:

```
public abstract class System.Windows.Media.StaticDrawingContext :
    ContextAffinityObject, IDisposable
{
10 // No public constructors

    // Vector graphics drawing methods
    public abstract void DrawLine(Pen pen, Point point0, Point point1);
    public abstract void DrawLine(Pen pen,
15     Point point0, PointAnimationCollection point0Animations
        Point point1, PointAnimationCollection point1Animations);

    public abstract void DrawRectangle(Brush brush, Pen pen, Rect rectangle);
    public abstract void DrawRectangle(Brush brush, Pen pen,
20     Rect rectangle, RectAnimationCollection rectangleAnimations);

    public abstract void DrawRoundedRectangle(Brush brush, Pen pen, Rect rectangle,
        double radiusX, double radiusY);
    public abstract void DrawRoundedRectangle(Brush brush, Pen pen,
25     Rect rectangle, RectAnimationCollection rectangleAnimations,
        double radiusX, double radiusY);

    public abstract void DrawEllipse(Brush brush, Pen pen,
        Point center, double radiusX, double radiusY);
30 public abstract void DrawEllipse(Brush brush, Pen pen,
        Point center, PointAnimationCollection centerAnimations,
        double radiusX, DoubleAnimationCollection radiusXAnimations,
```

65

```

double radiusY, DoubleAnimationCollection radiusYAnimations);

public abstract void DrawGeometry(Brush brush, Pen pen, Geometry geometry);

5 public abstract void DrawDrawing(Drawing drawing, Point location);

public abstract void DrawDrawing(Drawing drawing,
    Point location,
    PointAnimationCollection locationAnimation);

10 // Image and Video Drawing
public abstract void DrawImage(ImageData image, Rect rectangle);
public abstract void DrawImage(ImageData image,
    Rect rectangle, RectAnimationCollection rectangleAnimations);

15 public abstract void DrawVideo(MediaData video, Rect rectangle);
public abstract void DrawVideo(MediaData video,
    Rect rectangle, RectAnimationCollection rectangleAnimations);

20 // Text Drawing
public abstract void DrawText(FormattedText text, Point origin);
public abstract void DrawText(FormattedText text,
    Point origin, PointAnimationCollection originAnimations);

25 public abstract void DrawGlyphs(GlyphRun glyphRun);

// Context state methods
30 public abstract void PushClip(Geometry clipGeometry);
public abstract void PushOpacity(double opacity);
public abstract void PushOpacity(double opacity,

```

26

```

    DoubleAnimationCollection doubleAnimations);
    public abstract void PushTransform(Transform transform);
    public abstract void PushBlendMode(BlendMode blendMode);
    public abstract void Pop(); // applies to last Pushed state.

    // Quality metrics

    // Enumeration hints
    public abstract bool PreserveReadbackOrder { get; set; } // default = false

    // Close() can be called without all Push's having been Pop'd.
    public abstract void Close(); // same as IDisposable.Dispose();
}

```

Most of the methods of the DrawingContext object are self-explanatory to those skilled in the art, however it should be noted that DrawingContext is a ContextAffinityObject and is to be used only from a single UIContext. DrawingContext objects are also IDisposableables, and the recommended pattern, in C#, is to use them in a "using" clause if received from a RenderOpen/Append, for instance. Further, note that methods such as DrawArc, DrawPie, DrawBezier and DrawPolyline are not here. These require construction of the relevant Geometry, and use of DrawGeometry (as described below).

Further, while there are multiple Push* methods, there is only a single Pop method. This implies that there cannot be overlapped attributes. The attributes established by Push*() compose appropriately. For instance, Clip composes via the Intersection operator, Opacity via the Multiplication operation, and Transform via the ComposeTransform operation.

Since a user can call, for instance, DrawGeometry, with a null Brush, or with a null Pen, it is valid to call it with nulls for both Brush and Pen, however there will not be anything rendered or hit test against.

Any animate properties provided to the drawing context when used in a non-animate target (for instance, when rendering directly to a raster) will snap to time zero (although non-animate targets may alternatively snap to time "now"). This allows code that was written for a dynamic DrawingContext to more easily transition over to using a DrawingContext that does not process animation.

Enumeration of the content inside of a "retained" Visual is closely related to the DrawingContext by which that content was inserted. Enumeration of the contents of a Visual, and/or modification of the instruction stream may be performed via the Changeable mechanism, if desired, as described below. The general idea is to provide a "push-model" enumeration mechanism.

The DrawingContext class itself provides an interface to register for push-mode walking of content, a subclass of DrawingContext called DrawingContextWalker, (with most of the methods remaining abstract). Users subclass DrawingContextWalker, and then pass an instance into a Visual to begin enumeration. The Visual calls back the appropriate DrawingContext method to communicate the content that it has. DrawingContextWalker will also provide a piece of state that can be manipulated on it to control how the enumeration progresses (i.e., whether it should be stopped immediately, for instance).

The following is example code of for a DrawingContextWalker object:

239

```

public class MyContextWalker : DrawingContextWalker
{
    public void DrawLine(Pen pen, Point point0, Point point1)
    {
5       Console.WriteLine("Line from {0} to {1} with Pen {2}", point0, point1, pen);
    }
    public void DrawRectangle(Brush brush, Pen pen,
                               Rect rectangle, RectAnimationCollection rectangleAnimations)
    {
10      Console.WriteLine("AnimateRectangle with ...", ...);
    }
    // etc, etc...
}

15 // Get the contents of any RetainedVisual (including Controls, DrawingVisuals, etc)
MyContextWalker ctxWalker = new MyContextWalker ();
ctxWalker.WalkContent(myFancyControl, 0); // results in invocations on ctxWalker

```

The WalkContent method is provided in the DrawingContextWalker base
 20 class:

```

public abstract class DrawingContextWalker : DrawingContext
{
    protected DrawingContextWalker(); // this lets public subclasses start happening

25    public void WalkContent (RetainedVisual visual,
                               DrawingContextWalkOptions options);
    protected void StopWalking();

    // implementation of abstract methods from DrawingContext

30    // This doesn't make sense on the context walker, and will throw an exception
    // if accessed.

```

67

```

public bool PreserveReadbackOrder { get; set; }

// DrawText() expands into GlyphRun's, so it will never be called back,
// therefore we don't want the user to have to implement it.
5 public void DrawText(... regular ...); // should assert if ever called
public void DrawText(... animate ...); // should assert if ever called

// other methods remain abstract, and the subclasser implement.
}

```

10

Users subclass this and implement all abstract methods as desired. Users then create an instance of the object and call WalkContent on it. WalkContent then calls back the appropriate methods as it walks the Visual. The implementation of any of these methods can stop the walk, if desired, by calling the protected method

15 StopWalking(). Note that it is illegal to begin a walk when a DrawingContext on the Visual is open for rendering into.

The options determine how the walk proceeds:

```

[Flags]
public enum DrawingContextWalkOptions
20 {
    IncludeAnimations = 0x1
}

```

If IncludeAnimations is set, the walker invokes the appropriate methods with

25 animate content. Otherwise, the instantaneous value of the content is provided to the DrawingContextWalker methods.

The DrawingContext has a PreserveReadbackOrder Boolean that may provide the order or structure of the content to the Visual in the order and structure returned via the DrawingContextWalker class. This defaults to false, but may be set to true

30 before content is inserted when it is important to preserve the order. For example, as described above, DrawGeometry may be provided with a null Brush and a null Pen. If PreserveReadbackOrder is true, this instruction needs to be maintained in the Visual's

231
state. If PreserveReadbackOrder is false, the implementation is free to discard this instruction.

Note that there are numerous advantages of this push approach over other type-safe approaches, including that there is no need for a parallel set of types to reflect the
5 output of the enumeration, and that there is not necessarily a requirement for heap allocation in the callback interface. Further, the methods can be called directly without creating objects to pass back, the DrawingContext interface is already present, and there is no additional API needed on the Visual itself to allow for walking.

Modifying visual content is another consideration. One way of expressing
10 modifications to VisualContent is to use resources (Pens, Brushes, etc), which as described herein are subclasses of Changeable, with StatusOfNextUse == UseStatus.ChangeableReference. This will allow references to be maintained by the application to data that is in the managed structures sent into the DrawingContext. This represents a uniform way of allowing changes to be made, and, because these
15 objects are in a known state that they have been explicitly set into, the implementation knows which objects are likely to be modified. Note that it does not allow, for example, changes in ordering of instructions, or addition or deletion of instructions (although RenderAppend() is there for additions).

20 DRAWING

The Drawing class contains a collection of drawing commands. It is exactly equivalent to the content stored on a DrawingVisual, and is built by a DrawingContext. The Drawing does not have context affinity when immutable, and thus it and the related class DrawingBrush can be used across contexts and in default
25 property sheets when it itself is immutable.

It does not directly support hierarchy in that it provides no means of iterating children or finding a parent, but, via a DrawingContext, a Drawing can be drawn into another Drawing. A Drawing can be drawn into a DrawingContext via DrawDrawing(Drawing), and it can be used as the content description for a
30 DrawingBrush.

The Drawing is fully animatable, and supports readback/iteration in the same manner as the RetainedVisual.

```

public class System.Windows.Media.DrawingCollection: Changeable
{
    // Constructors
    public Drawing Collection();

    public new DrawingCollection Copy();

    // Properties
    public Rect Bounds { get; }

    // Open/Close operations
    public DrawingContext Open();
    public DrawingContext Append();
}

```

THE CHANGEABLE PATTERN

For purposes of explanation, the present invention will primarily be described in the context of a programming environment in which example objects in a graphics scene are constructed, used and modified. However, as will be understood, although the present invention provides significant benefits in graphics-related programming environments, the present invention is not limited to graphics-related programming environments, but more generally applies to many other types of programming environments.

In one implementation, the present invention provides a single set of types that derive from a common base class, e.g., System.Windows.Changeable. Any class can be mutable, by deriving from the Changeable class and thereby obtaining the value-type semantics that the Changeable offers. For example, in a graphics programming, the object model includes Brushes, Pens, Geometries, FloatAnimations, GradientStops, Segments, and so forth, as generally described in the aforementioned United States Patent Application Serial No. 10/402,268. For example, the hierarchy for a drawing brush may be something like:

Object:Changeable:Animatable:Brush:TileBrush:DrawingBrush.

For purposes of basic usage, a changeable object includes the following property and methods:

```
public class System.Windows.Changeable
{
    public bool IsChangeable { get; } // defaults to true
    public Changeable Copy();
    public void MakeUnchangeable();
}
```

5

The IsChangeable property specifies whether the changeable object can be modified or not, depending on its current value. For example, an attempt to set the opacity property of a brush will only succeed if that brush object has the IsChangeable property equal to true. Otherwise, an exception will be raised. When constructed, changeable objects have the IsChangeable property equal to true by default, and are thus immediately modifiable.

As represented in FIG. 14, requests 1402 are received, such as via function calls originated from an application program, that are directed to changeable classes 1404. In general, a request handler 1406, including a state machine 1408, processes the requests and maintains state and object data via a supporting data structure 1410, cloning the data structure in a cloned copy 1412 with an appropriate property state, based on a current property state, as described below. Exceptions 1414 may be thrown, such as when the request seeks a transition that is not allowed from within a current property state. The property states are described below with reference to FIGS. 15-17.

It should be noted that the function calls directed to changeable classes may be handled directly or indirectly. For example, the request handler 1406 in FIG. 14 may include an API set that provides the interfaces to the state machine. Alternatively, the request handler 1406 may comprise middleware code that converts the requests received in one operating system to API calls handled by another operating system. Thus, as used herein, the requests, via the request handler, "cause" the requested

behavior to occur, regardless of where the actual state machine processing occurs or where the data structures and classes are provided.

In this manner, (among other mechanisms), applications may construct changeable objects via a "new" request, set values into them, use them, continue
5 setting values into them, and continue using them.

The following is an example of how an application creates a brush of a solid color (scb), modifies the brush to have a solid color (red) and uses the brush to color a button's background red:

```
SolidColorBrush scb = new SolidColorBrush();
scb.Color = Colors.Red;
Button1.Background = scb; // this qualifies as a "use" (as described below)
```

10 The notion of "using" a value has a specific meaning, that is, values are only considered to be used under certain conditions. These conditions include when a value is set into a Property System property, when a value is used as a sub-object in a more complex changeable, and when a value is used in a DrawingContext command or the like. Note that system extenders can easily define other instances of using a
15 changeable object that qualifies as a "use" and modifies the changeable state of the object.

When a value is used in one of these qualified kinds of uses, from the user-model perspective, a clone of it is made, and that clone has its IsChangeable property set to false. Note that in actuality, a clone is not necessarily created, and when one is
20 created, it is not necessarily deep (within the object hierarchy, as described below). Notwithstanding, from the perspective of a model, it is appropriate consider a clone being made, and thus as used herein, the concept of a "clone" covers a clone that is actually created, a clone that is created in part, and/or a clone that is logically created from the model's perspective even though not necessarily created. The clone is what
25 actually gets used, and, by default, the clone cannot be modified.

As shown above, a changeable object also contains methods including Copy() and MakeUnchangeable(). An explicit call to the Copy() method creates a copy of the changeable with the copy's IsChangeable property set to true. This call does not alter the object on which the method is called. The MakeUnchangeable() method may be

235

called on any changeable, and modifies the `IsChangeable` property to make it false, (unless already false, in which case the call has no effect).

The above mechanisms facilitate a pattern for replacing a property. To change a changeable object, the `IsChangeable` property value needs to be true. Since a
5 qualified use of an object creates a clone that is not changeable, that object needs to be copied out via the `Copy()` method, changed, and used again. This is effectively replacing the initial object that was present with a new object that is a modified copy of the original. Examples of this are set forth below. Note that an in-use changeable is one that has been used, and thus by the definition above, is not changeable because
10 the `IsChangeable` property is set to false upon use. Thus, the changeable is not modified, but rather a changeable is copied out and replaced. Note that with changeables, there is only a single set of types, which is generally far more desirable from a programming perspective. Further, true mutability is provided by additional properties, as described below.

15 As described above, it is straightforward to create a brush, modify it and use it. An example of a simple use in a drawing operation is set forth below:

```
20 SolidColorBrush scb = new SolidColorBrush();  
    scb.Color = Colors.Red;  
    ctx.DrawRectangle(scb, ...); // this is a "use"  
    scb.Color = Colors.Green;  
    ctx.DrawRectangle(scb, ...); // this is a "use"
```

Executing the above instructions draws one red rectangle, and one green rectangle. Note that 'scb' is, in effect, cloned on each usage of it.

25 A more complex construction using a linear gradient brush (lgb), in which colors vary (e.g., linearly) from one stop to another, is set forth below:

```
30 LinearGradientBrush lgb = new LinearGradientBrush();  
    GradientStop gs1 = new GradientStop(Colors.Red, 0.3);  
    GradientStop gs2 = new GradientStop(Colors.Blue, 0.2);  
    lgb.Stops.Add(gs1); // this is a "use" of gs1  
    lgb.Stops.Add(gs2); // this is a "use" of gs2  
    Button2.Background = lgb; // this is a "use" of lgb
```

236

Here, the process is building values (GradientStops) and using them in definitions of more complex values.

Consider another example directed towards changing the opacity (which can range from zero to one) of a button's (Btn) background to 0.4. In this particular usage, the Background is copied into an object with IsChangeable property set to a true value, the background is modified, and set back in.

```
Brush b = Btn.Background.Copy(); // get IsChangeable copy
b.Opacity = 0.4;
10 Btn.Background = b; // "use" modified value
```

Note also that the assignment to "Btn.Background" in the last line severs any inherited or property sheet value that might have come in.

Modifications that are deeper within the object hierarchy appear no different to the user than modifications that are shallow:

```
Brush b = Btn2.Background.Copy();

LinearGradientBrush lgb = (LinearGradientBrush)b;
lgb.Stops[1].Color = Colors.Green;
20 lgb.Stops[0].Location = 0.2;

Btn2.Background = b;
```

Note that Copy() only needs to be called on the top level object, and not on the individual GradientStops. This is because the system takes care of ensuring that sub-objects of an object with a true IsChangeable property are themselves set with IsChangeable equal to true when they are accessed.

FIG. 15 shows a state diagram representing the states of a changeable object in basic usage, beginning with IsChangeable property equal to true when newly created. In general, the solid arrows show the states of the object transitioning from a current state to a target state, while any dashed arrow represents an operation that leaves the object the same, but creates a new object in the target state. In this state diagram, there

77

237

are two states, and the transitions occur when either `Copy()` or `MakeUnchangeable()` are called, and when an object is used in a manner that qualifies as a use, as described above. Note that calling the `Copy()` method from either state results in a new value with its `IsChangeable` property set to true, while calling `MakeUnchangeable()` results in the target value being `IsChangeable` set to false.

The above description presents a straightforward, self-consistent model, that describes the basic usage with just two states, `Copy()` and `MakeUnchangeable()` methods, and the notion of "using" a `Changeable` value. However, with respect to changes, the modification examples above are based on the concept of replacement, that is, copying an existing item, changing it in place, and copying it back. This implies heap allocation (which may be potentially significant, depending on how deep the change to be made is, and how broad the object itself is for the shallow clones), as well as an additional burden on the programmer to maintain some mechanism to find the path to the attribute to modify.

In accordance with an aspect of the present invention, to add support for the concept of true mutability of values, another property is added to the model, named `StatusOfNextUse` of type `UseStatus`. Note that a fundamental issue preventing mutability in the single property model is that a qualified use of a value unconditionally results in the resultant value with the `IsChangeable` property equal to false. The `StatusOfNextUse` property resolves this issue.

```
public enum System.Windows.UseStatus
{
    Unchangeable,
    ChangeableCopy
}
```

By default, the `StatusOfNextUse` property is `UseStatus.Unchangeable`, but may be set to `UseStatus.ChangeableCopy`, whereby a use of the value on which it is set will result in a clone object being made that has the `IsChangeable` property set to true. As a result, the object value can be modified in-place, without any additional heap allocation.

Further, because in-use values can be mutated in this model, a notification is provided when such changes happen, via a simple changed event. Furthermore, since the object is no longer immutable, context-affinity is provided through a `UIContext` member. Note that when the object is mutable, this has a null value. Otherwise it belongs to the `UIContext` in which it was created. The resultant `Changeable` class definition becomes:

```

public class System.Windows.Changeable
{
10  // Carryover from "basic usage" above
    public bool IsChangeable { get; } // defaults to true
    public Changeable Copy();
    public void MakeUnchangeable();

15  // New members
    public bool CanMakeUnchangeable { get; } // will
        // MakeUnchangeable succeed?
    public UseStatus StatusOfNextUse { get; set; } // defaults
        // to Unchangeable
20  public event EventHandler Changed { add; remove; }
    public UIContext UIContext { get; } // null when immutable
}

```

The above example of simple, shallow mutability described the requirements to change the opacity on a `Brush`, with the code needing to run each time opacity was to be changed. In contrast, with the mutability mechanism based on the `StatusOfNextUse` property, first the `Btn.Background` itself has an `IsChangeable` property value of true:

```

30  Brush b = Btn.Background.Copy(); // get IsChangeable copy
    b.StatusOfNextUse = UseStatus.ChangeableCopy;
    Btn.Background = b;

```

The above has used (in a qualified use) a value with `StatusOfNextUse` of `UseStatus.ChangeableCopy`, so the result itself is changeable. Once set up, the programmer can make modifications as desired, as in the following example:

5 `Btn.Background.Opacity = 0.3;`

The programmer can make such modifications as often as desired, and the modification will happen directly, without any object allocation on subsequent settings.

10 Note that the above example does not describe how the `Btn.Background` came into being in the first place, and thus a copy of it needs to be made via the `Copy()` method. In a deliberate mutability situation where the programmer wants to create a background to be modified, a better way to do this might be directly, as in the following example:

15 `SolidColorBrush b = new SolidColorBrush();`
`b.Color = Colors.Red;`
`b.StatusOfNextUse = UseStatus.ChangeableCopy; // set this up`
`// directly`
`Btn.Background = b;`

20

At this point, the programmer can specify the opacity (`Btn.Background.Opacity = ...`) whenever desired, as the brush was created initially as `StatusOfNextUse` equal to `UseStatus.ChangeableCopy`.

25 It should be noted that using the replacement-based model rather than the mutating-based model is not particularly difficult given the above examples. This is because the changes are made at the first level, and it may not appear prohibitively expensive to always replace rather than mutate. Indeed, this is valid technique when only limited mutability is desired. However, when changes are made to values deeper an object, the mutating model is clearly superior.

30 As an example of such deeper mutability, consider the `LinearGradientBrush` (`lgb`), wherein the programmer repeatedly wants to change the color of the seventh

stop (lgb.Stops[6]). The programmer can use the same instructions as above to install a mutable version into Btn.Background:

```

5  Brush b = Btn.Background.Copy(); // get IsChangeable copy
    b.StatusOfNextUse = UseStatus.ChangeableCopy;
    Btn.Background = b;

```

Thereafter, the programmer can repeatedly make the desired changes:

```

10 LinearGradientBrush lgb = ((LinearGradientBrush)Btn.Background);
    lgb.Stops[6].Color = ... some new color ...;

```

The programmer can also access the "lgb" variable once, store it away, and then repeatedly set into it, which is very efficient.

15 FIG. 16 is an extension of the state diagram of FIG. 15 with the additional state represented by the StatusOfNextUse property added. Note that the model is only slightly more complicated, as the basic state diagram of FIG. 16 has two states and seven transitions, whereas the mutability state diagram has three states and eleven transitions. As can be seen from FIG. 15, a significant addition is the
20 (StatusOfNextUse == ChangeableCopy) state, and the "Use" transition out of that state, which results in a new copy with the IsChangeable property value set to true.

As in FIG. 15, calling the Copy() method results in a new value with an IsChangeable property equal to true, with the StatusOfNextUse property set to Unchangeable. Similarly, the MakeUnchangeable() method call results in the target
25 value for the IsChangeable property set to false. Note that even though flexibility has been added via the mutability, these constants have not changed.

There are some situations where use as StatusOfNextUse equal to ChangeableCopy will not be allowed, because subsequent modifications are truly not well-defined, or are explicitly disallowed. Examples of this include attempting to
30 modify values in a shared default style sheet, or setting non-local property engine properties. In such situations, the subsystems that disallow such use will either choose to raise an exception, or make the value unchangeable themselves. It is recommended

that an exception be raised as a more clear indication to the programmer of what has occurred and thereby avoid later confusion.

Further, there are situations in which a changeable object cannot be made unchangeable. Examples include a VisualBrush, (as described in the aforementioned United States Patent Application Serial No. 10/402,268), in which the underlying visual can not be restricted from changing, and therefore it would be nonsensical to state that the VisualBrush is "unchangeable." Animations and VideoData (because they are time varying) are also examples. Attempts to call MakeUnchangeable() on such objects will raise exceptions, or worse, may leave the object in a bad state, as portions of it may have been made unchangeable while others have not. These problems can be avoided via another property, CanMakeUnchangeable. If this property returns a "true" value, then MakeUnchangeable() is guaranteed to succeed, provided no changes to the object happen between these calls.

There is a conflict in semantics that occasionally occurs between StatusOfNextUse and CanMakeUnchangeable. If CanMakeUnchangeable equals false, then a value of UseStatus.Unchangeable for StatusOfNextUse really does not make sense, as the next use cannot be unchangeable. Therefore, when StatusOfNextUse is queried when CanMakeUnchangeable equals false, it never will return UseStatus.Unchangeable. Instead, when it otherwise would have returned UseStatus.Unchangeable, it returns UseStatus.ChangeableCopy.

The above presents a model where every (qualified) use of a changeable (IsChangeable equals true) object results in a copy of that object being made, and, depending on the value of StatusOfNextUse, that "use" may or may not itself be changeable. What the above model does not provide is the use of a value in multiple places, and maintaining a shared reference to that value. For example, in the above model a programmer cannot create a LinearGradientBrush, use it on two Button controls, and then change the LinearGradientBrush once to affect both controls. Rather the programmer would need to use it twice, get the brush back out of the controls, and then set each one independently. In general, this model turns out to be the most expected and/or least surprising to programmers, but there are scenarios where additional functionality is desirable.

242

One such scenario is in Animation, where, if a programmer wants to create a scene with n elements that each respond to the same timeline, that timeline needs to be cloned n times, and asked to `BeginIn()` n times. A much better approach, from both a performance and efficiency point of view, as well as for programming model convenience, is to share out a reference to a single timeline, invoke `BeginIn()` on it, and have it propagate as appropriate.

To enable this scenario, a third value, `ChangeableReference`, is provided with the `UseStatus` enumeration. `UseStatus` now looks like:

```
10 public enum System.Windows.UseStatus
    {
        Unchangeable,
        ChangeableCopy,
        ChangeableReference
15 }
```

When a changeable object that has `StatusOfNextUse` set to `UseStatus.ChangeableReference` is used (in a qualified manner), that value is no longer copied. Rather, a reference to the existing value is handed out, and subsequent modifications to that value (or any previously or subsequently handed out references) will affect the result of the use. In other words, the changeable value is now shared with potentially any number of uses.

The following is an example of element level usage:

```

Brush b = new SolidColorBrush(...);
b.Color = Colors.Yellow;
b.StatusOfNextUse = UseStatus.ChangeableReference;
Btn1.Background = b;
5 Btn2.Background = Btn1.Background;
  Btn3.Background = Btn2.Background;
  Btn4.Background = b;
  // at this point, all four buttons are yellow

10

((SolidColorBrush)Btn3.Background).Color = Colors.Purple;
// They change to purple, and they are notified.

```

15 In an above example, a simple drawing operation was described which generated two rectangles, one red, and one blue:

```

SolidColorBrush scb = new SolidColorBrush();
scb.Color = Colors.Red;
ctx.DrawRectangle(scb, ...); // this is a "use"
scb.Color = Colors.Green;
20 ctx.DrawRectangle(scb, ...); // this is a "use"

```

This is the desired behavior. However, if a programmer instead wanted the brush to be shared, and still be changeable, the following instructions could be used:

```

25 SolidColorBrush scb = new SolidColorBrush();
  scb.UseResult = UseResult.ChangeableReference;
  scb.Color = Colors.Red;
  ctx.DrawRectangle(scb, ...); // this is a "use"
  scb.Color = Colors.Green; // initial rectangle becomes green
30 ctx.DrawRectangle(scb, ...); // this is a "use"

```


Here, both rectangles are green. If later the color is changed, e.g., `scb.Color = Colors.Yellow`, both rectangles will become yellow. Note that `ctx.DrawRectangle(...)` appears to be an immediate mode drawing command, however it is actually building a display list / metafile to be retained and subsequently displayed.

5 From a user-model point of view, the `ChangeableReference` mode guarantees that the parties that are using a changeable object will be notified of any changes to that value. This will be done through the "Changed" event which, like other events, is a multicast delegate. To implement, the system need to be sure that multiple uses with a single notification sink do not notify that sink for each use. Further, cleanup
10 mechanisms have requirements when removing items, so as to only remove a sink when the uses connected to that sink are gone. One approach for doing this is to reference count delegates. The current implementation may achieve these requirements via a private data structure, e.g., `RefCountedMulticastEventHandler`.

FIG. 17 is a state diagram based on FIGS. 15 and 16 but having a
15 `ChangeableReference` state (via another setting in the `StatusOfNextUse` property) added. Note that in keeping with an aspect of the present invention, the `ChangeableReference` state and the `Use` transition off of that node does not make a copy. Rather, a qualified use results in the status of next use property remaining in the changeable reference state, thereby providing true mutability. Further, note that while
20 FIG. 17 is more complex than those of FIGS. 15 and 16, the behavior of the `Copy()` and `MakeUnchangeable()` methods remain constant; the `Copy()` method still results in a new value object having the `IsChangeable` property of true and `StatusOfNextUse` property of `Unchangeable`, and the `MakeUnchangeable()` method still results in the target value object having a `IsChangeable` property of false.

25 It should be noted that along with the advantages of a single set of types, the present invention provides significant flexibility to programmers. For example, most values constructed by an application ordinarily will not be mutable, as immutable values are consume less resources. However, as described above, mutability is available, giving programmers a powerful and intuitive way to change values,
30 particularly deep values, with high performance. Note also that although not represented in FIG. 17, the state in which a new type is created (the `IsChangeable` property of true, the `StatusOfNextUse` property of `Unchangeable`) is only one possible

85

245

default state. Thus, in alternative implementations, the type may be in another state following creation, e.g., the `IsChangeable` property of true, the `StatusOfNextUse` property of `ChangeableReference`) such as to default to mutable values.

Turning to an explanation of the operation of the present invention, the present invention provides significant advantages when dealing with deep properties of an object, referred to as dotting-down. For example, consider the following:

```
10 GeometryCollection g;  
    g.Geometries[12].Figures[2].Segments[0].Points[17] =  
        new Point(0.2, 0.3);
```

The deep access into the geometry object 'g' is an example of what is referred to as dotting-down. Note that the access to the properties (`Geometries`, `[12]`, `Figures`, `[2]`, `Segments`, `[0]`, and `Points`) are calling property getters, not setters; `[17]` is the only property access that results in a setter being called. Programming languages generally cannot distinguish between access to a property for the sake of setting a property value deeper down versus access for reading a value deeper down.

When dotting-down starts from an unchangeable object, the local member variable is accessed. An example includes accessing an element that has not been made changeable through use of the "`ChangeableValue`" property.

When a property get happens from a changeable object, the resultant value is changeable as well, so that it can be mutated. To this end, the property getter on the parent returns the sub-object directly if already changeable, or makes a shallow clone of the sub-object, sets that into the local member, and returns that clone. Note that these attributes make the above code, after running the first time and allocating and assigning shallow clones, free in terms of not requiring heap allocation.

An aspect of the present invention is that on-demand, shallow cloning is performed only when necessary. This maximizes sharing, minimizes heap allocation, allows for modifications without heap allocation, and does not force the concept of cloning onto the user model. This becomes more important with deeper trees, and when working in three dimensions. To this end, the `Copy()` method provides the illusion of a deep copy, but really first only makes a shallow copy, and then lazily

get

5

```
event EventHandler Repeated { add; remove; }
event EventHandler Resumed { add; remove; }
event EventHandler Reversed { add; remove; }
event EventHandler Searched { add; remove; }
}
```

The following table summarizes the semantics of the ITimingControl interface:

Method, Property or Event	Meaning
Acceleration	A value between 0 and 1 representing the fraction of the simple duration spent in the time acceleration phase. The sum of this attribute and the Deceleration attribute may not exceed 1.
AutoReverse	If this attribute is "True" then the timeline progresses from beginning to end and then immediately progresses backwards from end to beginning. In that case, the timeline will be active for twice the amount of time specified by the Duration attribute.
Begin	The time at which this timeline should begin. By default this time is relative to the parent timeline's begin time, but the offset may also be specified to be relative to some other timeline's begin or end time. In the latter case, the other timeline is parented to the same timeline as this one.
BeginIn	Triggers an interactive begin at the specified point in time in the future or past. The parameter is in the frame of reference of this timeline's parent timeline. If the parent timeline is not active, this method has no effect.
Begun	Raised whenever the object enters a period in which its internal state is continually changing.
Changed	Raised by the modifier whenever its internal state

231

	changes.
Ended	Raised whenever the object leaves a period in which its internal state is continually changing.
CurrentRepeat	The current iteration of the timeline, if it repeats. The first iteration is iteration 1. If <code>IsOverridingBaseValue</code> is false this property returns 0.
CurrentTime	The current time local to this timeline. If <code>IsOverridingBaseValue</code> is false this property returns <code>Time.Unspecified</code> .
Deceleration	A value between 0 and 1 representing the fraction of the simple duration spent in the time deceleration phase. The sum of this attribute and the <code>Acceleration</code> attribute may not exceed 1.
Disable	Disables this timeline, effectively removing it from the timing tree.
Duration	The duration of a single period from begin to end.
Enable	Enables this timeline, effectively inserting it into the timing tree. This method has no effect if this is an auto-parented timeline and a default parent has not been specified.
End	The maximum end time for this timeline. If this value is less than the sum of the <code>Begin</code> and <code>Duration</code> properties, then the activation period is cut short by this attribute. In addition, all begins (scheduled or interactive) past the time specified by this attribute are ignored.
EndIn	Triggers an interactive end at the specified point in time in the future or past. The parameter is in the frame of reference of this timeline's parent timeline. If the parent timeline is not active, this method has no effect.
EndSync	This attribute is used to define the implicit duration of

298

	a timeline, which is used if the Duration attribute is not set explicitly. The implicit duration of a timeline may be defined by the timed object that it controls or by other timelines that may be parented to it.
Fill	The behavior of the timeline after the end time passes. By default, the timeline is only "on" from begin to end, but if this attribute is set to "Freeze" then the timeline remains on past the end time. In that case, the progress value after the end time is equal to whatever it was at the end time. The possible values are Remove (the global default), Freeze, Hold, Transition and Auto.
FillDefault	The default value for the Fill attribute. If the Fill attribute is not specified then this attribute is used to determine the fill behavior. In addition, this default is inherited by timelines parented to this one, unless they have their own FillDefault attribute set. The possible values are the same as for the Fill attribute.
IsChanging	True if the timeline is active, false otherwise.
IsEnabled	True if the timeline is part of a timing sub-tree, false otherwise. If this property is true this doesn't guarantee that the sub-tree this timeline is a part of is, itself, enabled.
IsForwardProgressing	True if progress in this timeline moves from 0 to 1, as compared to wall-clock time. This property takes into account the effect of being nested in potentially reversed timelines. If IsOverridingBaseValue is false this property returns the same value as that which this timeline's parent timeline would return.
IsOverridingBaseValue	True if the timeline is active or in a fill period.
IsPaused	True if the timeline is active, but in a paused state.
IsReversed	True if the timeline is in a reversed period, as seen

233

249

from the timeline's own local frame of reference. This property does not take into account the effect of being nested in potentially reversed timelines. If `IsOverridingBaseValue` is false this property returns false.

ParentTimeline The timeline that is the timing parent of this timeline. This may be a reference to any other timeline, or one of two special reference values:

`Timeline.VisualParent` or `Timeline.RootTimeline`. If this property is set to `Timeline.VisualParent` then this timeline is auto-parented on use to the timeline associated with the Visual in which it is used (if the Visual doesn't have an associated `DefaultTimeline`, then the parent Visual is inspected, recursively). If it is set to `Timeline.RootTimeline` then this timeline is auto-parented on use to the "root" of the timing tree.

Pause Pauses this timeline and all of its children timelines. If this timeline is not active this method has no effect.

Paused Raised by the timeline whenever it or one of its descendants is paused.

Progress The current progress value of the timeline. If `IsOverridingBaseValue` is false this property returns 0. In all cases, the return value of this property is always a value between 0 and 1, inclusive.

RepeatCount The number of times a begin to end period should be repeated. This may be a fractional value, as well as the special value `float.PositiveInfinity` to indicate that the timeline should repeat forever. If both this attribute and the `RepeatDuration` attribute are specified, the total active duration is the minimum of the two.

RepeatDuration The length of time for which a begin to end period

23

23

	should be repeated. This may imply a fractional repeat count, or it may be the special value <code>Time.Indefinite</code> to indicate that the timeline should repeat forever. If both this attribute and the <code>RepeatCount</code> attribute are specified, the total active duration is the minimum of the two.
Repeated	Raised by the timeline whenever it repeats its simple duration.
Restart	The behavior of the timeline when a second (or later) begin time is reached. By default, a begin time interrupts any active period and goes back to time $t=0$ for the timeline, but if this attribute is set to <code>WhenNotActive</code> then a begin time that would interrupt an active period is ignored. The possible values are <code>Always</code> , <code>WhenNotActive</code> and <code>Never</code> .
RestartDefault	The default value for the <code>Restart</code> attribute. If the <code>Restart</code> attribute is not specified then this attribute is used to determine the restart behavior. In addition, this default is inherited by timelines parented to this one, unless they have their own <code>RestartDefault</code> attribute set. The possible values are the same as for the <code>Restart</code> attribute.
Resume	Resumes this timeline and all of its children timelines. If this timeline is not active and paused this method has no effect.
Resumed	Raised by the timeline whenever it is resumed.
Reversed	Raised by the timeline whenever the direction of time changes.
Seek	Changes the current time for this timeline, which may have an effect on all of its children timelines. If this timeline is not active this method has no effect.
Seeked	Raised by the timeline whenever its time changes as a

235 -

result of a seek operation.

Speed

The relative speed at which time should pass for this timeline, compared to its parent timeline. E.g., A value of 1 means normal speed, whereas a value of 2 means that time elapses twice as fast (and, therefore, the perceived duration ends up being only half that specified by the Duration attribute). This value may be negative, in which case time flows backwards in this timeline, from end to begin times, as if the parent timeline was reversed.

Graphics scenes can be animated by specifying animated parameters to some rendering operations, or by adding animations to certain element properties. An animation is a function that takes some arbitrary set of inputs (at least one of which generally being a timeline) and produces an output of the right type to be passed to a rendering operation. For example, a PointAnimation converts a timeline progress value into a Point value type. At the same time, various rendering operations that take one or more Point values as parameters can also receive a PointAnimation instead of a Point, in which case the animation function is evaluated at each frame to calculate the Point to use in that frame.

Animations are grouped into collections. An animation collection works as a pipeline, taking as an input the base value of a property and producing as its output the current value that should be used for that property. The collection chains zero or more animation objects, each supporting similar semantics of taking an input value and producing an output of similar type. The pipeline is evaluated at regular intervals, and the output is used in rendering operations, producing the effect of animation.

Since the values that can be animated have various types, there are also various differently typed animations. However, all animations follow a common pattern and all implement a set of common interfaces. Animation objects are organized into three groups of classes, namely modifiers, timed modifiers and animations.

A straightforward animation interpolates a value between the begin and end points. When both the begin and end points are specified, the base value is ignored for

236

752

the time the animation is "on". When the animation is "off", the value of the property may revert to the base value. Note that an animation is "on" as long as its associated timeline is "on". Therefore, a from-to animation can be made to permanently override the base value by setting the Fill timing attribute to "Freeze". FIG. 36 shows a point animated in y with From=10 and To=70.

If only the begin or the end point is specified, but not both, the base value of the property is used for the value of the other point. While this seems redundant with the previous example, there is a key difference in that in this case, the base value is not ignored, but composed with the animation. This can produce interesting effects if the base value is changing (because the property is being modified by another process), or if the animation is chained to another animation.

Another way to specify the animation function is to specify a delta from the base value. This is conceptually similar to a from-to animation that interpolates from the base value to the base value plus the delta. However, in this case both the begin and end points are composed with the base value.

If the timeline associated with an animation is set to repeat, the animation runs from begin to end several times. FIG. 37 shows an point animated in y with From=10, By=60 and RepeatCount=2. Instead of repeating the same trajectory on every iteration, an animation can be programmed to accumulate the effect of each iteration, in essence composing with itself. FIG. 38 shows a point animated in y with From=10, By=60, RepeatCount=2 and IsAccumulating=True.

Although the default behavior of a from-to animation is to ignore the base value of the animated property, this behavior can be changed to an additive behavior, where the from and to values are both deltas from the base value.

The following table summarizes the basic animation types.

Type	Output value
From	The "From" value at t=0 and the base value at t=1.
To	The base value at t=0 and the "To" value at t=1.
From-To	The "From" value at t=0 and the "To" value at t=1.
By	The base value at t=0 and the sum of the base value and the "By" value at t=1.
From-By	The "From" value at t=0 and the sum of the "From" and "By"

753

253

values at $t=1$.

With basic animations, the output value at the begin and end points are specified, and a linear interpolation is used to compute the values in between. For more complex animation functions, a list of values can be specified instead. Each value corresponds to a key frame. In a simple case, these key frames occur at regular intervals. Animations can also be programmed to use a paced spacing between the key frames. In the paced interpolation method, the space between each pair of key frames is proportional to the ratio of the "distance" between the two key values to the "total distance" covered by the animation. This is possible for those animations whose types have a meaningful concept of "distance," such as, for example, float or point animations. In such a case, the interpolation between key frames is linear. A third option is to not interpolate at all, in which case the output value function is discrete. FIG. 39 shows a point animated in y with $KeyValues=10,90,70$ and various interpolation methods.

For additional control, the time for each key frame can be explicitly specified. The interpolation between the key frames can be linear or discrete. The key times are specified as percentages of the total animation duration, and must cover the entire period. In other words, the first key time is 0, and for linear interpolation the last key time is 1. FIG. 40 shows a point animated in y with $KeyValues=10,90,50$ and $KeyTimes=0,.2,1$.

For still further control over the interpolation, a set of cubic Bezier curves can be used to describe the time curve used for the animation. This should not be confused with a Bezier curve rendered on screen; the curve is used to modify the shape of the timing curve, but the key frame values still interpolate linearly over the progress value. This spline interpolation method adds a filter that converts the linear 0-1 progress value provided by the timeline associated with the animation into a non-linear 0-1 progress curve.

The following table contains a list of animation-specific attributes and their meanings. This list is a template followed by all animations objects. Where the type of an attribute is "<ValueType>", a real object would expose the attribute with a type matching the animation type. For example, a `ColorAnimation` object types these

736

attributes as "Color." In addition to the attributes listed below, animation objects support the attributes specified in the `ITimingAttributes` interface.

Attribute	Type	Meaning
By	<ValueType>	The delta value at the end of the animation. The value at the beginning is either the From value, if specified, or the base value of the property.
From	<ValueType>	The initial value of the animation.
InterpolationMethod	InterpolationMethod	The method used to interpolate between key values. The possible values are Discrete, Linear, Paced or Spline.
KeySplines	KeySplineCollection	A set of Bezier control points associated with a KeyTimes list, which defines a cubic function that controls the interval pacing of the animation. This list must contain one less element than the KeyTimes list. This list is only used if the InterpolationMethod attribute is set to Spline.
KeyTimes	KeyTimeCollection	A list of time values used to control the pacing of the animation. This list must contain the same

15

number of elements as the KeyValues list. The list is ordered in increasing time values, and the first value in this list needs to be 0 and the last 1, unless InterpolationMethod is set to Discrete, in which case the last value may be anything less than or equal to 1.

KeyValues	<ValueType>KeyValueCollection	A list of values for the animation.
To	<ValueType>	The value at the end of the animation.

The Animatable class is derived from the Changeable class. It is used as a base class by any object or object collection that may be animated or contain animated values, such as a media resource. Modifiers, TimedModifiers, and Animations
5 ironically derive from Changeable instead of Animatable because their individual properties are not animatable.

```
public abstract class Animatable : Changeable
{
    public abstract bool HasAnimations { get; }
10    public abstract bool IsAnimating { get; }
    public abstract bool IsOverridingBaseValue { get; }

    public abstract Animatable GetCurrentValue();
    public abstract void SetDefaultParentTimeline(Timeline defaultParentTimeline);
15 }
```

Method, Property or	Meaning

240

Event

- HasAnimations** True if the object may vary with time. In general, this property is true if the object is holding on to any animation collections.
- IsAnimating** True if any of the animations in the object are changing (see **Modifier.IsChanging**).
- IsOverridingBaseValue** True if any of the animations in the object are changing or in a fill state and therefore currently active and modifying the object.
- GetCurrentValue** Returns an object that has the same value as the instantaneous value of this object, but which doesn't vary over time. If the **DoesChange** property is false, the **CurrentValue** property may return the object itself rather than a new copy.
- SetDefaultParentTimeline** The timeline that is the parent of any auto-parented timelines. If this property is set, any auto-parented timelines are re-parented, but a new clone is not created either for the timelines or for this object.

Modifier classes, and therefore **TimedModifiers** and **Animations**, will derive from **Changeable** instead of **Animatable** because their individual properties should not be animated. This enforces the fact that programmers should not be able to animate the **From** property on an animation.

Modifier classes cannot have a **StatusOfNextUse** property value of **Unchangeable**. The default value for **StatusOfNextUse** for **Modifiers** is **ChangeableCopy**, however it can also be set to **ChangeableReference** if the user wishes to re-use a **Modifier**. If the user sets the **StatusOfNextUse** to **ChangeableReference** an exception is thrown if any attached **Modifier** does not have a **ParentTimeline** property set. This will prevent situations having conflicting inherited parent timelines. Non-animated, non-changeable branches of an **Animatable** may have a **StatusOfNextUse** value of **Unchangeable**, and may be made unchangeable on use.

24

Properties on a Modifier class, such as From, To, or By remain changeable throughout the lifetime of that Modifier.

5 Modifiers are changeable throughout their lifetime, so MakeUnchangeable will throw an exception on these classes. For an Animatable that currently contains an animation, MakeUnchangeable will throw an exception.

If the user signs up for Changed notifications on an Animatable class, the user will receive notifications for changes caused either by property changes or through the nature of animation. That is, the user will receive changed notifications when the Timelines associated with animations used by the Animatable are Seeked or moved
10 forward as they are on each frame that is presented.

In the case of independently animated properties (e.g., Opacity) or Animatables (e.g., SolidColorBrush), the Changed notifications sent to any user who has provided a handler will occur at the UI thread frame rate not at the compositor frame rate. The
15 exact value of an animation in this case is not guaranteed to be exactly what is on the screen, although the values should be close.

If the animation is dependent or MIL dependent it is possible to get a value that matches what will be on the screen although it is not currently possible to tell which notification corresponds to the rendering pass and therefore which one reflects the
20 value about to be displayed. If the timing tree is changed during the rendering pass, as may often occur, it is possible for the user to receive multiple notifications, and therefore even less likely that the user will know which one corresponds to the eventual value on the screen.

A modifier is an object that implements a GetValue method which takes as an
25 input an object, called the "base value", of a certain type, and returns another object of the same type as the input. The value of the output depends both on the input and on the internal state of the modifier. In particular, this means that calling GetValue more than once with the same input is not guaranteed to return the same output. Graphical animation occurs when a modifier's GetValue method is called once per frame
30 producing a new value for each frame.

In a general case there are no guarantees about the return value of the GetValue method, and every time the method is called it may return a different value. Objects

consuming modifiers may assume that this is the case and call the modifier repeatedly, as in the following example:

```

class MyObject
{
5   private Modifier myModifier;
   private object myBaseValue;

   public DoSomeWork()
   {
10    object currentValue = myModifier.GetValue(myBaseValue);

        DoSomethingWithCurrentValue(currentValue);
        PostCallbackToDoSomeWork(); // Causes this method to be called again
   }
15 }

```

In practice, however, there may be times when a modifier expects to produce the same output given the same input, depending on its internal state. A modifier is said to "be changing" when it is in a period in which the return value of `GetValue` may be different on each invocation. It is "not changing" when the return value of `GetValue` is the same on each invocation. If a modifier is "not changing" a user of that modifier may safely cache the return value of the `GetValue` method, and perhaps avoid evaluating the `GetValue` method repeatedly and unnecessarily, as in the following example:

```

25 class MyObject
   {
       private Modifier myModifier;
       private object myBaseValue;

30   public Initialize()
       {
           myModifier.ChangeBegun += new EventHandler(this.OnChangeBegun);

```

239

```

    }

    public DoSomeWork()
    {
5       object currentValue = myModifier.GetValue(myBaseValue);

        DoSomethingWithCurrentValue(currentValue);
        if(myModifier.IsChanging)
        {
10         PostCallbackToDoSomeWork(); // Do more work shortly
        }
        else
        {
            // Do nothing, since the modifier isn't going to change
15         // any time soon. If it does start to change, the delegate
            // above will restart our processing loop.
        }
    }

20    public OnChangeBegun()
    {
        // The modifier is changing again, so start doing work on
        // a schedule again
        PostCallbackToDoSomeWork();
25    }
    }

```

An abstract Modifier class is implemented, from which modifiers need to inherit. This class provides default implementations for all but the GetValue and

30 GetUniqueInstance methods:

```
public abstract class System.Windows.Media.Animation.Modifier
```



```
{
    public virtual Timeline ParentTimeline { get; set; }
    public virtual bool IsChanging { get; }
    public virtual bool IsOverridingBaseValue { get; }
5    Modifier abstract GetUniqueModifierInstance(Timeline defaultParentTimeline);
    object abstract GetValue(object baseValue);
    public virtual bool UsesBaseValue { get; }

    // Event notifications
10    public virtual event EventHandler Changed { add; remove; }
}
```

The following table summarizes the semantics of the Modifier class:

Method, Property or Event	Meaning
Changed	Raised by the modifier whenever its internal state changes.
ParentTimeline	The timeline that is the parent of any auto-parented timelines in this modifier. If this property is set, any auto-parented timelines in this modifier are reparented to the new parent timeline.
GetUniqueInstance	Returns an instance of this modifier that can maintain its own run-time state separately from other instances. If this modifier contains auto-parented timelines, the returned instance has those timelines parented to the timeline passed in as a parameter.
GetValue	Calculates the current output value of this modifier, based on the base value passed in as an argument and the internal state of the modifier. When the IsOverridingBaseValue property is false, this function is guaranteed to return the base value.
IsChanging	True if the modifier is currently changing, false if it's in a

period of non-change. This flag is best used in conjunction with the **ChangeBegan** and **ChangeEnded** events. If this flag is true, the **IsOverridingBaseValue** also needs to be true.

IsOverridingBaseValue True if the return value of the **GetValue** method is currently being affected by the modifier. When this value is false, **GetValue** is guaranteed to return the same object that is passed to it as an argument. Note that a modifier may be overriding the base value but not changing.

UsesBaseValue True if the return value of **GetValue** depends on the base value. If this property is false, it means that the modifier ignores the base value altogether. If the modifier is in a list, this property allows an optimization where only a subset of the modifiers need to be evaluated in some cases.

In addition, a set of type-specific classes is implemented that inherit from **Modifier** but expose type-safe versions of the interface methods. The following example shows the **FloatModifier** class:

```
5 public abstract class System.Windows.Media.Animation.FloatModifier : Modifier
  {
    // Type-specific methods
    public sealed override object GetValue(object baseValue)
    {
10      return GetValue((float)baseValue);
    }
    public abstract float GetValue(float baseValue);
  }
```

15 A timed modifier is a modifier whose behavior is controlled at least partially by a **Timeline** object. The aforementioned modifier rules apply, but in addition a timed modifier implements the **ITimingControl** interface to expose controlling the modifier's timeline. There is no abstract **TimeModifier** class. Instead, type-specific

262

classes inherit from the type-specific Modifier classes. The following example shows the FloatTimedModifier class:

```
public abstract class System.Windows.Media.Animation.FloatTimedModifier :  
    FloatModifier, ITimingControl  
5 {  
    protected FloatTimedModifier(FloatTimedModifier example);  
  
    // FloatModifier methods, properties and events  
    public override Timeline ParentTimeline { get; set; }  
10    public override bool IsChanging { get; }  
    public override bool IsOverridingBaseValue { get; }  
    public override FloatModifier GetUniqueInstance(Timeline defaultParentTimeline);  
    public override event EventHandler Changed { add; remove; }  
  
15    // ITimingControl methods, properties and events  
    double Acceleration { get; set; }  
    bool AutoReverse { get; set; }  
    TimeSyncValue Begin { get; set; }  
    double Deceleration { get; set; }  
20    Time Duration { get; set; }  
    TimeSyncValue End { get; set; }  
    TimeEndSync EndSync { get; set; }  
    TimeFill Fill { get; set; }  
    TimeFill FillDefault { get; set; }  
25    Timeline ParentTimeline { get; set; }  
    double RepeatCount { get; set; }  
    Time RepeatDuration { get; set; }  
    TimeRestart Restart { get; set; }  
    TimeRestart RestartDefault { get; set; }  
30    double Speed { get; set; }  
    int CurrentRepeat { get; }  
    Time CurrentTime { get; }
```

259

```

    bool IsForwardProgressing { get; }
    bool IsPaused { get; }
    bool IsReversed { get; }
    double Progress { get; }
5   void BeginIn(Time offset);
    void EndIn(Time offset);
    void Pause();
    void Resume();
    void Seek(Time offset, TimeSeekOrigin origin);
10  event EventHandler ChangeBegan { add; remove; }
    event EventHandler ChangeEnded { add; remove; }
    event EventHandler Paused { add; remove; }
    event EventHandler Repeated { add; remove; }
    event EventHandler Resumed { add; remove; }
15  event EventHandler Reversed { add; remove; }
    event EventHandler Searched { add; remove; }

    // Data
    protected Timeline Timeline;
20 }

```

Note that the **Modifier** and **ITimingControl** interface have some similar methods, properties and events. A **Timed Modifier** exposes a single implementation for those. A **Timed Modifier** is free to implement **ITimingControl** by forwarding all calls to the controlling **Timeline**, though it is not required to do so. The default implementation of **ITimingControl** provided by the type-specific **Timed Modifier** implementations does forward calls to the controlling **Timeline**.

An animation is a timed modifier implementing a particular animation function.

```

30 public sealed class System.Windows.Media.Animation.FloatAnimation :
    FloatTimedModifier
    {

```

26A

```

public FloatAnimation(float from, float to, Time duration);
public FloatAnimation(float from, float to, Time duration, TimeFill fill);
public FloatAnimation(float to, Time duration);
public FloatAnimation(float to, Time duration, TimeFill fill);

// All FloatTimedModifier methods, properties and events are inherited,
// plus the following are added:

public float By { get; set; }
public float From { get; set; }
public InterpolationMethod InterpolationMethod { get; set; }
public bool IsAccumulating { get; set; }
public KeySplineEnumerator KeySplines { get; set; }
public KeyTimeEnumerator KeyTimes { get; set; }
public FloatKeyValueEnumerator KeyValues { get; set; }
public float To { get; set; }
}

```

An animation collection is a list of animation objects (inheriting from <Type>Modifier) where the output of the GetValue method from the first object is used as the base value parameter for the GetValue method on the second object, and so on. For flexibility, the objects contained in an animation collection are actually of type type-specified Modifier. The collection as a whole supports a GetValue method which looks like IModifier.GetValue. In fact, animation collections support most of the IModifier interface, but they do not actually implement IModifier because they do not support the "UsesBaseValue" property (this property is always assumed to be "true" for the collection as a whole).

```

public sealed class System.Windows.Media.Animation.FloatAnimationCollection :
    ICollection
{
    public Timeline DefaultParentTimeline { get; set; }
    public bool IsChanging { get; }
}

```

26A

5

10

```
public bool IsOverridingBaseValue { get; }
public FloatAnimationCollection GetUniqueInstance(Timeline
    defaultParentTimeline);
public float GetValue(float baseValue);

// Event notifications
public event TimeEventHandler ChangeBegun { add; remove; }
public event TimeEventHandler Changed { add; remove; }
public event TimeEventHandler ChangeEnded { add; remove; }

// ICollection MPEs supported
public FloatModifier this[int index] { get; }
}
```

15

Events fired from animation collections are coalesced.

Path animations are a specialization of the TimedMatrixModifier class. A MatrixModifier can be used along with a MatrixTransform. A MatrixTransform has a Matrix property, and MatrixAnimations property, and since a PathAnimation is a MatrixModifier, it can be used as a MatrixAnimation.

20

25

```
public sealed class System.Windows.Media.Animation.PathAnimation :
    TimedMatrixModifier
{
    public PathGeometry PathGeometry { get; set; }
    public bool DoesRotateWithTangent { get; set; }
}
```

Method, Property or Event	Meaning
Geometry	This can be any geometry. For ellipses, an appropriate starting point for progress 0 is selected. If the geometry

266

has many sub geometries, each of their paths will be moved along in turn in the order they were defined inside the geometry.

DoesRotateWithTangent If this property is set to false, the object will move along the geometry path with no rotation. If set to true, the object will rotate to match the tangent of the path at any given location.

Markup usage:

```
5  <Canvas>
    <Canvas.TransformEffect>
    <MatrixTransform> <!-- default Matrix base value is identity -->
    <MatrixTransform.Matrix>
    <PathAnimation Begin="0" Duration="10" DoesRotateWithTangent="true"/>
    <PathAnimation.Geometry>
    <PathGeometry> ... </PathGeometry>
10 </PathAnimation.Geometry>
    </PathAnimation>
    </MatrixTransform.Matrix>
    </MatrixTransform>
    </Canvas.TransformEffect>
15 </Canvas>
```

Every resource, method or object that may be animated follows a number of rules, including that it implements the Animatable interface. For every animatable property (or parameter) called "Foo", of type "Bar", there is another property (or parameter) called "FooAnimations", of type "BarAnimationCollection". Wherever animation is desirable, animation collections are used. Basic Modifiers or Animation objects are not directly used because that precludes animation composition.

Resources can be animated by adding animation collections to individual properties. The following example shows how to create a SolidColorBrush with an animate color:

253

```

ColorAnimation anim = new ColorAnimation ();
animBuilder.From = new Color(1.0f, 0.0f, 0.0f, 0.0f);
animBuilder.Begin = new Time(0);
5 animBuilder.To = new Color(1.0f, 1.0f, 0.0f, 0.0f);
animBuilder.Duration = new Time(1000);
animBuilder.AutoReverse = true;
animBuilder.RepeatDuration = Time.Indefinite;

10 SolidColorBrush brush = new SolidColorBrush ();
brush.Color = new Color(1.0f, 0.0f, 0.0f, 0.0f);
brush.ColorAnimations = anim;

```

15 **Animate resources can be used in rendering operations or as values for Element properties.**

A rendering operation can be animated by adding animation collections to drawing context method calls, or by using animate resources. The following example shows how to push an animated opacity value into a drawing context:

```

20 FloatAnimation anim = new FloatAnimation ();
anim.From = 0.0f;
anim.Begin = Time.Immediately;
anim.To = 1.0f;
anim.Duration = new Time(1000);
anim.Fill = TimeFill.Freeze;

25 myDrawingContext.PushOpacity(0.0f, anim);

```

Elements can be animated by adding animation collections to Element properties. The following example shows how to animate the width of a button in C#:

```

30 LengthAnimation anim = new LengthAnimation ();
anim.From = new Length(50);
anim.Begin = Time.Immediately;

```



```

5  anim.To = new Length(100);
    anim.Duration = new Time(1000);
    anim.Acceleration = 0.2;
    anim.Deceleration = 0.2;
    anim.Fill = TimeFill.Freeze;

    myButton.Width = new Length(50);
    myButton.WidthAnimations = anim;

```

10 The following shows the same example in XAML:

```

15 <Button ID="myButton" Width="50">
    <Button.Width>
        <LengthAnimationCollection>
            <LengthAnimation
                From="50"
                Begin="Immediately"
                To="100"
                Duration="1"
                Acceleration="0.2"
                Deceleration="0.2"
                Fill="Freeze"
            />
        </LengthAnimationCollection>
    </Button.Width>
25 </Button>

```

Whenever an animation (or an animated resource) is used, the animation (or resource) is cloned (in a shallow, efficient way) to provide the destination with a unique, independently controllable timeline. A side-effect of this behavior is that the original animation is not part of a visual scene, and therefore it does not respond to control calls through the `ITimingControl` interface. To achieve this effect, the calling code first uses an animation and then reads the animation back. The value that is read

back can then be cached and used for timing control. The following example shows a pattern that code intending to control animations may follow:

```

private FloatAnimation myOpacityAnimation;

5 public void Initialize()
{
    FloatAnimation anim = new FloatAnimation ();

    // Set the Begin property to Indefinite because we want to start
    // this animation interactively, not automatically.
10 anim.Begin = Time.Indefinitely;
    anim.From = 1.0f; // Fully opaque
    anim.Duration = new Time(500); // half a second
    anim.To = 0.5f; // Half transparent
15 anim.AutoReverse = true;

    // Animate the opacity of some element we own
    myElement.Opacity = 1.0f;
    myElement.OpacityAnimations = anim;
20

    // ERROR: The following line doesn't have the intended result:
    // myOpacityAnimation = anim;
    //
    // This line caches an animation "template", not the actual animation
25 // that controls the opacity of the element.

    // This caches the right animation -- the one that is actually in use:
    myOpacityAnimation = (FloatAnimation)myElement.OpacityAnimations[0];
30 }

public void OnSomeEvent()
{

```

```
// Whenever we detect some event, "blink" the element
myOpacityAnimation.BeginIn(0);
}
```

5 The user will create a new class using AnimationEffect as a base class to implement an AnimationEffect. The user will also need to create a builder for their AnimationEffect.

```
public abstract class AnimationEffect : Changeable
{
10     protected AnimationEffect(AnimationEffect effect);
       public abstract AnimationEffect GetUniqueInstance(Timeline
               defaultParentTimeline);

       protected void    Invalidate();
15     protected void    InvalidatePassive();
       protected bool    IsValid { get; }
       protected Element Element { get; }

       protected virtual void AttachImpl();
20     protected virtual void DetachImpl();
       protected virtual void PreLayoutReadImpl();
       protected virtual void PreLayoutWriteImpl();
       protected virtual void PostLayoutReadImpl();
       }
25
```

Method, Property or Meaning
Event

Invalidate The user will call this when they want their AnimationEffect placed in the list of AnimationEffects to be processed during the next RenderQueueItem and they want to make sure a

271

	RenderQueueItem is scheduled. The list of invalid animations is reset at the beginning of the RenderQueueItem.
InvalidatePassive	The user will call this when they want their AnimationEffect placed in the list of AnimationEffects to be processed during the next RenderQueueItem, but they don't want to cause a RenderQueueItem to be scheduled.
IsInvalid	Returns true if the animation is currently in the list of AnimationEffects to be processed during the next RederQueueItem. It could be true because Invalidate has been called.
Element	This is the Element that the AnimationEffect is attached to. If the AnimationEffect attached to an Element, this will throw an exception. The user should not do any setup until OnAttach has been called.
AttachImpl	When an AnimationEffect is attached to an element, it is automatically cloned and the new clone is added to the collection of AnimationEffects on the element and has OnAttach called. At this point, the protected Element property on the AnimationEffect will be set. If the user has added an AnimationEffect to a collection, only the new AnimationEffect will have OnAttach called. There are no guarantees that when OnAttach is called that the Element will have its markup properties set or that the Element's children will all be in place. An AnimationEffect is cloned. Although the Element could be passed to the AnimationEffect when calling every function, it cannot be passed to event handlers for events from other elements which will be where the AnimationEffect needs it most. The AnimationEffect may set up event handlers on other elements but will still need to know that it's assigned to this Element.
DetachImpl	This will be called on an AnimationEffect when it is detached

2582

los objetos de "modelaje" 3D (Lights, MModel3DCollections, MeshPrimitive3D's, Materials, Mesh3D, Brush, y demás) son de tipo Changeable. Como se describió anteriormente, valores cambiables suministran un método Copy() que conceptualmente hace una copia profunda, pero de hecho en demanda bajo reproducción. El uso de este método Copy() da como resultado compartido óptimo de objetos mientras que aún permite a subpartes de este objeto cambiar.

"Formas" bidimensionales, como Rectangle, Poligon, Ellipse, y demás, son introducidas en el Marco de Trabajo de Presentación de Windows por medio de la clase Shape, la cual es en sí misma un UiElement. Las formas como rectángulos pueden ser puestos en un Canvas, el cual en sí mismo es un UiElement entendido para ser usado como un contenedor de formas. El siguiente es un ejemplo:

```
<Canvas Background="black" Top="100" Left="100" Height="600" Width="800">
  <Rectangle Top="600" Left="100" Width="100" Height="50"
    Fill="red" Stroke="blue" StrokeWidth="10"/>
  <Line x1="100" y1="300" x2="300" y2="100" Stroke="green" StrokeWidth="5" />
</Canvas>
```

En este caso, el Canvas es un UiElement, y el Rectangle y Line son cada uno UiElements. Cualquier cosa que sea contenida en el Canvas, en este caso, es un UiElement.

Sin embargo, considere Rutas combinadas en marcas, como el ejemplo que sigue:

```

<Canvas ...>
  <Path Fill="Red" Stroke="Blue">
    <Path.Data>
      <CircleGeometry ... />
      <RectangleGeometry ... />
      <PathGeometry ... />
    </Path.Data>
  </Path>
</Canvas>

```

En este caso, el CircleGeometry, RectangleGeometry, (es decir, que está dentro del Path.Data), en sí mismos no son UiElement, pero el Path contenedor lo es.

Un elemento de nivel "3D Shape" posee un problema en que las características 3D del Retained3DVisual no van más allá del Visual mismo. Esto es, los contenidos de dos Visuales nunca serán interpenetrados en z (por ejemplo, uno nunca tendrá una flecha visual penetrando una manzana visual). Puesto que Controls y Shapes derivan de Visual, esto también significa que no hay soporte para tener Controls o Shapes de interpenetración. Ellos son siempre proyecciones 2D de su contenido.

Para tener "objetos" 3D que interpenetren a nivel Element, y también para expresar que en marcas, un elemento ViewPort3D es suministrado para que pueda contener un modelo complejo 3D con la interpenetración necesitada. Sin embargo, este no interactúa en z con ningún otro ViewPort3D (y es llamado ViewPort3D debido a que es un viewport 2D en un espacio 3D).

ViewPort3D se deriva de la misma clase de la cual se deriva Shape (FrameworkElement o Control). Note que ViewPort3D no debería derivar de Shape, aunque Shape tiene propiedades como Fill, Stroke, y demás los cuales no tienen sentido en 3D. Así, ViewPort3D, Shape, e Image son todos hermanos. ViewPort3D tendrá un número de propiedades que pueden ser asignadas, incluyendo algunas de las propiedades que existen en Shape (Top, Left, Width, Height, Transform y Clip). Estas propiedades, mientras no sean relevantes en 3D, tienen sentido para la proyección 2D en 3D, la cual es que ViewPort3D, es un UiElement.

ViewPort3D podrá tener una Camera asociada con este (y un razonable default será suministrado) así como propiedades de ancho de escena 3D como niebla (es decir, propiedades que también están Retained3DVisual). ViewPort3D también tiene un valor de propiedad Model3DCollection (la cual es en sí misma un Model3D) que representa una colección de objetos 3D a ser representados a través de la forma. Este modelo también puede

incluir luces. Lo que una aplicación desea permitir para interactuar en z será puesto en este simple Model3DCollection para un simple ViewPort3D.

Si la cámara no es especificada, una por defecto será utilizada que suministra una proyección tal que las fronteras de ViewPort3D determinadas por Top, Left, Width, Height, sea llenada con el modelo, los planos cerca/lejos son asignados justo en frente de y justo detrás del modelo, la cámara está mirando el eje z abajo, con y arriba. Si no hay luces especificadas, una luz direccional blanca apuntando bajo el eje y será implícitamente usada.

Las marcas para el ViewPort3D son similares a la Ruta mostrada abajo, ya que es un ejemplo donde no son especificados UIElements dentro de la Ruta de marcas. El siguiente es un ejemplo de marcas ViewPort3D:

<Canvas ...>

726

<!-- 2D elements go here, and Viewport3D is a 2D element. -->

<Viewport3D optionalCamera="...">

<XFilePrimitive path="..." />

<Mesh ...>

<BrushMaterial>

<SolidColorBrush color="yellow">

</BrushMaterial>

</Mesh>

<Mesh ...>

<VisualMaterial>

<Button ... />

</VisualMaterial>

</Mesh>

<SpotLight ... />

<Model3DCollection Transform="3DTransform">

<Mesh ... />

<More models, meshes, lights, groups, etc./>

</Model3DCollection>

</Viewport3D>

<!-- More 2D elements -->

<Rectangle ... />

</Canvas>

727

237

Dentro de la etiqueta <ViewPort3D>, el miembro valorado Model3DCollection es poblado de un Model3D de especificación de marcas individual. Note que este es un mapeo uno a uno entre las etiquetas y las subclases de Model3D. Además, note que el segundo Mesh en este ejemplo tiene un material Visual que comprende un control específico de marcas (un Button en este caso). Este control puede ser completamente interactivo, y puede especificar un OnClick y similares.

Los hijos nodos directos de ViewPort3D son Model3D, porque la colección "por defecto" es la colección Model. La sintaxis de este puede cambiar generalmente a una especificación requerida de <ViewPort3D.Model> en sintaxis de propiedad compleja. Esta marca puede seguir la misma ruta que siguen marcas regulares 2D, por ejemplo, para definir un Material y usarlo en múltiples lugares. Esta funcionalidad es suministrada por el mecanismo de recurso general.

Generalmente, las pruebas ocurren a nivel UiElement o Control. Sin embargo, las pruebas deben ser limitadas en un ViewPort3D como un todo, u otra manera de probar requerirá partir modelos en múltiples ViewPort3D y probarlos. Sin embargo, tan pronto como ellos están en múltiples ViewPort3D, ellos no hacen parte del ambiente 3D. Esto es resuelto permitiendo a un manejador de evento ser especificado en miembros individuales de un Model3D en el código o marcas,

238

también como un "id". Cuando las pruebas se resuelven, el método apropiado será invocado (el enviador todavía es el ViewPort3D como un todo), y el identificador viene como parte de EventArgs. Las aplicaciones deben estar en capacidad de aislar su lógica en diferentes manejadores de eventos, o apagando el "id".

El uso de un VisualMaterial es una indicación para el sistema de que las pruebas pueden proceder en un elemento embebido. El sistema asegura que estas se hacen, de hecho, suceden. Así la operación de la prueba de transición de 2D en 3D y regresar en 2D. La siguiente es una especificación de API para ViewPort3D:

```
public class Viewport3D : UIElement // (or Control / FrameworkElement)
{
    // Stock 2D properties
    public BoxUnit Top { get; set; }
    public BoxUnit Left { get; set; }
    public BoxUnit Width { get; set; }
    public BoxUnit Height { get; set; }
    public Transform Transform { get; set; }
```

```

public Geometry Clip { get; set; }

// 3D scene-level properties
public Fog Fog { get; set; }
public Camera Camera { get; set; } // have good default

// The 3D Model itself
public Model3D Model { get; set; }
}

```

Para hacer la transparencia correctamente en 3D, las primitivas (y, en el límite, triángulos individuales) son ordenados para representar de atrás a adelante, por lo cual el z-búfer no participa. Esto es necesario para asegurar el mezclado correcto.

Un `EmbeddingHostVisual` también es suministrado. Esta es una subclase de `Visual` que no puede ser construida por partes externas, y solo puede ser construida internamente. `EmbeddingHostVisual` contiene un `Visual` que representa que el `Visual` especificado está siendo usado como un "anfitrión encajado" para alguna relación `Visual` a `Visual` diferente a padre/hijo. Los `Visuals` 2D que son embebidos en una escena 3D son otro ejemplo de esto. Los efectos son otro ejemplo de una relación entre `Visuals` que no es padre/hijo.

Una definición se dispone abajo:

```
public class EmbeddingHostVisual : Visual
{
    internal EmbeddingHostVisual(Visual hostingVisual); // can't be externally built
    public Visual HostingVisual { get; }

    // All IVisual read services will result in exceptions raised.
}
```

Debido a que la relación en un EmbeddingHostVisual "padre" es definida para no ser la relación típica padre/hijo, que significa que los servicios de lectura como extraer un Transform entre Visuals no tiene sentido porque este no es un mapeo uniforme. Estas operaciones son manejadas en el subsistema 3D en sí mismo, pero note que el mecanismo es único a Retained3DVisual.

MEDIOS

MediaData puede ser utilizado para reproducir cualquier contenido audio/video por medio del siguiente objeto, también descrito en el Apéndice adjunto:

```
public System.Windows.Media.MediaData : Changeable, IDisposable,
ITimingControl
{
    public MediaData (string source);

    //
    // MediaData properties
    //
    public MediaState State { get; }
    public Time MediaDuration { get; }
    public bool HasAudio { get; }
    public bool HasVideo { get; }
    public long Width { get; }
    public long Height { get; }
    public double Volume { get; set; }
    public bool Mute { get; set; }

    //
    // MediaData Methods
    //
    public void Play();

    //
    // Changeable
    //
    public bool IsChangeable { get; } // defaults to true
}
```

281

```

public Changeable Copy();
public void MakeUnchangeable();
public UseStatus StatusOfNextUse { get; set; } // defaults to false
public event EventHandler Changed { add; remove; }
public UIContext UIContext { get; }

//
// IDisposable interface
//
public void Dispose();

//
// ITimingControl interface
//
double Acceleration { get; set; }
bool AutoReverse { get; set; }
TimeSyncValue Begin { get; set; }
double Deceleration { get; set; }
Time Duration { get; set; }
TimeSyncValue End { get; set; }
TimeEndSync EndSync { get; set; }
TimeFill Fill { get; set; }
TimeFill FillDefault { get; set; }
Timeline ParentTimeline { get; set; }
double RepeatCount { get; set; }
Time RepeatDuration { get; set; }
TimeRestart Restart { get; set; }
TimeRestart RestartDefault { get; set; }
double Speed { get; set; }
int CurrentRepeat { get; }
Time CurrentTime { get; }
bool IsForwardProgressing { get; }

```

1283

```

bool IsPaused { get; }
bool IsReversed { get; }
double Progress { get; }
bool IsChanging { get; }
bool IsEnabled { get; }
bool IsOverridingBaseValue { get; }
bool HasChanged { get; }

void Enable();
void Enable(Timeline parentTimeline);
void Disable();

void BeginIn(Time offset);
void EndIn(Time offset);
void Pause();
void Resume();
void Seek(Time offset, TimeSeekOrigin origin);

event EventHandler Begun { add; remove; }
event EventHandler Ended { add; remove; }
event EventHandler Paused { add; remove; }
event EventHandler Repeated { add; remove; }
event EventHandler Resumed { add; remove; }
event EventHandler Reversed { add; remove; }
event EventHandler Searched { add; remove; }
}

```

El objeto `AudioData` es suministrado para datos de audio:

```

public System.Windows.Media.AudioData : MediaData
{
    public AudioData (string source);
}

```

749


```

//
// MediaData
//
public MediaState State { get; }
public Time MediaDuration { get; }
public bool HasAudio { get; }
public bool HasVideo { get; }
public long Width { get; }
public long Height { get; }
public double Volume { get; set; }
public bool Mute { get; set; }

public void Play();

public void Dispose();

public bool IsChangeable { get; } // defaults to true
public Changeable Copy();
public void MakeUnchangeable();
public UseStatus StatusOfNextUse { get; set; } // defaults to false
public event EventHandler Changed { add; remove; }
public UIContext UIContext { get; }

double Acceleration { get; set; }
bool AutoReverse { get; set; }
TimeSyncValue Begin { get; set; }
double Deceleration { get; set; }
Time Duration { get; set; }
TimeSyncValue End { get; set; }
TimeEndSync EndSync { get; set; }
TimeFill Fill { get; set; }
TimeFill FillDefault { get; set; }

```

```

Timeline ParentTimeline { get; set; }
double RepeatCount { get; set; }
Time RepeatDuration { get; set; }
TimeRestart Restart { get; set; }
TimeRestart RestartDefault { get; set; }
double Speed { get; set; }
int CurrentRepeat { get; }
Time CurrentTime { get; }
bool IsForwardProgressing { get; }
bool IsPaused { get; }
bool IsReversed { get; }
double Progress { get; }
bool IsChanging { get; }
bool IsEnabled { get; }
bool IsOverridingBaseValue { get; }
bool HasChanged { get; }

void Enable();
void Enable(Timeline parentTimeline);
void Disable();

void BeginIn(Time offset);
void EndIn(Time offset);
void Pause();
void Resume();
void Seek(Time offset, TimeSeekOrigin origin);

event EventHandler Begun { add; remove; }
event EventHandler Ended { add; remove; }
event EventHandler Paused { add; remove; }
event EventHandler Repeated { add; remove; }
event EventHandler Resumed { add; remove; }

```

286

```

event EventHandler Reversed { add; remove; }
event EventHandler Socked { add; remove; }
}

```

El objeto VideoData es suministrado para datos de video:

```

public System.Windows.Media.VideoData : MediaData
{
    public VideoData (string source);

    //
    // MediaData
    //

    public MediaState State { get; }
    public Time MediaDuration { get; }
    public bool HasAudio { get; }
    public bool HasVideo { get; }
    public long Width { get; }
    public long Height { get; }
    public double Volume { get; set; }
    public bool Mute { get; set; }

    public void Play();

    public void Dispose();

    public bool IsChangeable { get; } // defaults to true
    public Changeable Copy();
    public void MakeUnchangeable();
    public UseStatus StatusOfNextUse { get; set; } // defaults to false
    public event EventHandler Changed { add; remove; }
    public UIContext UIContext { get; }
}

```

7987

```

double Acceleration { get; set; }
bool AutoReverse { get; set; }

bool IsOverridingBaseValue { get; }
bool HasChanged { get; }

void Enable();
void Enable(Timeline parentTimeline);
void Disable();

void BeginIn(Time offset);
void EndIn(Time offset);
void Pause();
void Resume();
void Seek(Time offset, TimeSeekOrigin origin);

event EventHandler Begun { add; remove; }
event EventHandler Ended { add; remove; }
event EventHandler Paused { add; remove; }
event EventHandler Repeated { add; remove; }
event EventHandler Resumed { add; remove; }
event EventHandler Reversed { add; remove; }
event EventHandler Searched { add; remove; }
}

```

A nivel de elemento un objeto Video es también suministrado:

```

public System.Windows.Controls.Video : FrameworkElement
{
    public Video();
    //

```

```

// Video Element properties
//
public VideoData Source { get; set; }
public MediaState MediaState { get; }
public double Volume { get; set; }
public bool Mute { get; set; }
public int NativeVideoHeight { get; }
public int NativeVideoWidth { get; }
public Time MediaDuration { get; }

public TimeSyncValue Begin { get; set; }
public TimeSyncValue End { get; set; }
public Time Duration { get; set; }
public double RepeatCount { get; set; }
public Time RepeatDuration { get; set; }

public HorizontalAlignment HorizontalAlignment { get; set; }
public VerticalAlignment VerticalAlignment { get; set; }
public Stretch Stretch { get; set; }

//
// Video Element methods
//
public void BeginIn(Time offset);
public void EndIn(Time offset);
public void Pause();
public void Resume();
public void Seek(Time offset, TimeSeekOrigin origin);
}

```

A nivel de elemento un objeto Audio también es suministrado:

```
public System.Windows.Controls.Audio: FrameworkElement
{
    public Audio();

    //
    // Audio Element properties
    //
    public AudioData Source { get; set; }
    public MediaState MediaState { get; }
    public double Volume { get; set; }
    public bool Mute { get; set; }
    public Time MediaDuration { get; }

    public TimeSyncValue Begin { get; set; }
    public TimeSyncValue End { get; set; }
    public Time Duration { get; set; }
    public double RepeatCount { get; set; }
    public Time RepeatDuration { get; set; }

    //
    // Audio Element methods
    //
    public void BeginIn(Time offset);
    public void EndIn(Time offset);
    public void Pause();
    public void Resume();
    public void Seek(Time offset, TimeSeekOrigin origin);
}
```

CONCLUSIÓN

Como se puede ver de la descripción precedente detallada, se suministra una capa de integración de medios incluyendo interfaces y un modelo de objeto que suministra código de programa con la habilidad de hacer interfaz con una escena gráfica. El sistema, método y modelo de objeto son directamente para ser usados, con gran alcance, flexibilidad y extensibilidad.

Mientras que la invención es susceptible de varias modificaciones y construcciones alternativas, ciertas modalidades ilustradas al respecto son mostradas en los dibujos y son descritas arriba en detalle. Debe entenderse, sin embargo, que no hay intención de limitar la invención a las formas específicas discutidas, pero al contrario, la intención es cubrir todas las modificaciones, construcciones alternativas, y equivalentes al caer dentro del espíritu y alcance de la invención.

REIVINDICACIONES**LO QUE SE REIVINDICA ES:**

1. Un método para organizar datos de gráficos de computador para procesarlos en una salida, que comprende:

Recepción del llamado de una función por medio de una interfaz de una capa de integración de medios, el llamado de la función corresponde a datos relacionados a gráficos; y

Causando datos en una estructura de datos de escena gráfica a ser modificados basados en el llamado de la función.

2. El método de la reivindicación 1 en donde causa datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función para inicializar una nueva instancia de una clase visual.

3. El método de la reivindicación 2 además comprende, que la recepción de una llamada de función por medio de una interfaz correspondiente a una transformación asociada con el visual.

4. El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende la invocación a una función para inicializar una nueva instancia de una clase visual de dibujo.
5. El método de la reivindicación 4 comprende, la recepción de una llamada de función vía una interfaz para abrir el dibujo de una instancia visual para representar, y en respuesta, causar un contexto de dibujo a ser retornado, el contexto de dibujo suministra un mecanismo para representarlo en el dibujo visual.
6. El método de la reivindicación 1 además comprende, recepción de datos de brocha en asociación con el llamado a la función, y donde los datos causados en una estructura de datos gráfica a ser modificada comprende la invocación a una función de brocha para modificar una estructura de datos en la estructura de datos de la escena gráfica tal que cuando un marco de representado de la escena gráfica, un área podrá ser llenada con datos visibles correspondientes a los datos de la brocha.

7. El método de la reivindicación 6 en donde recibir datos de brocha comprende recibir datos correspondientes a color sólido.
8. El método de la reivindicación 6 en donde recibir datos de brocha comprende recibir datos correspondientes a una brocha de gradiente lineal y una colección de parada que comprende t menos una parada.
9. El método de la reivindicación 6 en donde recibir datos de brocha comprende recibir datos correspondientes al gradiente radial de la brocha.
10. El método de la reivindicación 6 en donde recibir datos de brocha comprende recibir datos correspondientes a una imagen.
11. El método de la reivindicación 10 además comprende, recibir el llamado de una función por medio de una interfaz que corresponde a un efecto de imagen a ser aplicada a la imagen.
12. El método de la reivindicación 1 además comprende, recibir datos de pluma en asociación con el llamado a la función, y donde causa datos en una estructura de datos de escena gráfica a ser modificada comprende

24

invocar una función pluma que define un contorno de una forma.

13.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función relacionada con la geometría para representar una elipse en una estructura de datos de escena gráfica.

14.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función relacionada con geometría para representar un rectángulo en una estructura de datos de escena gráfica.

15.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función relacionada con geometría para representar una ruta en la estructura de datos de escena gráfica.

16.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función

460

relacionada con geometría para representar una línea en la estructura de datos de escena gráfica.

17.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función relacionada con las pruebas de un visual en una estructura de datos de escena gráfica.

18.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función relacionada con la transformación de coordenadas de un visual en una estructura de datos de escena gráfica.

19.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función relacionada con el cálculo de la caja límite de un visual en una estructura de datos de escena gráfica.

20.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función por medio de una interfaz común a un objeto visual en la estructura de datos de escena gráfica.

abl

21.El método de la reivindicación 1 además comprende invocar un manejador visual para representar un árbol de al menos un objeto visual en una representación de destino.

22.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función para ubicar un objeto contenedor en una estructura de datos de escena gráfica, el objeto contenedor configurado para contener al menos un objeto visual.

23.El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función para poner los datos de una imagen en la estructura de datos de escena gráfica.

24.El método de la reivindicación 23 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función para ubicar un objeto de efecto de imagen en una estructura de datos de escena gráfica que es asociado con el dato de la imagen.

25. El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función para ubicar datos correspondientes al texto en una estructura de datos de escena gráfica.

26. El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función para suministrar un contexto de dibujo en respuesta a la función llamada.

27. El método de la reivindicación 26 en donde el llamado de una función corresponde a un visual retenido, y además comprende, el llamado posterior para tener el contexto de dibujo del visual retenido retornado a la estructura de datos de escena gráfica.

28. El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función para ubicar un visual tridimensional en una estructura de datos de escena gráfica.

29. El método de la reivindicación 28 en donde se causan datos en una estructura de datos de escena gráfica a

ser modificados comprende el mapeo de una superficie bidimensional en un visual tridimensional.

30. El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificados comprende invocar una función para ubicar un dato de animación en una estructura de datos de escena gráfica.

31. El método de la reivindicación 30 además comprende comunicación de información de línea de tiempo correspondiente a datos de animación a un motor de composición en otra capa de la capa de integración de medios.

32. El método de la reivindicación 31 donde el motor de composición interpola datos gráficos basados en la línea de tiempo para animar una salida correspondiente a un objeto en la estructura de datos de escena gráfica.

33. El método de la reivindicación 1 en donde recibir una llamada de función vía una interfaz de una capa de integración de medios comprende recibir marcas, y donde los datos causados en una estructura de datos de escena

294

gráfica a ser modificados comprende el análisis de marcas en un llamado a una interfaz de un objeto.

34. El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificada comprende invocar una función para ubicar un objeto correspondiente a datos de audio y/o video en una estructura de datos de escena gráfica.

35. El método de la reivindicación 1 en donde se causan datos en una estructura de datos de escena gráfica a ser modificada comprende cambiar un valor mutable de un objeto en la estructura de datos de escena gráfica.

36. En un ambiente de computación, un sistema comprende:

Una estructura de datos de escena gráfica de un sistema de capas para contener datos que pueden ser representados en una salida que para una salida subsecuente integrada que puede ser vista; y

un modelo de objeto incluyendo objetos y otros datos que pueden ser contenidos en la estructura de datos de escena gráfica, por lo menos algunos de los objetos del modelo objetos teniendo interfaces para invocar

295

funciones para modificar contenidos de la estructura de datos de escena gráfica.

37. El sistema de la reivindicación 26 en donde al menos una función es invocada para ubicar un árbol de objetos visuales en una estructura de datos de escena gráfica.

38. El sistema de la reivindicación 37 además comprende un manejador visual que cuando se invoca representa el árbol de objetos visuales en una representación destino.

39. El sistema de la reivindicación 37 en donde el árbol de objetos visuales es contenido en un objeto colección visual.

40. El sistema de la reivindicación 26 en donde al menos una función es invocada para ubicar un objeto visual en una estructura de datos de escena gráfica.

41. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar una brocha con un objeto visual.

33'

42. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar una geometría con un objeto visual.
43. El sistema de la reivindicación 42 en donde la geometría comprende al menos un conjunto que contiene una geometría de elipse, una geometría de rectángulo, una geometría de línea y una geometría de ruta.
44. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar una transformación con un objeto visual.
45. El sistema de la reivindicación 44 en donde la transformación comprende una transformación de rotación para cambiar el ángulo percibido de un objeto visual.
46. El sistema de la reivindicación 44 en donde la transformación comprende una transformación de escala para cambiar el tamaño percibido de un objeto visual.
47. El sistema de la reivindicación 44 en donde la transformación comprende una transformación de traslación para cambiar la posición percibida de un objeto visual.

962

48. El sistema de la reivindicación 44 en donde una transformación comprende una transformación de inclinación para cambiar la inclinación percibida de un objeto visual.
49. El sistema de la reivindicación 44 además comprende información asociada de animación con la transformación, y donde la información de animación causa transformación de datos asociados con la transformación para cambiar sobre el tiempo de tal modo que se anime la transformación del objeto visual sobre el tiempo.
50. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar un color con el objeto visual.
51. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar un dato de gradiente con el objeto visual.
52. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar un trozo de brocha con el objeto visual.

53. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar una imagen con el objeto visual.
54. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar datos tridimensionales con el objeto visual.
55. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar un dibujo que comprende primitivas de dibujo con el objeto visual.
56. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar medios de audio y/o video con el objeto visual.
57. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar un efecto de imagen con el objeto visual.
58. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar una pluma con el objeto visual, para describir como una forma es contorneada.

59. El sistema de la reivindicación 40 en donde al menos una función es invocada para obtener un contexto de dibujo asociado con el objeto visual.
60. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar datos de prueba con el objeto visual.
61. El sistema de la reivindicación 40 en donde al menos una función es invocada para asociar un rectángulo con el objeto visual.
62. El sistema de la reivindicación 61 en donde al menos una función es invocada para describir como un rectángulo fuente debe ser estirado para fijar un rectángulo destino correspondiente al objeto visual.
63. El sistema de la reivindicación 61 en donde al menos una función es invocada para describir como el contenido es posicionado verticalmente dentro de un contenedor correspondiente al objeto visual.
64. El sistema de la reivindicación 61 en donde al menos una función es invocada para describir como el

contenido es posicionado horizontalmente dentro de un contenedor correspondiente al objeto visual.

65. En un ambiente de computación, un sistema comprende:

Medios de la interfaz para recibir llamadas de función;

Medios de composición de alto nivel para integrar datos relacionados con gráficos y/o datos relacionados con medios recibidos por medio de la interfaz en una escena gráfica; y

Medios de representación para convertir la escena gráfica en una salida que pueda ser transmitida o desplegada.

66. El sistema de la reivindicación 65 en donde los medios de representación incluyen medios de composición de bajo nivel para construir un marco de visualización basado en datos recibidos de un motor de composición de alto nivel.

67. El sistema de la reivindicación 65 además comprende medios de animación, motor de composición de alto nivel que suministra datos de línea de tiempo para medios de composición de bajo nivel para interpolar la apariencia de datos visibles a través de al menos dos cuadros para animar los datos visibles sobre el tiempo.

RESUMEN

Una capa de integración de medios incluye una interfaz de programación de aplicación (API) y un modelo de objetos que permite a los desarrolladores de código de programa hacer una interfaz de manera consistente con una estructura de datos de escena gráfica con el fin de sacar gráficos. Por medio de las interfaces, el código de programa adiciona visuales hijos a otros visuales para construir una escena gráfica jerárquica, escribir listas de instrucciones tales como datos de geometría, datos de imagen, datos de animación y otros datos para la salida y puede transformar específicamente propiedades de unión y opacidad en visuales. La capa de integración de medios y el API habilitan a los programadores a lograr efectos de composición dentro de sus aplicaciones de manera directa, influenciando la unidad de procesamiento de gráficos de manera que no tenga impacto adverso en el rendimiento normal de las aplicaciones. Un sistema de múltiples niveles incluye la habilidad para combinar diferentes tipos de medios (tales como 2D, 3D, Video, Audio, texto e imagen) y animarlos suavemente y sin ribetes.

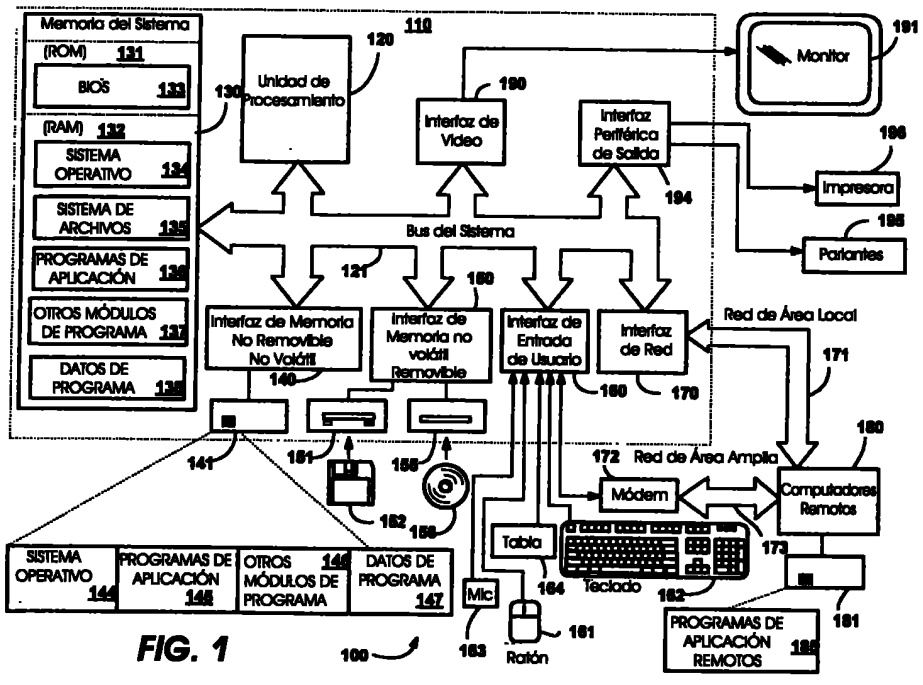


FIG. 1

4/23

2 / 29

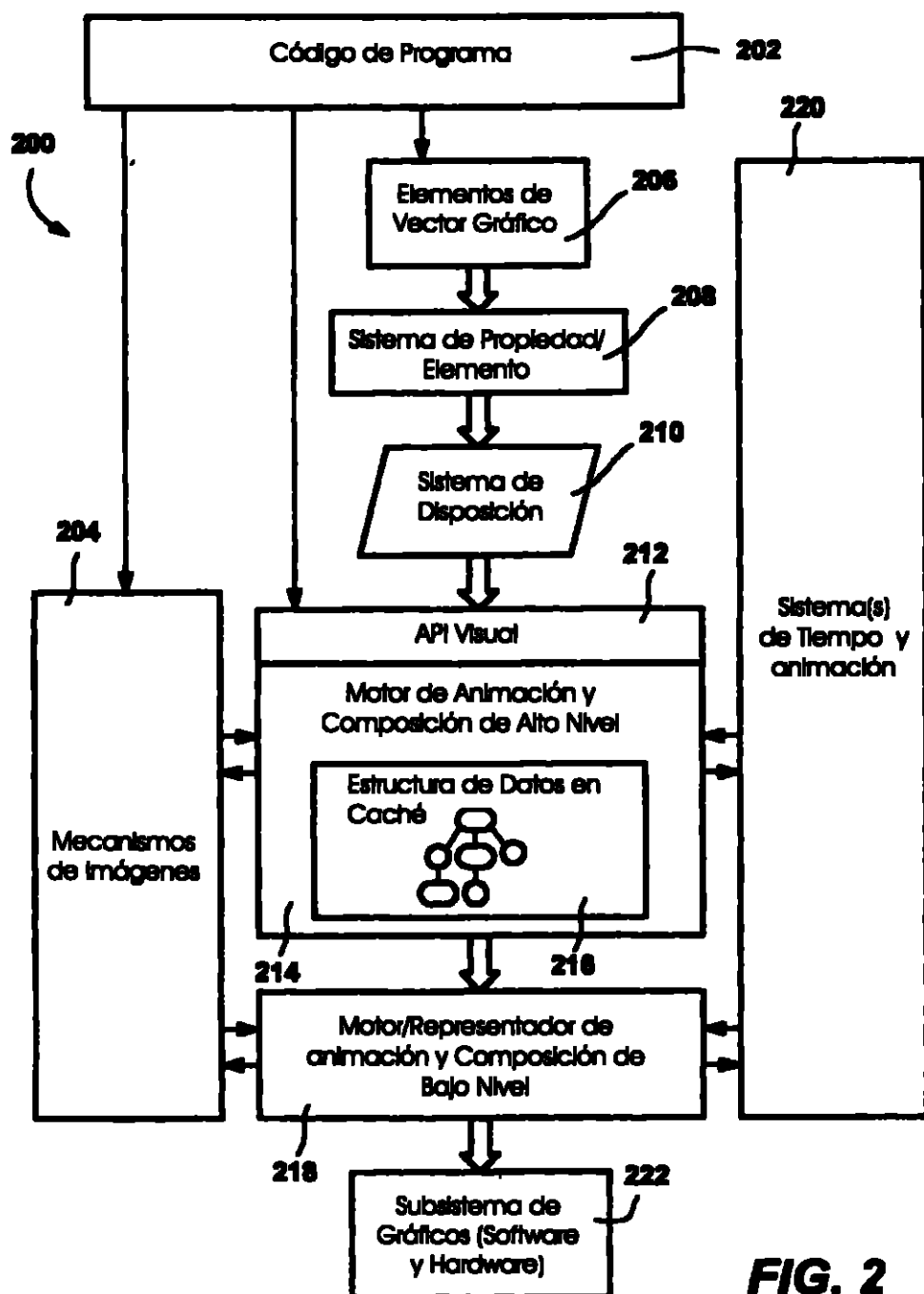


FIG. 2

9/11/14

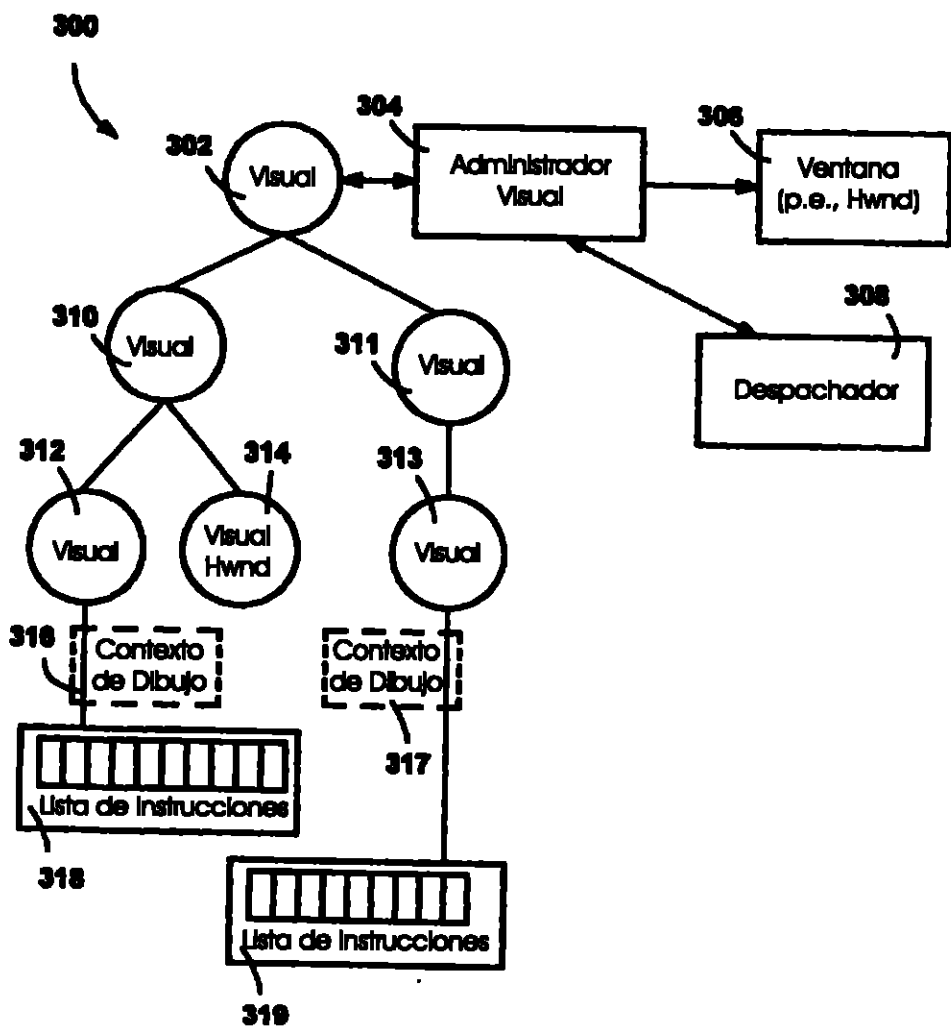
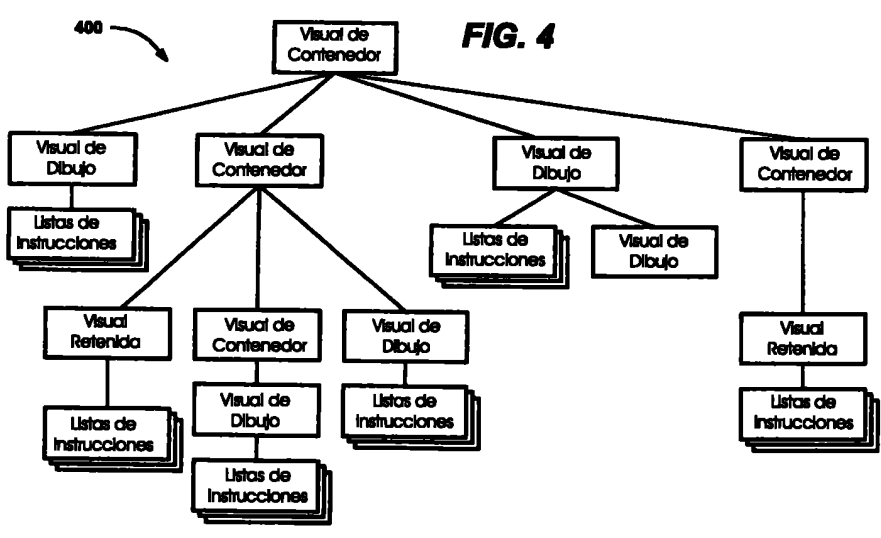


FIG. 3

278



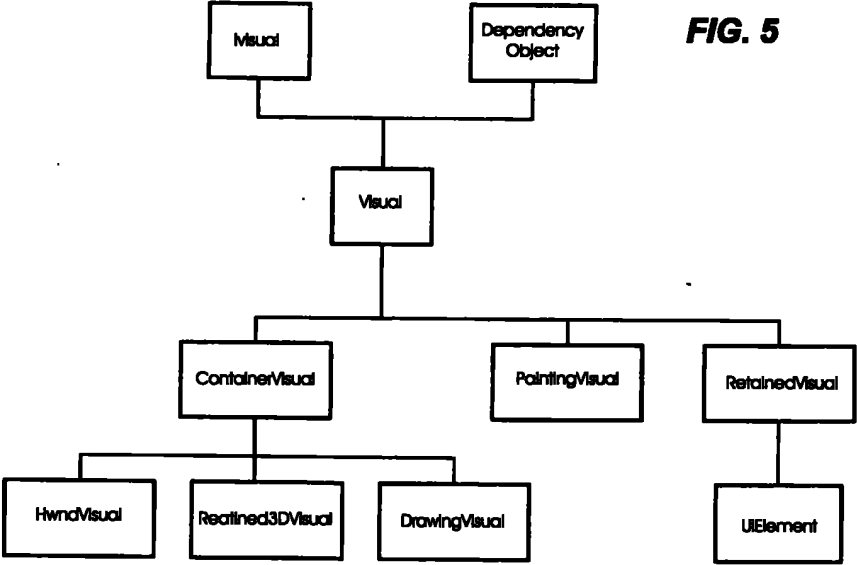


FIG. 5

4/8/8

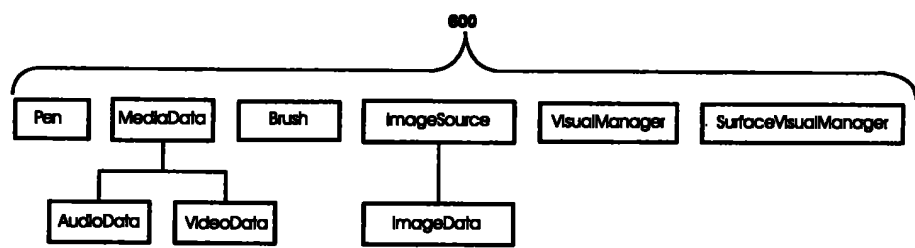


FIG. 6

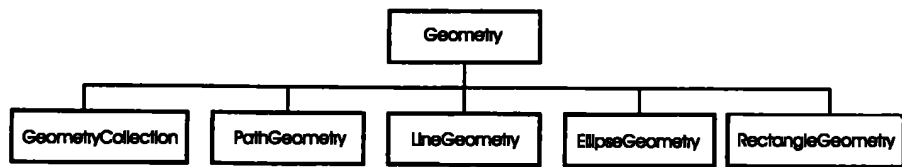


FIG. 10

276

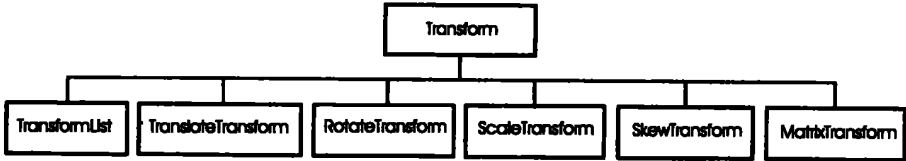


FIG. 7

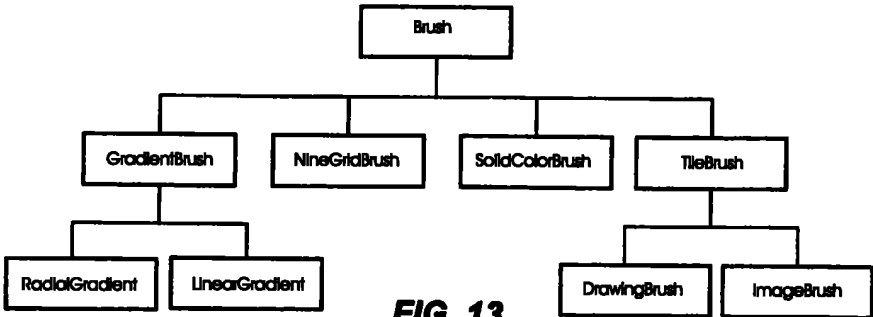


FIG. 13

9/8/21

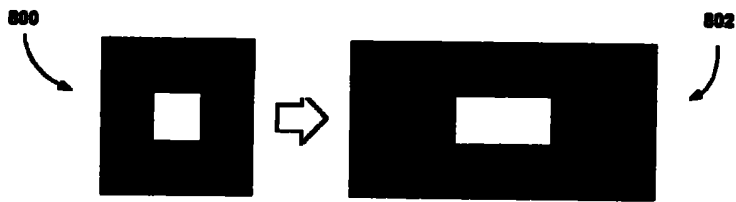


FIG. 8

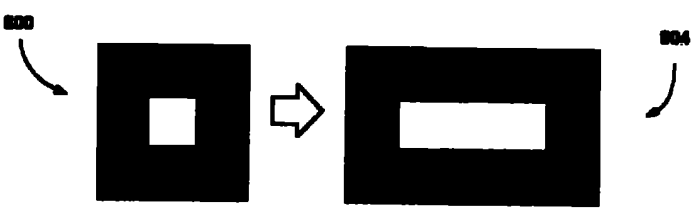


FIG. 9

9/28

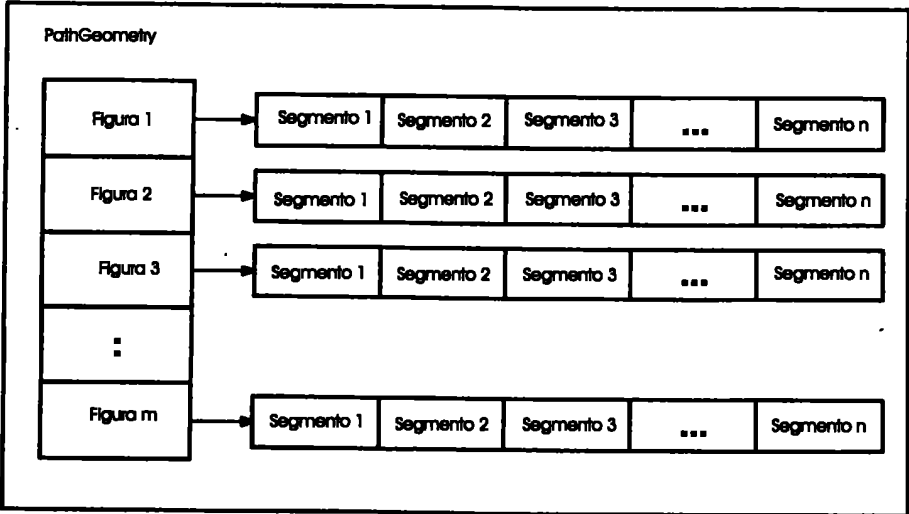


FIG. 11

659

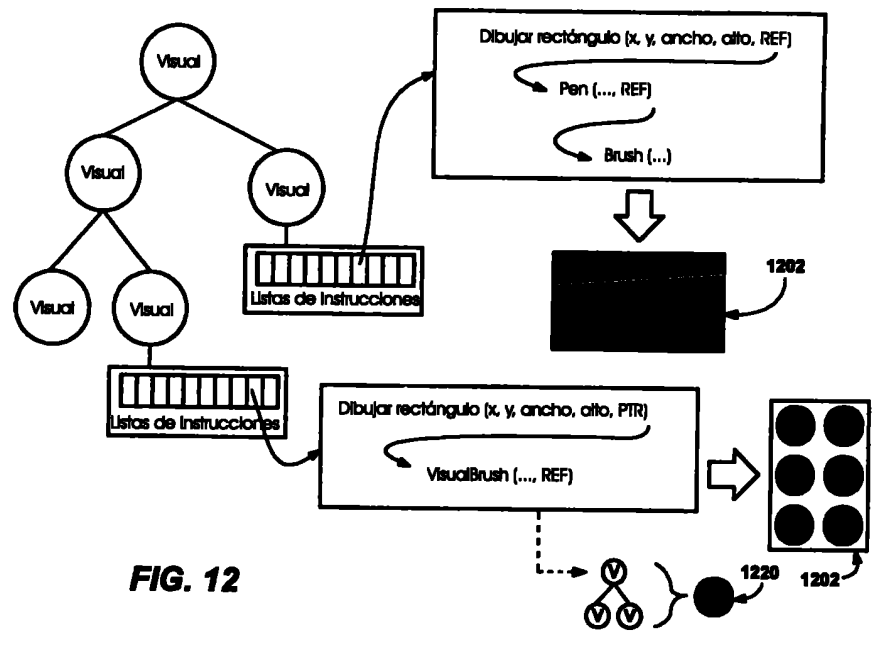


FIG. 12

9888

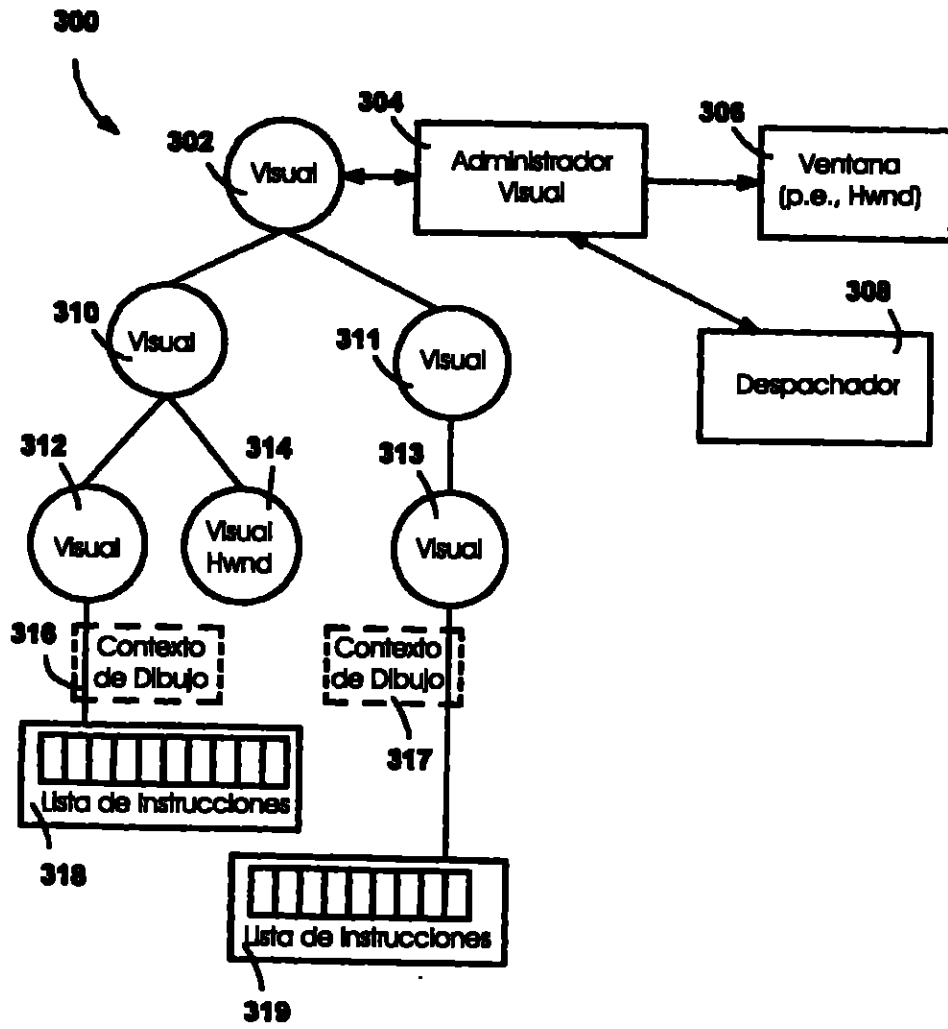
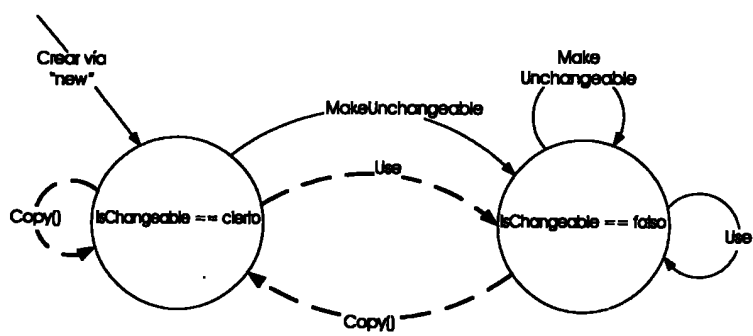


FIG. 3



12 / 29

FIG. 15

2/26

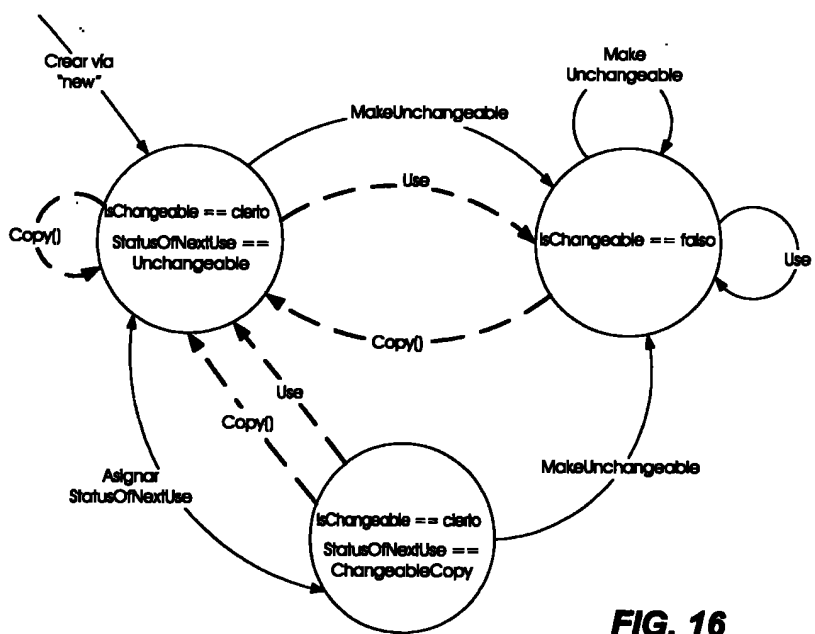


FIG. 16

3/9

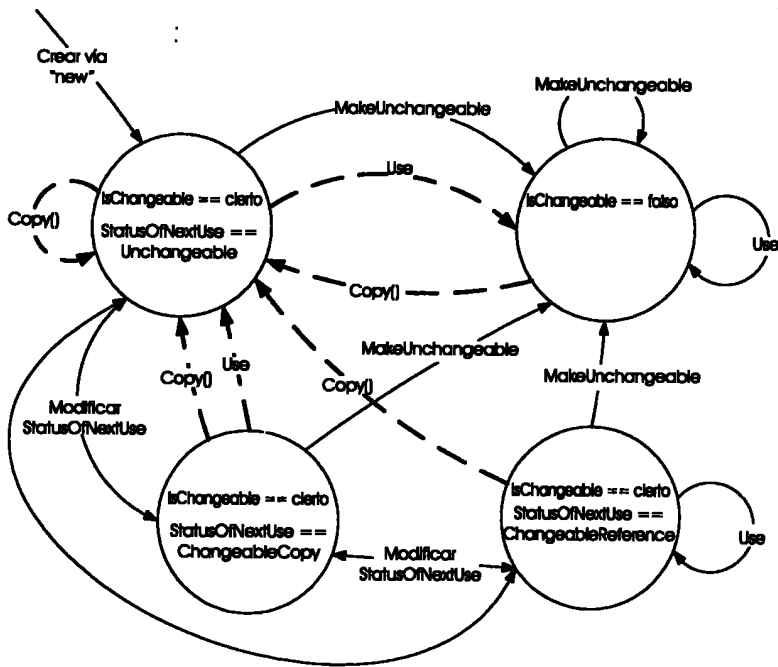


FIG. 17

989

FIG. 18

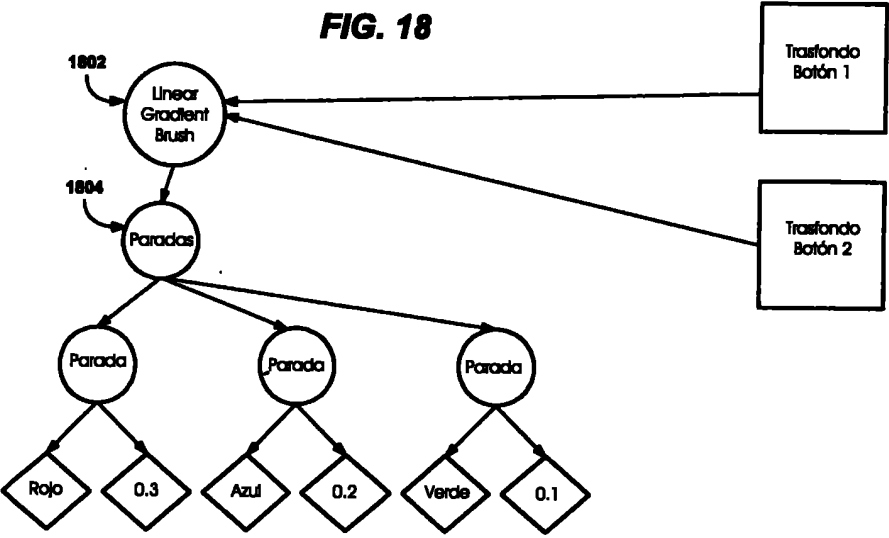
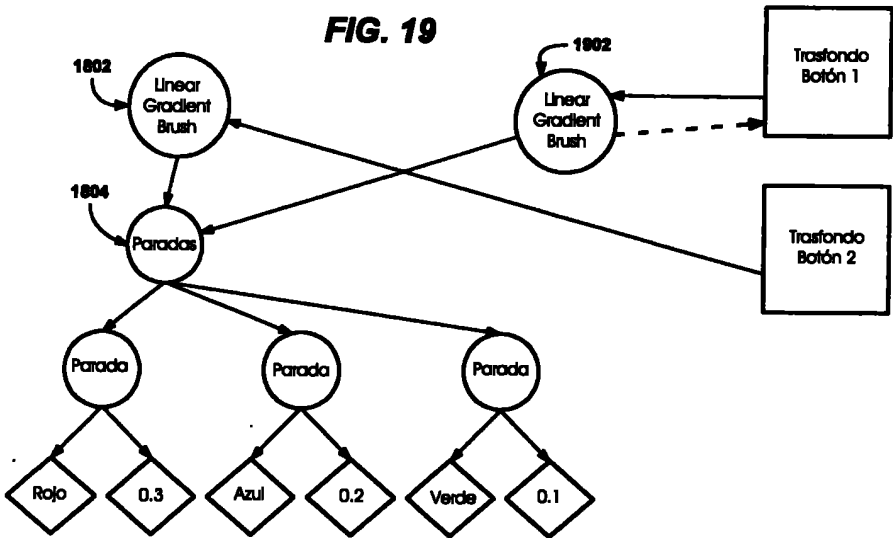


FIG. 19



288

FIG. 20

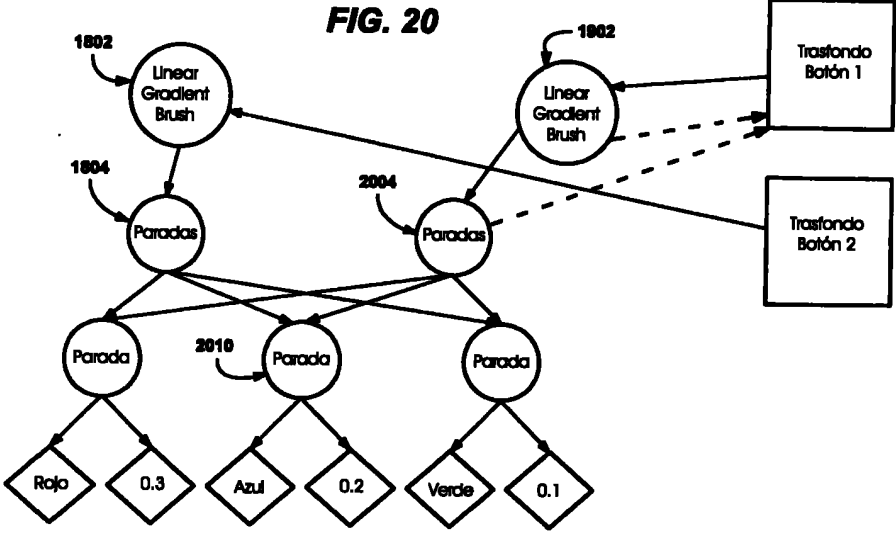
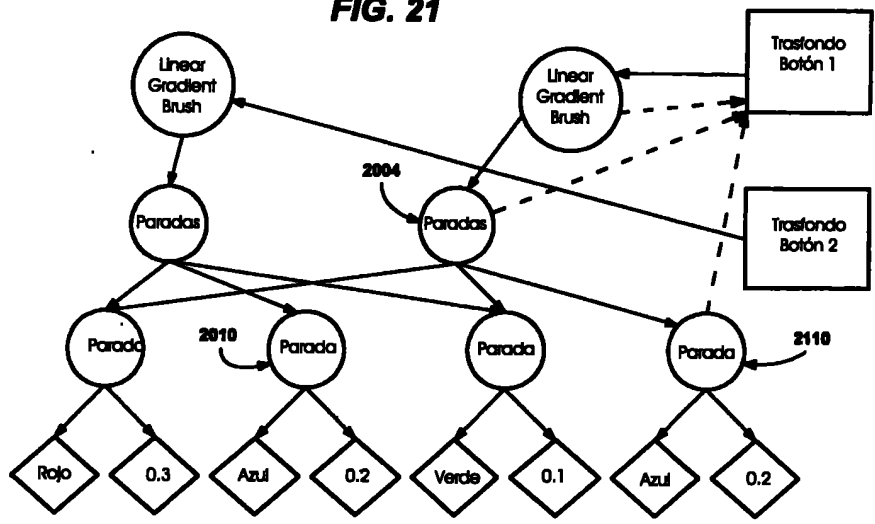
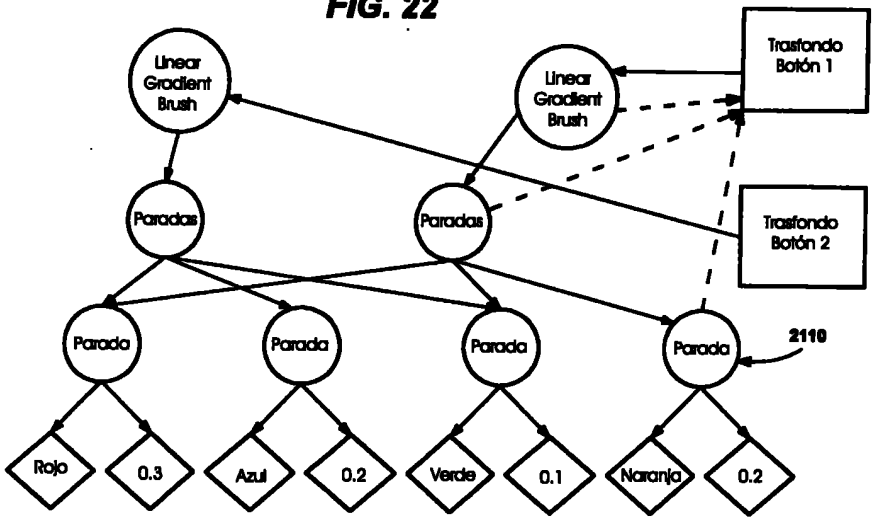


FIG. 21



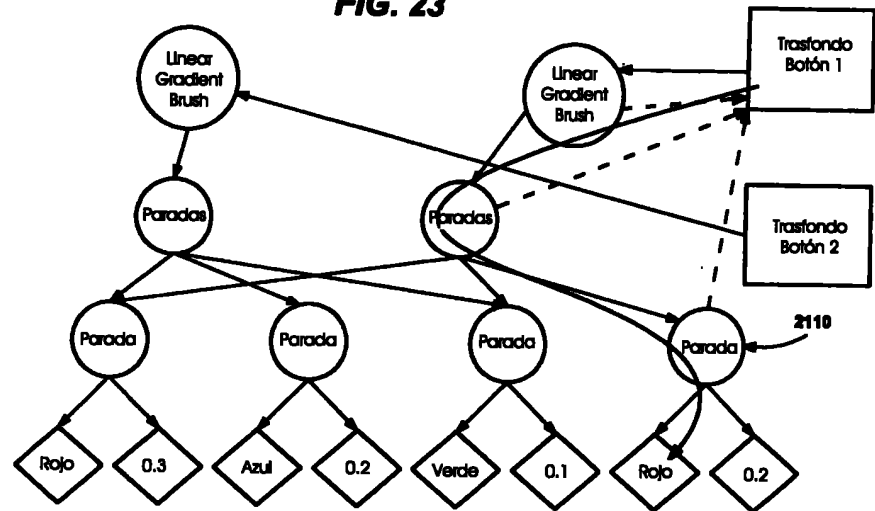
0.000

FIG. 22



604

FIG. 23



420

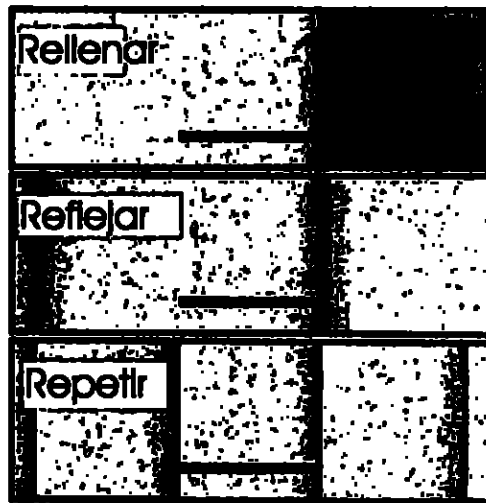


FIG. 24

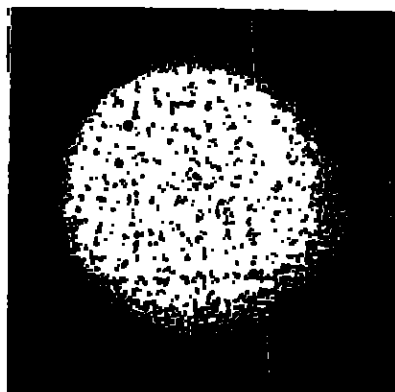


FIG. 25

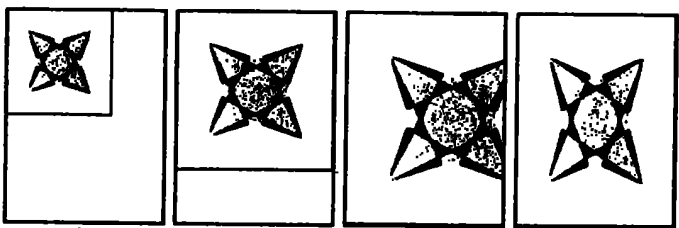


FIG. 26

47

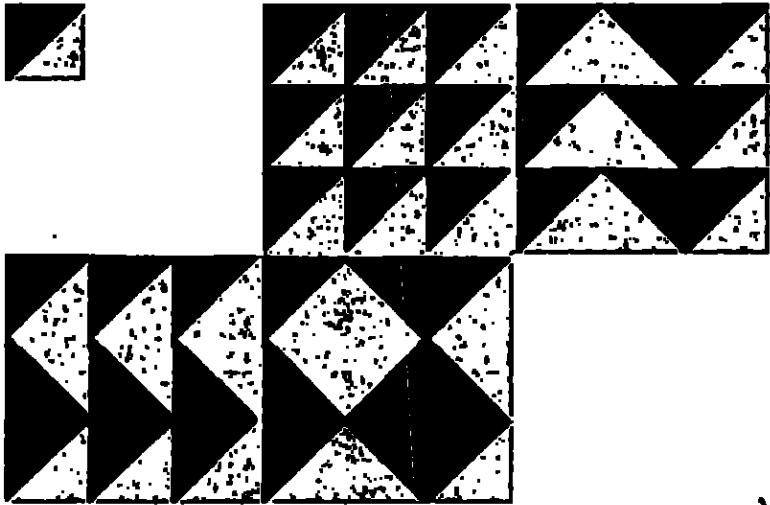


FIG. 27

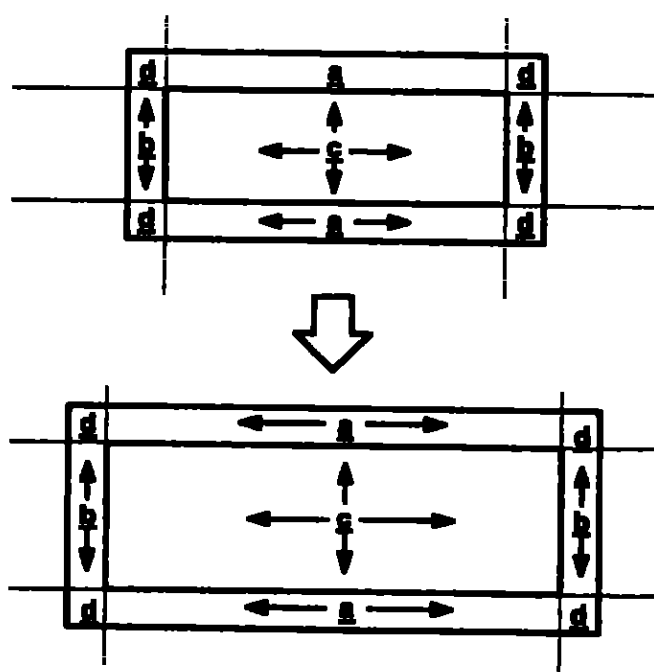


FIG. 28

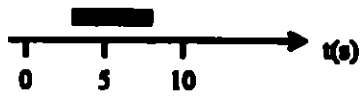


FIG. 29

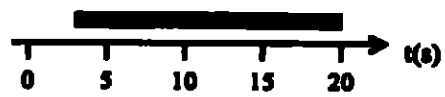


FIG. 30

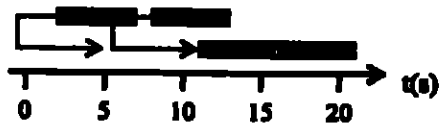


FIG. 31

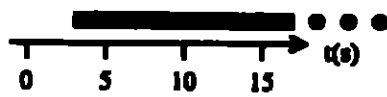


FIG. 32

332

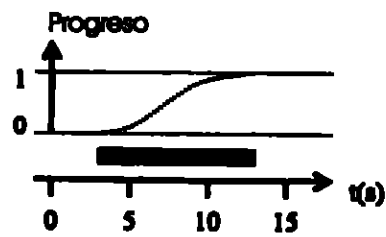


FIG. 33

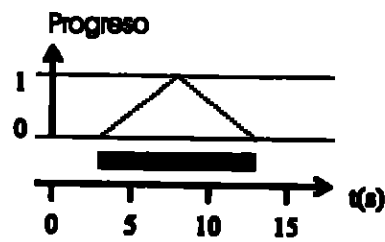


FIG. 34

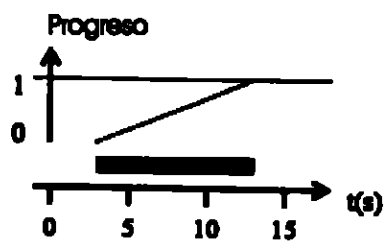
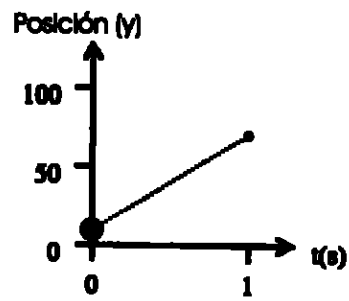
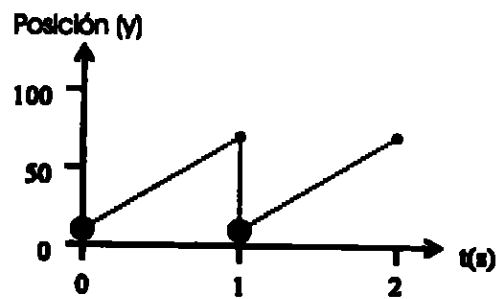
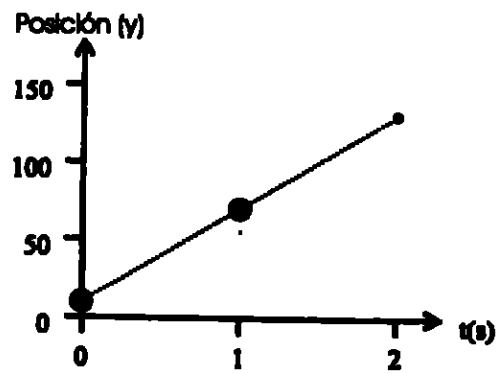


FIG. 35

446

27 / 29

**FIG. 36****FIG. 37****FIG. 38**

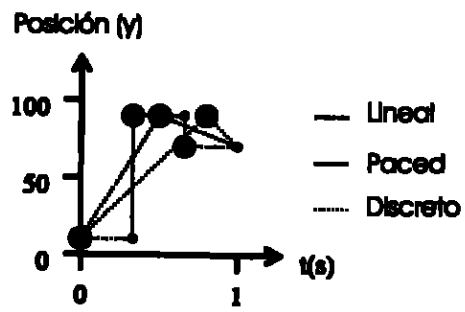


FIG. 39

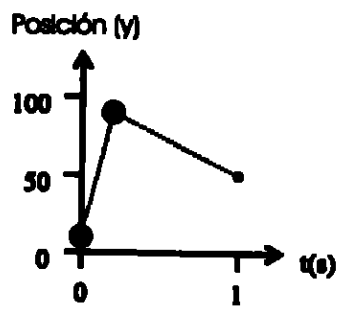


FIG. 40

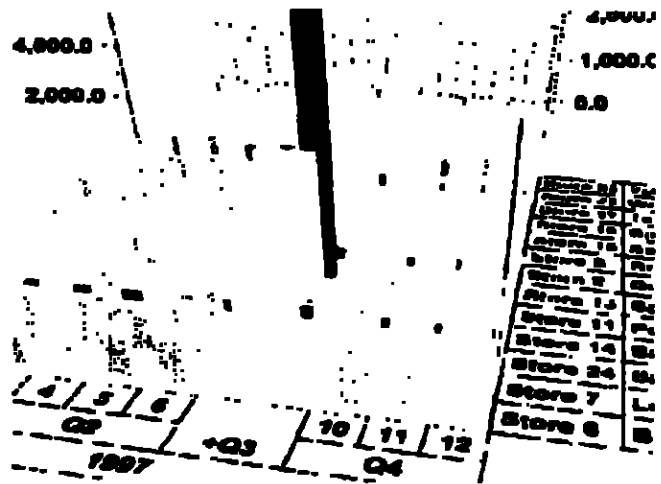


FIG. 41

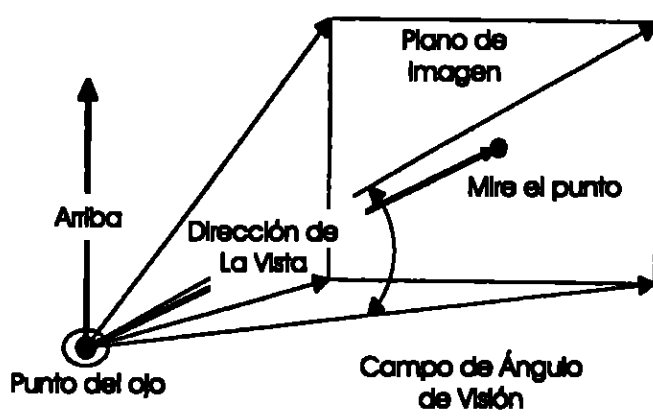


FIG. 42

1000
 336

REQUISITOS NECESARIOS PARA PRESENTACION Y AGILIZACION DEL
 TRAMITE

	USUARIO	RADICADOR
PATENTE DE INVENCION	()	()
MODELO DE UTILIDAD	()	()
- Indicación de que se solicita la concesión de una patente.	()	()
- Datos de identificación del solicitante o de la persona que se presenta la solicitud.	()	()
- Descripción de la invención.	()	()
- Dibujos de ser estos pertinentes.	()	()
- Comprobante de Pago de las tasas establecidas.	()	()

COMPLETA () INCOMPLETA ()

DISEÑO INDUSTRIAL ()

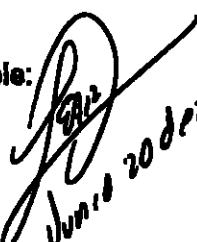
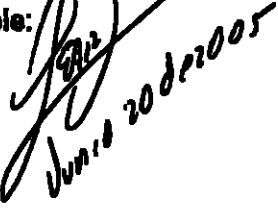
- Indicación de que se solicita el registro de Diseño Industrial	()	()
- Datos de identificación del solicitante o de la persona que presenta la solicitud.	()	()
- Representación gráfica o fotográfica del Diseño Industrial o muestra del Material que incorpora el diseño.	()	()
- Comprobante de pago de las tasas establecidas	()	()

COMPLETA () INCOMPLETA ()

ESQUEMA DE TRAZADO ()

- Indicación de que solicita el registro de un Esquema de trazado.	()	()
- Datos de identificación del solicitante o de la persona que presenta la solicitud.	()	()
- Representación gráfica del esquema de trazado	()	()
- Comprobante de pago de las tasas establecidas.	()	()

COMPLETA () INCOMPLETA ()

Firma Responsable: 
 Fecha: 

```

public void Transform(Point4D[] points);

// Since this is a vector ignores the offset parts of the matrix
public Vector3D Transform(Vector3D vector);
public void Transform(Vector3D[] vectors);

// Characteristics of the matrix
public bool IsAffine { get; } // true if  $m_{1,2,3,4} = 0$ ,  $m_{44} = 1$ .
public double Determinant { get; }
public bool HasInverse { get; }
public Matrix3D Inverse { get; } // Throws InvalidOperationException if
    !HasInverse

// Individual members
public double M11 { get; set; }
public double M12 { get; set; }
public double M13 { get; set; }
public double M14 { get; set; }
public double M21 { get; set; }
public double M22 { get; set; }
public double M23 { get; set; }
public double M24 { get; set; }
public double M31 { get; set; }
public double M32 { get; set; }
public double M33 { get; set; }
public double M34 { get; set; }
public double OffsetX { get; set; }
public double OffsetY { get; set; }
public double OffsetZ { get; set; }
public double M44 { get; set; }
};

```

338

Transform3D, como el Transform 2D, es una clase base abstracta con subclases concretas que representan tipos específicos de transformaciones 3D. Subclases específicas de Transform3D están también donde las animaciones aparecen:

matrix3D:

(coordinate comma-wsp){15} coordinate | "Identity"

En una implementación, la jerarquía general de Transform3D es asignada como viene:

Transform3D

— Transform3DCollection

— AffineTransform3D

—— TranslateTransform3D

—— ScaleTransform3D

—— RotateTransform3D

— MatrixTransform3D

Root Transform3D tiene métodos estáticos para construir clases específicas de Transform; (Note que este no expone una representación Matrix3D, como este Transform puede ser general):

338

```
public abstract class System.Windows.Media.Media3D.Transform3D : Animatable
{
    internal Transform3D();
    public new Transform3D Copy();

    // static helpers for creating common transforms
    public static MatrixTransform3D CreateMatrixTransform(Matrix3D matrix);
    public static TranslateTransform3D CreateTranslation(Vector3D translation);
    public static RotateTransform3D CreateRotation(Vector3D axis, double angle);
    public static RotateTransform3D CreateRotation(Vector3D axis, double angle,
                                                    Point3D rotationCenter);
    public static RotateTransform3D CreateRotation(Quaternion quaternion);
    public static RotateTransform3D CreateRotation(Quaternion quaternion,
```



```

        Point3D rotationCenter);

    public static ScaleTransform3D CreateScale(Vector3D scaleVector);
    public static ScaleTransform3D CreateScale(Vector3D scaleVector,
        Point3D scaleCenter);

    public static Transform3D Identity { get; }

    // Instance members
    public bool IsAffine { get; }

    public Point3D Transform(Point3D point);
    public Vector3D Transform(Vector3D vector);
    public Point4D Transform(Point4D point);

    public void Transform(Point3D[] points);
    public void Transform(Vector3D[] vectors);
    public void Transform(Point4D[] points);
}

```

Note que los métodos Transform() que toman Point3D/Vector3D pueden levantar una excepción si la transformación no es afín.

Transform3DCollection puede imitar TransformCollection, con los métodos Add* modificados en la misma forma que los métodos Create* son modificados arriba:

gal

```

public sealed class System.Windows.Media3D.Transform3DCollection :
    Transform3D, IList
{
    // follow the model of TransformCollection
}

```

AffineTransform3D es una clase base que concreta transformaciones afines 3D derivadas de (trasladar, inclinar, rotar, escalar), y esta expone métodos de lectura a Matrix3D:

```

public abstract class System.Windows.Media3D.AffineTransform3D : Transform3D
{
    internal AffineTransform3DQ; // non-extendable

    public virtual Matrix3D Value { get; }
}

```

TranslateTransform3D:

```

public sealed class System.Windows.Media3D.TranslateTransform3D :
    AffineTransform3D
{
    public TranslateTransform3DQ;
    public TranslateTransform3D(Vector3D offset);
    public TranslateTransform3D(Vector3D offset,
        Vector3DAnimationCollection offsetAnimations);
    public new TranslateTransform3D Copy();

    [Animations("OffsetAnimations")]
    public Vector3D Offset { get; set; }
    public Vector3DAnimationCollection OffsetAnimations { get; set; }

    public override Matrix3D Value { get; }
}

```

ScaleTransform3D:

```
public sealed class System.Windows.Media3D.ScaleTransform3D :
    AffineTransform3D
{
```

```
    public ScaleTransform3D();
    public ScaleTransform3D(Vector3D scaleVector);
    public ScaleTransform3D(Vector3D scaleVector, Point3D scaleCenter);
    public ScaleTransform3D(Vector3D scaleVector,
        Vector3DAnimationCollection scaleVectorAnimations,
        Point3D scaleCenter,
        Point3DAnimationCollection scaleCenterAnimations);
    public new ScaleTransform3D Copy();

    [Animations("ScaleVectorAnimations")]
    public Vector3D ScaleVector { get; set; }
    public Vector3DAnimationCollection ScaleVectorAnimations { get; set; }

    [Animations("ScaleCenterAnimations")]
    public Point3D ScaleCenter { get; set; }
    public Point3DAnimationCollection ScaleCenterAnimations { get; set; }

    public override Matrix3D Value { get; }
}
```

RotateTransform3D es más que un simple mapeo de una rotación 2D debido a la introducción del concepto de un eje rotando alrededor (y así el uso de quaternions):

```

public sealed class RotateTransform3D : AffineTransform3D
{
    public RotateTransform3D();
    public RotateTransform3D(Vector3D axis, double angle);
    public RotateTransform3D(Vector3D axis, double angle, Point3D center);

    // Quaternions supplied to RotateTransform3D methods need to be normalized,
    // otherwise an exception will be raised.
    public RotateTransform3D(Quaternion quaternion);

```

```

    public RotateTransform3D(Quaternion quaternion, Point3D center);
    public RotateTransform3D (
        Quaternion quaternion,
        QuaternionAnimationCollection quaternionAnimations,
        Point3D center,
        Point3DAnimationCollection centerAnimations);

    public new RotateTransform3D Copy();

    // Angle/Axis are just a different view on the QuaternionRotation parameter. If
    // Angle/Axis changes, QuaternionRotation will change accordingly, and vice-versa.
    public double Angle { get; set; }
    public Vector3D Axis { get; set; }

    [Animations("QuaternionRotationAnimations")]
    public Quaternion QuaternionRotation { get; set; }
    public QuaternionAnimationCollection QuaternionRotationAnimations { get; set; }

    [Animations("CenterAnimations")]
    public Point3D Center { get; set; }
    public Point3DAnimationCollection CenterAnimations { get; set; }

    public override Matrix3D Value { get; }
}

```

Note que solo la propiedad Quaternion aquí es animable. En general, las animaciones de ejes/ángulos no tienden a trabajar bien. Es mejor para animar el quaternion, y nosotros podemos extraer ejes y ángulos del valor base del quaternion. Si usted desea simplemente animar un ángulo contra un eje fijo, la manera fácil de especificar esto es construir dos quaternions representando esas posiciones, y animarla entre ellos.

MatrixTransform3D puede construir un Transform3D directamente de un Matrix3D:

```
public sealed class System.Windows.Media.MatrixTransform3D : Transform3D
{
    public MatrixTransform3D();
    public MatrixTransform3D(Matrix3D matrix);

    public new MatrixTransform3D Copy();

    public Matrix3D Value { get; set; }
}
```

Cuando una propiedad tipo Transform3D es especificada en marcas, el sistema de propiedades utiliza el convertidor tipo Transform para convertir la representación de cadena en el apropiado objeto derivado Transform. Las propiedades animadas

no son descritas utilizando esta sintaxis, sin embargo la sintaxis de propiedades complejas puede ser usada para descripciones de animaciones.

La sintaxis es modelada fuera del Transform 2D, donde "<>" representa un parámetro opcional:

- **matrix(m00 m01 m02 m03 m11 ... m33)**
- **translate(tx ty tz)**
- **scale(sx <sy> <sz> <cx> <cy> <cz>)**
 - If <sy> or <sz> is not specified it is assumed to be a uniform scale.
 - If <cx> <cy> <cz> are specified, then they all need to be specified, and <sx> <sy> do as well. They are used for the scaling center. If they're not, center is assumed to be 0,0,0.
- **rotate(ax ay az angle <cx> <cy> <cz>)**
 - ax,ay,az specifies axis of rotation
 - angle is the angle through that axis
 - If cx, cy, cz is not specified it's assumed to be 0,0,0.

El siguiente es un ejemplo de gramática:

transform-list:

wsp* transforms? wsp*

transforms:

transform

| transform comma-wsp+ transforms

transform:

matrix

| translate

| scale

| rotate

| skewX

| skewY

matrix:

"matrix" wsp* "(" wsp*

number comma-wsp

number comma-wsp

... 13 more times ...

number wsp* ")"

translate:

"translate" wsp* "(" wsp* number (comma-wsp number comma-wsp number)?
wsp* ")"

scale:

"scale" wsp* "(" wsp* number (comma-wsp number comma-wsp number
(comma-wsp number comma-wsp number comma-wsp number)?
)? wsp* ")"

rotate:

```
"rotate" wsp* "(" wsp* number wsp* number wsp* number wsp* number
  ( comma-wsp number comma-wsp number comma-wsp number )? wsp* ")"
```

Retained3DVisual se deriva de Visual, y al tomar sus propiedades, incluyendo opacidad, Clip Geométrico 2D, Modo Combinación 2D, Pruebas del API, consulta de Límites 2D y Participación en el árbol Visual. Note que de la opacidad, recortar, modo combinar y límites aplican a la proyección 2D de la escena 3D.

```
public class System.Windows.Media.Visuals.Retained3DVisual : Visual
{
    public Retained3DVisual();
    public Retained3DVisual(UIContext Context);

    // Modeling-oriented semantics. Default value is an empty Model3DCollection
    public Model3DCollection Models { get; set; }

    // Ambient properties

    // Camera -- there's no default, it's an error not to provide one.
    public Camera Camera { get; set; }

    // ViewPort establishes where the projection maps to in 2D. Default is 0,0,1,1
    [Animation("ViewPortAnimations")]
    public Rect ViewPort { get; set; }
    public RectAnimationCollection ViewPortAnimations { get; set; }

    public Fog Fog { get; set; }
}
```


La caja ViewPort establece donde la proyección determinada por la combinación Camera/Models es mapeada en el espacio de coordenadas local 2D. Note que no hay método Bounds3D en Retained3DVisual. Este es disponible como Models.Bounds.

El Drawing3DContext es muy paralelo al DrawingContext 2D, y es accesible desde el Model3DCollection de un Retained3DVisual por medio de RenderOpen/RenderAppend. Esto se parece a un contexto de despliegue de modo inmediato, aún si bien este retiene instrucciones internamente.

```

public class System.Windows.Media3D.Drawing3DContext : IDisposable
{
    internal Drawing3DContext(); // can't be publicly constructed

    // Rendering
    public void DrawMesh(Mesh3D mesh, Material material, object hitTestToken);

    // These are for drawing imported primitives like .x files
    public void DrawImportedPrimitive(ImportedPrimitive3DSource primitiveSource,
                                      object hitTestToken);
    public void DrawImportedPrimitive(ImportedPrimitive3DSource primitiveSource,
                                      Material overridingMaterial,
                                      object hitTestToken);

    public void DrawVisual(Visual visual, Point3D centerPosition,
                          object hitTestToken);

    public void DrawModel(Model3D model);

    public void AddLight(Light light);

    // Stack manipulation
    public void PushTransform(Transform3D transform);
    public void Pop();

    public void Close(); // Also invoked by Dispose();
}

```

Detalles específicos en la semántica de las operaciones Drawing3DContext son descritas abajo con referencia al Modeling API, para el cual el Drawing3DContext es una

conveniencia. Por ejemplo, `DrawImportedPrimitive` (`ImportedPrimitive3DSource` `primitiveSource`, `objectHitTestToken`) crea un `ImportedPrimitive3D`, y lo agrega al acumulador actual `Model3D` (el cual en cambio es manipulado por los métodos `Push/Pop` en el contexto).

`DrawModel()` es otro punto de cruce entre el mundo "contexto" y el mundo "modelo", permitiendo a `Model3D` ser "dibujado" en un contexto. Este no tiene "retroalimentación" del `Drawing3dContext`, porque este tiene el `Model3DCollection` respaldándolo, y esta colección puede ser enumerada como sea necesario.

El API de modelado es el API público y protegido para estas clases, (no muestran miembros heredados), en donde `Model3D` es el modelo abstracto donde cualquiera puede construir de él:

```
public abstract class Model3D : Animatable
{
    public Transform3D Transform { get; set; } // defaults to Identity
    public ShadingMode ShadingMode { get; set; }
    public object HitTestToken { get; set; }

    public Rect3D Bounds3D { get; } // Bounds for this model

    // singleton "empty" model.
    public static Model3D EmptyModel3D { get; }
}
```

Model3dCollection es donde una combinación de modelos es construida, y tratada como una unidad, opcionalmente transformando o aplicando otros atributos a él:

```
public sealed class Model3DCollection : Model3D, IList
{
    public Model3DCollection(params Model3D[] models);
    public Model3DCollection(ICollection collection);
```

```
// Drawing3DContext semantics
public Drawing3DContext RenderOpen();
public Drawing3DContext RenderAppend();

// IList interface (bunch of stuff I'm not going into here, plus...)
public int Add(Model3D model);
public Model3D this[int index] { get; set; }
public void Insert(...);
// etc...
}
```

Note que el Model3DCollection también tiene RenderOpen/Append, los cuales retornan un Drawing3DContext. El uso de este contexto modifica en sí mismo el ModelCollection. La diferencia entre RenderOpen() y Render Append() es que RenderOpen() limpia la colección primero. Además note que en esta implementación, solo un Drawing3DContext puede ser abierto al tiempo en un

Model3DCollection, y cuando este es abierto, las aplicaciones no pueden acceder directamente (para leer o escribir) el contenido de Model3DCollection.

Las luces son Model3D. Estas incluyen Ambiente, Posicional, Direccional y luces de Punto. Las luces tiene una propiedad adicional que hace parte de una jerarquía de modelaje, y son así sometidas al espacio de coordenada de transformaciones. Los colores suministrados en luces son ambientes, difusos y especulares. Note también que no hay luces que son jerárquicamente alcanzadas o alcanzadas a un volumen particular. Una jerarquía apropiada de luz es asignada abajo:

Model3D

- Light (abstract)
- AmbientLight (concrete)
- DirectionalLight (concrete)
- PointLight (concrete)
- SpotLight (concrete)

La clase de luz base es una clase abstracta:

```

public abstract class Light : Model3D
{
    internal Light(); // only allow public construction -- no 3rd party lights

    [Animation("AmbientColorAnimations")]
    public Color AmbientColor { get; set; }
    public ColorAnimationCollection AmbientColorAnimations { get; set; }

    [Animation("DiffuseColorAnimations")]
    public Color DiffuseColor { get; set; }
    public ColorAnimationCollection DiffuseColorAnimations { get; set; }

    [Animation("SpecularColorAnimations")]
    public Color SpecularColor { get; set; }
    public ColorAnimationCollection SpecularColorAnimations { get; set; }
}

```

Las luces ambiente iluminan los objetos uniformemente, independientemente de su forma:

```

public sealed class AmbientLight : Light
{
    public AmbientLight(Color ambientColor);
}

```

Las luces direccionales no tienen posición en el espacio y proyectan su luz a lo largo de una dirección particular, especificada con el vector que la define:

SA

```
public sealed class DirectionalLight : Light
{
    public DirectionalLight(Color diffuseColor, Vector3D direction); // common usage

    [Animation("DirectionAnimations")]
    public Vector3D Direction { get; set; }
```

```
public Vector3DAnimationCollection DirectionAnimations { get; set; }
}
```

La dirección necesitada no es normalizada, pero esta necesita tener magnitud diferente de cero.

Las luces posiccionales tienen una posición en el espacio y proyectan su luz en todas las direcciones. La disminución de la luz es controlada por las propiedades atenuación y rango:

acs

355

[strong name inheritance demand so 3rd parties can't derive... we can't seal, since
SpotLight derives from this ...]

public class PointLight : Light

{

public PointLight(Color diffuseColor, Point3D position); // common usage

[Animation("PositionAnimations")]

public Point3D Position { get; set; }

public Point3DAnimationCollection PositionAnimations { get; set; }

// Range of the light, beyond which it has no effect. This is specified

// in local coordinates.

[Animation("RangeAnimations")]

public double Range { get; set; }

public DoubleAnimationCollection RangeAnimations { get; set; }

// constant, linear, and quadratic attenuation factors defines how the light

// attenuates between its position and the value of Range.

[Animation("ConstantAttenuationAnimations")]

public double ConstantAttenuation { get; set; }

public DoubleAnimationCollection ConstantAttenuationAnimations { get; set; }

[Animation("LinearAttenuationAnimations")]

public double LinearAttenuation { get; set; }

public DoubleAnimationCollection LinearAttenuationAnimations { get; set; }

[Animation("QuadraticAttenuationAnimations")]

public double QuadraticAttenuation { get; set; }

public DoubleAnimationCollection QuadraticAttenuationAnimations { get; set; }

}

206


```

public sealed class SpotLight : PointLight
{
    public SpotLight(Color color,
        Point3D position,
        Vector3D direction,
        double outerConeAngle,
        double innerConeAngle);

    [Animation("DirectionAnimations")]
    public Vector3D Direction { get; set; }
    public Vector3DAnimationCollection DirectionAnimations { get; set; }

    [Animation("OuterConeAngleAnimations")]
    public double OuterConeAngle { get; set; }
    public DoubleAnimationCollection OuterConeAngleAnimations { get; set; }

    [Animation("InnerConeAngleAnimations")]
    public double InnerConeAngle { get; set; }
    public DoubleAnimationCollection InnerConeAngleAnimations { get; set; }
}

```

Note que PointLight necesita tener una fuerte demanda de herencia en éste; para habilitar a SpotLight que derive de ésta, pero no terceras partes.

El SpotLight se deriva de PointLight como este tiene una posición, rango, y atenuación, pero también adiciona una

dirección y parámetros para controlar el "cono" de luz. Para controlar el "cono", `outerConeAngle` (más allá del cual no hay iluminación), y `innerConeAngle` (dentro del cual la iluminación es completa) necesitan ser especificadas. La iluminación entre la parte exterior del cono interno y el exterior del cono disminuye linealmente. (Note que esta es una disminución "angular" entre el límite del cono interno y el cono externo, y una disminución en distancia, relativa a la posición de la luz, y es afectada por atenuación y rango).

```
public sealed class SpotLight : PointLight
{
    public SpotLight(Color color,
                    Point3D position,
                    Vector3D direction,
                    double outerConeAngle,
                    double innerConeAngle);

    [Animation("DirectionAnimations")]
    public Vector3D Direction { get; set; }
    public Vector3DAnimationCollection DirectionAnimations { get; set; }

    [Animation("OuterConeAngleAnimations")]
    public double OuterConeAngle { get; set; }
    public DoubleAnimationCollection OuterConeAngleAnimations { get; set; }
```

```

[Animation("InnerConeAngleAnimations")]
public double InnerConeAngle { get; set; }
public DoubleAnimationCollection InnerConeAngleAnimations { get; set; }
}

```

Note que, como cualquier cosa en el API MIL, los ángulos son especificados en grados. Las primitivas 3D son nodos hoja que resultan en la representación en el árbol. Clases concretas vienen en mallas especificadas explícitamente, así como primitivas importadas (archivos .x). El constructor es interno; (la extensibilidad es disponible por medio de MeshPrimitive3D).

```

public abstract class Primitive3D : Model3D
{
    internal Primitive3D(object hitTestToken);
}

```

MeshPrimitive3D es para modelar con una malla y un material:

```

public sealed class MeshPrimitive3D : Primitive3D
{
    public MeshPrimitive3D(Mesh3D mesh, Material material, object hitTestToken);
    public Mesh3D Mesh { get; set; }

    public Material Material { get; set; }
}

```

Note que MeshPrimitive3D es una geometría de hoja, y que contiene pero no es en sí misma, un Mesh. Esto significa que un Mesh puede ser compartido entre múltiples MeshPrimitive3D, con diferentes materiales, sujetos a diferentes pruebas, sin replicar los datos de la malla.

ImportedPrimitive3D representa una primitiva adquirida externamente (potencialmente con material y animación) traída y convertida en una forma interna apropiada. Esto es tratado como un modelo rígido. El ejemplo canónico de esto es un Archivo .X, y hay una subclase de ImportedPrimitive3DSource que explícitamente importa Archivos X.

```
public sealed class ImportedPrimitive3D : Primitive3D
{
    public ImportedPrimitive3D(ImportedPrimitive3DSource primitive,
                               object hitTestToken);
    public ImportedPrimitive3DSource PrimitiveSource { get; set; }

    // Allow overriding the imported material(s) if there was any. If not specified,
    // this is null, and the built in material is used.
    public Material OverridingMaterial { get; set; }
}
```

Aunque archivos .x son esperados para ser comúnmente incluidos en las escenas, un formato simple TypeConverter para expresar esto es soportado, tal como:

```
<ImportedPrimitive3D xfile="myFile.x" />
```

El VisualModel3D toma cualquier Visual (2D, por definición), y lo coloca en la escena. Cuando es representado, este puede ser alineado en la pantalla, y su tamaño no será afectado, pero este tendrá un plano z particular de la cámara. El Visual permanecerá interactivo.

```
public sealed class VisualModel3D : Model3D
{
    public VisualModel3D(Visual visual, Point3 centerPoint, object hitTestToken);

    public Visual Visual { get; set; }
    public Point3D CenterPoint { get; set; }
}
```

Para representar un VisualModel3D primero se transforma en CenterPoint en coordenadas del mundo. Este luego representa el Visual en una memoria de puntos en una pantalla de manera alineada con el z del CenterPoint transformado siendo el centro donde el visual es ubicado. Bajo el movimiento de la cámara, el VisualModel3D ocupará la misma cantidad de estado

real de pantalla, y podrá ser llevado al frente, y no ser afectado por las luces, o algo parecido. El punto fijado durante este movimiento de cámara del visual relativo al resto de la escena será el centro del visual, aunque la ubicación suceda basada en el punto.

El Visual suministrado es completamente interactivo, y es efectivamente "emparentado" al Retained3DVisual encerrándolo, lo cual significa que un Visual dado solo puede ser usado una vez en cualquier VisualModel3D, justo como un Visual puede tener un solo padre. Note que este es uno de los dos mecanismos para embeber interactivamente Visuals en 3D. El otro es usar un VisualMaterial en una primitiva.

DrawMesh y DrawVisual toman un "objeto" hitTestToken. Este podrá ser retornado por IVisual.HitTest() cuando un Visual3D toma un acierto. Este es luego usado para deshabilitar ambigüedad que obtenga en el acierto en 3D. Note que "camino de pizar" no son necesarios aquí, ya que cada DrawMesh, igual si este toma el mismo mesh, puede ser suministrado por un diferente "hitTestToken".

En una implementación, HitTesting es implementado por medio de la utilidad Direct3D RayIntersec, la cual suministra distancia para acertar, coordenadas u,v del punto de acierto, permite las pruebas de acierto más allá del objeto de acierto

inicial, y así sucesivamente. Note también que los resultados de las pruebas de acierto tienen suficiente información para dar las coordenadas u,v de la primitiva que fue acertada, para permitir traslación en una consulta de prueba de acierto en cualquier textura que fue cubierta acá. Además, para VisualModel3D, así como el uso del VisualMaterial, la prueba de acierto procederá dentro del visual 2D en el punto de mapeo correcto. Estos resultados son permitidos por el contexto de pruebas de acierto a venir en el ambiente externo 2D, continuar a través del ambiente 3D y coger de nuevo un ambiente anidado 2D, potencialmente para siempre. Un resultado final de esto es que cuando se suministra este paso a lo largo, interacción de usuario en vivo, los controles de textura mapeada Avalon 2D operan apropiadamente.

La presente invención soporta una noción general de importar primitivas 3D de fuentes externas. Una implementación de esto en vl será de un formato de archivo .x.

```
public abstract sealed class ImportedPrimitive3DSource
{
    // Don't allow external derivation
    internal ImportedPrimitive3DSource();
}
```

Archivos .X llegan al sistema como un XFile3DSource, y son usados para construir un ImportedPrimitive3D. Este es separado fuera en un tipo de datos separado tal que este pueda ser usado y compartido en múltiples ImportedPrimitive3D. Las conversiones y procesamiento apropiados del archivo .x puede suceder en la construcción de éste, o puede ser aplazado. Note que los archivos .x son importados como unidades, y nada dentro de éste es direccionable separadamente.

```
public sealed class XFile3DSource : ImportedPrimitive3DSource
{
    // .X file data is specified as a stream, allowing it to come from
    // arbitrary sources (files, resources, memory, URLs).
    public XFile3DSource(istream xFileStream);
}
```

La primitiva Mesh3D es una primitiva triángulo directa (permitiendo especificación indexada y no indexada) que puede ser construida programáticamente. Note que este soporte que es esperado para ser el uso más común de la primitiva (posición nombrada, normal, color, y e información de textura, con el último tercio opcional). La malla también permite la selección de si este será desplegado como triángulos, líneas o puntos. Este también soporta las tres

topologías para interpretar índices, lista de triángulo nombrada, tira de triángulo, y el abanico.

Para formatos vértice y otras construcciones primitivas que no están soportadas directamente por Mesh3D, un archivo .x puede ser construido e importado.

```
public sealed class System.Windows.Media3D.Mesh3D : Animatable
{
    public Mesh3D();

    // Vertex data. Normals, Colors, and TextureCoordinates are all optional.
    public Point3DCollection Positions { get; set; }
    public Vector3DCollection Normals { get; set; } // assumed to be normalized
    public ColorCollection Colors { get; set; }
    public ColorCollection SpecularColors { get; set; }
    public PointCollection TextureCoordinates { get; set; }
}
```

```
// Topology data. If null, treat as non-indexed primitive
public IntegerCollection TriangleIndices { get; set; }

// Primitive type - default = TriangleList
public MeshPrimitiveType MeshPrimitiveType { get; set; }
}
```

MeshPrimitiveType es definido como:

```
public enum System.Windows.Media3D.MeshPrimitiveType
{
    TriangleList,
    TriangleStrip,
    TriangleFan,
    LineList,
    LineStrip,
    PointList
}
```

La información por vértice en Mesh3D es dividida en Positions, Normals, Colors, y TextureCoordinates. De estos, solo Positions es requerido. Si cualquiera de los demás es suministrado, ellos necesitan tener exactamente la misma longitud que la colección Positions, de otra manera una excepción será levantada.

Los Normals, si son suministrados, son asumidos como normalizados. Note que el sistema no trata de calcular normales basadas en topología/proximidad; en cambio, cuando las normales son deseadas, ellas necesitan ser suministradas.

La colección TriangleIndices tiene miembros que se indexan en el dato vértice para determinar la información por vértice para los triángulos que componen la malla. Esta colección es interpretada basada sobre la asignación de MeshPrimitiveType. Para TriangleList, cada tres elementos en la colección

TriangleIndices definen un nuevo triángulo. Para TriangleFan, los índices 0,1,2 determinan el primer triángulo, luego cada subsecuente índice, i , determina un nuevo triángulo dado por los vértices 0, i , $i-1$. Para TriangleStrip, los índices 0,1,2 determinan el primer triángulo, y cada subsecuente índice i determina un nuevo triángulo dado por los vértices $i-2$, $i-1$, e i . LineList, LineStrip, y PointList tienen interpretaciones similares, pero la representación es en término de líneas y puntos en lugar de triángulos.

Si TriangleIndices es nulo, entonces el Mesh es implementado como una primitiva no indexada, la cual es equivalente a TriangleIndices manteniendo valores 0, 1, ..., $n-2$, $n-1$ para una colección Positions de longitud n .

Sobre la construcción del Mesh, la implementación crea la estructura óptima D3D que representa esta malla. En este punto, la actual estructura de datos Collection puede ser lanzada lejos por la implementación de Mesh para evitar duplicación de datos. Subsecuente re-lectura de la malla es accedida a través de otro mecanismo (recorriendo la jerarquía del modelo Retained3DVisual por ejemplo) posibilitarán la reconstrucción de datos de la información D3D que es mantenida, en lugar de retener los datos originales.

El mesh deriva de Changeable (a través de Animatable), y así puede ser modificado. La implementación necesitará bloquear conjuntos de datos de vértices e índices, y propagar estos cambios a las estructuras de datos D3D. No hay soporte explícito para animaciones declarativas de datos de índices o vértices en una malla. En otras palabras, Point3DAnimationCollections no son vistas aquí, por ejemplo. Esto es consistente por ejemplo con las rutas de polilínea 2D.

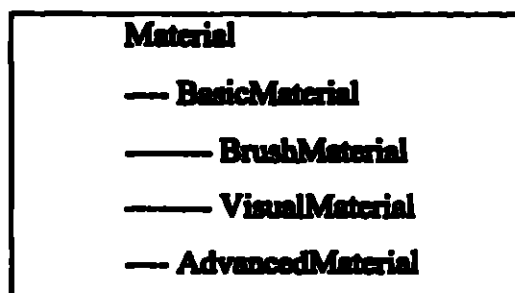
Como otros tipos, la sintaxis de propiedad compleja XAML puede ser usada para especificar las colecciones que definen Mesh3D. Sin embargo, eso puede ser engorroso y difuso, y por eso TypeConverters son suministrados para hacer especificaciones más concisas.

Cada colección definida en la malla puede tomar una sola cadena de números para ser analizada y utilizada para crear la colección. Por ejemplo, un Mesh que representa una franja de triángulo indexada con solo posiciones y colores puede ser especificada como:

```
<Mesh3D
  meshPrimitiveType="TriangleStrip"
  positions="1,2,3, 4,5,6, 7,8,9, 10,11,12, 13,14,15, 16,17,18"
  colors="red blue green cyan magenta yellow"
  triangleIndices="1,3,4,1,2,3,4,5,6,1,2,4,2"
/>
```

En efecto, cualquiera de estos puede ser representado más complejamente en la propiedad de sintaxis compleja.

Los métodos que construyen primitivas 3D toman un Material para definir su apariencia. Material es una clase base abstracta con tres subclases concretas: BrushMaterial, VisualMaterial, y AdvanceMaterial. BrushMaterial y VisualMaterial son ambas subclases de otra clase abstracta llamada BasicMaterial. Así:



El método de BrushMaterial toma un solo Brush y puede ser usado para un amplio rango de efectos, incluyendo alcanzar transparencia (punto por punto o escalar), tener una transformación de textura (aún una animación), usando texturas de video, mipmaps autogenerador implícitos, y así sucesivamente. Específicamente, para dar textura a colores sólidos, imágenes, gradientes, u otro Visual igual, un SolidColorBrush, ImageBrush, GradientBrush, o VisualBrush deben ser usados para crear el BrushMaterial.

El método `VisualMaterial` es diseñado para construir un material fuera de un `Visual`. Este material será interactivo en el sentido de que las entradas podrán pasar al `Visual` desde el mundo 3D en el cual está embebido. Una diferencia entre este y `BrushMaterial` con un `VisualBrush` es que `BrushMaterial` no es interactivo.

La clase `AdvancedMaterial`, aún cuando es considerablemente más complejo que simple usar un `BrushMaterial` o `VisualMaterial`, suministra flexibilidad adicional.

```
public abstract class Material : Animatable
{
    internal Material(); // don't allow external subclassing

    public new Material Copy(); // shadows Changeable.Copy()
```

```
public static Material Empty { get; } // singleton material
}

public abstract class BasicMaterial : Material
{
    internal BasicMaterial(); // don't allow external subclassing

    public new BasicMaterial Copy(); // shadows Changeable.Copy()
}
```

Los Materials ofrecen flexibilidad y "economía de concepto" por ser basadas en un Brush. Por ejemplo, aquí no es necesaria una jerarquía de Texturas separada reflejando cosas como texturas de video, texturas gradiente, o parecidos, ya que todos estos son especificables como Brushes. Los Brushes ya encapsulan valores de opacidad escalar y máscara alfa, de manera que ambos vienen disponibles para ser texturizados. Los Brushes ya tienen Transform 2D asociado con ellos los cuales, en el caso del texturizado, será interpretado como una transformación de textura para transformar coordenadas uv en la malla para mapearla en la textura. Los Brushes están bien ubicados para mantener una pila de sombreadores procedurales tales como un sombreador beta de madera. Esto sería luego usable en 2D como un relleno o pluma, y en 3D como una textura. No se requiere ningún soporte específico al API en el espacio 3D para los sombreadores procedurales.

Note que los sombreadores de píxel o vértice pueden ser suministrados como "pila" de sombreadores, muchos de los cuales serán parametrizados. La manera en que estos sean accesibles en el API es que para los sombreadores que tengan sentido en el mundo 2D, ellos serán expuestos como subclases concretas de Brush, con su parametrización expresada a través de constructores en la clase, o como propiedades en la clase. Esto puede ser aplicado a objetos 2D. Los sombreadores que solo tienen sentido en 3D serán expuestos como subclases

concretas de Material (quizás como subclases de BasicMaterial), donde ellos pueden también ser parametrizados a través de su constructor. Esta exposición permitirá a los sombreadores ser aplicados a mallas 3D (y 2D cuando sea apropiado).

Como se describió anteriormente, BrushMaterial encapsula un Brush. Un BrushMaterial aplicado a una primitiva 3D es tratado como una textura. Las texturas pueden ser mapeadas de manera directa, esto es, las coordenadas 2D u,v en la primitiva serán mapeadas directamente en las coordenadas correspondientes x,y en el Textura, modificada por la transformación de textura. Note que, como otro 2D, el sistema de coordenadas de textura va desde (0,0) hasta el extremo superior izquierdo con el y positivo disminuyendo.

Un VisualBrush usado para el Brush no acepta entrada, pero podrá actualizarse de acuerdo a cualquier animación en éste, o a cualquier cambio estructural que suceda en éste. Para usar un Visual como un Material y aún recibir entrada, es usado el VisualMaterial, como se describe aquí.


```
public sealed class BrushMaterial : BasicMaterial
{
    public BrushMaterial(Brush brush);

    public new BrushMaterial Copy(); // shadows Material.Copy()

    public Brush Brush { get; set; }

    // Additional texturing specific knobs.
}
```

Como se describió anteriormente, VisualMaterial encapsula un Visual interactivo. Esto difiere de un BrushMaterial usado como un Visual en el que el Visual permanece activo en su forma texturizada. Note que puesto que el mapeo es basado en una grilla u,v asociada con una primitiva, hay situaciones donde las entradas de usuario no serán completamente fieles con la imagen 2D. Por ejemplo, considere un escenario de captura de ratón donde un control 2D captura el ratón (para implementar el comportamiento de la barra de desplazamiento, por ejemplo). En estos escenarios, al ratón se le permite desviarse de la barra de desplazamiento, pero el sistema está en capacidad de simbolizarlo, por ejemplo, la posición y del ratón está con respecto a la barra de desplazamiento y opera apropiadamente. En una situación donde el control es texturizado en una primitiva con una grilla u,v, y el ratón se pierde fuera de la primitiva, entonces en general, no hay

manera de hacer esta determinación. Hay ciertas cosas que pueden ser realizadas mientras el escritor contrarresta esto, pero en general, esta será una limitación del mapeo en el espacio 2D. Note que el Visual es luego, en efecto, emparentado de alguna forma con la raíz del Retained3DVisual. En una implementación, es ilegal utilizar un solo UiElement en más de un Material, o usar un VisualMaterial en más de un lugar, debido al hecho de que el marco de trabajo está basado solamente en controles emparentados.

```
public sealed class VisualMaterial : BasicMaterial
{
    public VisualMaterial(Visual visual);

    public new VisualMaterial Copy(); // shadows Changeable.Copy()

    public Visual Visual { get; set; }

    -(need to add viewport/viewbox stuff for positioning...)

    // Additional texturing specific knobs.
}
```

BrushMaterials/VisualMaterials y BumpMaps son usados para definir AdvancedMaterials:

```

public class AdvancedMaterial : Material
{
    public AdvancedMaterial();

    // Add common constructors.

    public new AdvancedMaterial Copy(); // shadows Changeable.Copy()

    public BasicMaterial DiffuseTexture { get; set; }
    public BasicMaterial SpecularTexture { get; set; }
    public BasicMaterial AmbientTexture { get; set; }
    public BasicMaterial EmissiveTexture { get; set; }
    ..

    [ Animations("SpecularPowerAnimations") ]
    public double SpecularPower { get; set; }

```

```

    public DoubleAnimationCollection SpecularPowerAnimations { get; set; }

    public BumpMap DiffuseBumpMap { get; set; }
    public BumpMap ReflectionBumpMap { get; set; }
    public BumpMap RefractionBumpMap { get; set; }

    public BrushMaterial ReflectionEnvironmentMap { get; set; }
    public BrushMaterial RefractionEnvironmentMap { get; set; }
}

```

Note que los environmentMaps son texturas que son esperadas a ser en un formato particular para habilitar mapeo de cubo. Específicamente, las seis caras del mapa del cubo necesitarán ser representadas en las secciones del Brush asociado con el

qds

Textura (probablemente algunas cosas como una grilla 3x2 en el Brush). Note que las propiedades Ambient, Diffuse, y Specular toman un BasicMaterial, y no un Material general ya que a ellos no se les permite ser especificados en sí mismos como AdvancedMaterials. Además, note que los mapas de ambiente son BrushMaterials exclusivamente, ya que la interactividad que suministra VisualMaterial no tiene sentido para mapas de ambiente.

Mapas de protuberancia son grillas que, como texturas, son mapeadas en primitivas 3D vía coordenadas de textura en las primitivas. Sin embargo, los datos interpolados son interpretados como perturbaciones a la normalidad de la superficie, resultando en una apariencia "desigual" de la primitiva. Para alcanzar esto, los mapas de protuberancias transforman información tal como perturbaciones normales, y potencialmente otra información. Ellos no transportan color o información de transparencia. Debido a esto, es inapropiado utilizar un Brush como un mapa de protuberancias. Por lo tanto, una nueva clase BumpMap es suministrada, como un ImageSource de un formato de punto particular.

Material suministra un simple TypeConverter que permite la especificación en cadena de un Brush para automáticamente ser promovido en un BrushMaterial:

Material:

... delegate to Brush type converter ...

Esto permite especificaciones tales como:

<MeshPrimitive3D ... Material="yellow" />

<MeshPrimitive3D ... Material="LinearGradient blue green" />

<MeshPrimitive3D ... Material="HorizontalGradient orange purple" />

<MeshPrimitive3D ... Material="Resource(myImageResource)" />

Also compound property notation,

<MeshPrimitive3D>

<MeshPrimitive3D.Material>

<LinearGradientBrush>

.....

Los parámetros de ambiente del modelo, los cuales no son imbebibles en niveles arbitrarios en la jerarquía geométrica, pueden ser suministrados. Por ejemplo, niebla puede adicionarse a una escena asignando la propiedad Fog en el Retained3DVisual, por ejemplo, el Fog disponible es "niebla de punto .fog" es representado como una clase abstracta, y la jerarquía se muestra a continuación:

```

public abstract class Fog : Animatable
{
    // only constructable internally
    internal Fog(Color color);
    public new Fog Copy(); // hides Changeable.Copy()

    [Animation("ColorAnimations")]
    public Color Color { get; set; }
    public ColorAnimationCollection ColorAnimations { get; set; }

    // singleton representation of "no fog"
    public static Fog NoFog { get; }
}

```

```

public sealed class LinearFog : Fog
{
    public LinearFog(Color color, double fogStart, double fogEnd);

    [Animation("FogStartAnimations")]
    public double FogStart { get; set; }
    public DoubleAnimationCollection FogStartAnimations { get; set; }

    [Animation("FogEndAnimations")]
    public double FogEnd { get; set; }
    public DoubleAnimationCollection FogEndAnimations { get; set; }
}

public sealed class ExponentialFog : Fog
{
    public ExponentialFog(Color color, double fogDensity, bool squaredExponent);

    [Animation("FogDensityAnimations")]
    public double FogDensity { get; set; }
    public DoubleAnimationCollection FogDensityAnimations { get; set; }

    public bool SquaredExponent { get; set; }
}

```

Los rangos de fogDensity de 0-1, y es una representación normalizada de la densidad de la niebla. El fogStart y fogEnd comprenden profundidades z especificadas en un espacio de dispositivo [0,1] y representa donde comienza y donde termina la niebla.

La Camera es un mecanismo por el cual un modelo 3D es proyectado en un visual 2D. La Camera en sí misma es un tipo abstracto, con dos subtipos concretos - ProjectionCamera y MatrixCamera. ProjectionCamera toma parámetros bien entendidos tales como Position, LookAtPoint, y FieldOfView para construir el Camera. MatrixCamera toma un Matrix3D utilizado para definir la transformación World-to-Device.

```
public abstract class Camera : Animatable
```

```
{
    // Only allow to be built internally.
    internal Camera();

    public new Camera Copy(); // hides Changeable.Copy()
}
```

En un Reatined3DVisual, un Camera es utilizado para suministrar una vista en un MModel3D, y la proyección resultante es mapeada en un ViewPort 2D establecido en el

Retained3DVisual. Note que la caja de límite 2D del Retained3DVisual será simplemente la caja 3D proyectada del modelo 3D, envuelta con este convexo, casco alineado en eje, unido al clip establecido en el visual.

La ProjectionCamera es el medio por el cual la cámara es construida de los parámetros tales como Position, LookAtPoint, y FieldOfView. Este encapsula proyección de perspectiva y proyección ortográfica. La FIG. 42 suministra una buena indicación de aspectos relevantes de una ProjectionCamera mostrando la visión y posición, (donde FieldOfView debe estar en dirección horizontal).

```
public class ProjectionCamera : Camera
{
    // Common constructors
    public ProjectionCamera();
    public ProjectionCamera(Point3D position,
                           Vector3D lookDirection,
                           Vector3D Up,
                           double fieldOfView);

    public new ProjectionCamera Copy(); // hides Changeable.Copy()

    // Camera data
    public bool IsOrthographic { get; set; } // default == false: Perspective

    [Animations("NearPlaneDistanceAnimations")]
```



```

public double NearPlaneDistance { get; set; } // default = 0
public DoubleAnimationCollection NearPlaneDistanceAnimations { get; set; }

[Animations("FarPlaneDistanceAnimations")]
public double FarPlaneDistance { get; set; } // default = infinity
public DoubleAnimationCollection FarPlaneDistanceAnimations { get; set; }

[Animations("FieldOfViewAnimations")]
public double FieldOfView { get; set; }
public DoubleAnimationCollection FieldOfViewAnimations { get; set; }

[Animations("PositionAnimations")]
public Point3D Position { get; set; }
public Point3DAnimationCollection PositionAnimations { get; set; }

[Animations("LookDirectionAnimations")]
public Vector3D LookDirection { get; set; }
public Vector3DAnimationCollection LookDirectionAnimations { get; set; }

[Animations("UpAnimations")]
public Vector3D Up { get; set; }
public Vector3DAnimationCollection UpAnimations { get; set; }
}

```

Note que el FieldOfView representa el campo horizontal de la vista, y es especificado en grados (como otros ángulos MIL). Las PlaneDistances Near y Far representan distancias en el mundo de coordenadas 3D desde la posición de la cámara a lo largo del vector LookDirection. El NearPlaneDistance por

defecto es cero y el `FarPlaneDistance` por defecto es infinito.

Sobre la proyección actual, si las `NearPlaneDistance` y `FarPlaneDistance` son aún cero e infinito, respectivamente, entonces el modelo es examinado y este volumen limitado es proyectado de acuerdo a la proyección de la cámara. El volumen limitado resultante es luego examinado de modo que la distancia cercana al plano es asignada al plano de volumen limitado perpendicular al `LookDirection` más cercano a la posición de la cámara. Esto es esencialmente igual para el plano lejano, pero usando el plano más lejano. Esto resulta en un uso óptimo de la resolución del z-búfer mientras que aún se está mostrando el modelo entero. Si `IsOrthographic` es verdadero, entonces una proyección ortográfica es utilizada, y el `FieldOfView` es ignorado.

Note que el "plano de proyección" definido por los parámetros de `ProjectionCamera` e luego mapeado en el rectángulo `ViewPort` en el `Retained3DVisual`, y que representa la transición final del espacio 3D al espacio 2D.

La subclase `MatrixCamera` de `Camera` suministra un medio para la especificación directa de una matriz como una transformación de proyección. Esto es útil para aplicaciones

que tienen su propio mecanismo de cálculo de matriz de proyección.

```
public class MatrixCamera : Camera
{
    // Common constructors
    public MatrixCamera();
    public MatrixCamera(Matrix3D matrix);

    public new MatrixCamera Copy(); // hides Changeable.Copy()

    // Camera data
    public Matrix3D Matrix { get; set; } // default = identity
}
```

Note que los valores MinimumZ y MaximumZ controlan directamente el rango de z-búfer. Ellos son considerados después de la aplicación de la Matriz como una matriz de proyección.

La proyección resultante es mapeada en el rectángulo ViewPort en el Retained3DVisual, y que representa la transición final entre el espacio 3D y el espacio 2D.

Las siguientes son marcas más complejas que muestran la especificación de una jerarquía Model3D completa en XAML. Note que alguna de estas sintaxis puede cambiar.

El siguiente ejemplo simplemente crea un Model con dos archivos .x importados y una transformación de rotación (alrededor del eje z en 45 grados), y una sola luz de punto blanco situada encima en 0,1,0.

```

<Model3DCollection>

  <!-- Model children go as children here -->
  <PointLight position="0,1,0" diffuseColor="white" />

  <ImportedPrimitive3D xfile="myFile.x" />

  <Model3DCollection transform="rotate(0, 0, 1, 45), scale(2)" >
    <ImportedPrimitive3D xfile="mySecondFile.x" />
  </Model3DCollection>

</Model3DCollection>

```

Esta marca puede estar en un archivo, un flujo, un recurso o cualquier entidad conveniente. Un programa cliente invocará la carga del XAML, y que luego construye un Model3DCollection completo, para ser usado por la aplicación como se desee.

El siguiente ejemplo suministra un MeshPrimitive3D explícitamente declarado, a través del uso de la sintaxis de propiedad compleja XAML. La malla será texturizada con un

LinearGradient de amarillo a rojo. Hay también una luz en la escena.

```
<Model3DCollection>
```

```
<!-- Model children go as children here -->
```

```
<PointLight position="0,1,0" diffuseColor="white" />
```

```
<MeshPrimitive3D material="LinearGradient yellow red">
```

```
<MeshPrimitive3D.Mesh>
```

```
<Mesh3D
```

```
meshPrimitiveType="TriangleStrip"
```

```
positions="1,2,3, 4,5,6, 7,8,9, 10,11,12, 13,14,15, 16,17,18"
```

```
normals="... sensible normal vectors ..."
```

```
textureCoordinates=".5,.5, 1,1, 0,0, 25,25, .3,.4, .7,.8"
```

```
triangleIndices="1,3,4,1,2,3,4,5,6,1,2,4,2" />
```

```
</MeshPrimitive3D.Mesh>
```

```
</MeshPrimitive3D>
```

```
</Model3DCollection>
```

El ejemplo toma el primer archivo .x y lo adiciona en una animación especificada en XAML. Este ejemplo particular adiciona una escala uniforme que escala el archivo .x de 1x a 2.5x en 5 segundos, reversa, y repite indefinidamente. Este también usa aceleración/desaceleración para retardar dentro/retardar fuera de su escala:

```
<Model3DCollection>
```

```
<!-- Model children go as children here -->
```

```
<PointLight position="0,1,0" diffuseColor="white" />
```

```
<ImportedPrimitive3D xfile="myFile.x">
```

```
<ImportedPrimitive3D.Transform>
```

```
<ScaleTransform3D>
```

```
<ScaleTransform3D.ScaleVector>
```

```
<VectorAnimation
```

```
from="1,1,1"
```

```
to="2.5,2.5,2.5"
```

```
begin="immediately"
```

```
duration="5"
```

```
autoReverse="true"
```

```
repeatDuration="indefinite"
```

```
acceleration="0.1"
```

```
deceleration="0.1" />
```

```
</ScaleTransform3D.ScaleVector>
```

```
<ScaleTransform3D>
```

```
</ImportedPrimitive3D.Transform>
```

```
</ImportedPrimitive3D>
```

```
</Model3DCollection>
```

Este ejemplo importa un archivo .x y aplica un UI activo como su material:

```

<Model3DCollection>

  <!-- Model children go as children here -->
  <PointLight position="0,1,0" diffuseColor="white" />

  <ImportedPrimitive3D xfile="myFile.x">
    <ImportedPrimitive3D.OverridingMaterial>
      <VisualMaterial>
        <Button Text="Press Me" OnClick="button_OnClick" />
      </VisualMaterial>
    </ImportedPrimitive3D.OverridingMaterial>
  </ImportedPrimitive3D>

</Model3DCollection>

```

XAML soporta sintaxis para referenciar recursos. "Recursos" en este contexto es un término amplio, y este implica una sección "recurso" de un archivo XAML, así como la carga del recurso administrado "bifurcación" de un montaje administrado. Ambos mecanismos son soportados, y los recursos pueden ser referenciados desde múltiples sitios para compartirlos. Este no es un mecanismo actual para cargar directamente recursos nativos Win32, sin embargo, la implementación de una interfaz IResourceLoader lo permite.

Aunque esto no es directamente relacionado a XAML, esto es relacionado a como las aplicaciones podrán ser construidas para que compartan cosas como mallas y materiales. Note que

el mismo valor que el valor instantáneo de este objeto, pero que no varia en el tiempo. Si la propiedad DoesChange es falsa, la propiedad CurrentValue puede regresar al objeto mismo envés de una nueva copia.

SetDefaultParentTimeline

La línea de tiempo que es la pariente de cualesquier líneas de tiempo auto emparentadas. Si esta propiedad está establecida, cualesquier líneas auto emparentadas son reemparentadas, pero un nuevo clon no es creado sea para las líneas de tiempo o para este objeto.

Las clases Modifier, y por lo tanto TimeModifiers y Animations, se derivarán de Changable en lugar de Animatable porque sus propiedades individuales no pueden ser animadas. Esto refuerza el hecho de que los programadores no deben ser capaces de animar la propiedad From en una animación.

Las clases Modifier no pueden tener un valor de propiedad StatusOfNextUse de UnChangeable. El valor por defecto para StatusOfNextUse para los Modifier es ChangeableCopy, sin embargo, este también puede estar establecido en ChangeableReference si el usuario desea reutilizar un

Modifier. Si el usuario establece el `StatusOfNextUse` en `ChangeableReference` una excepción es lanzada si cualquier `Modifier` adjunto no tiene una propiedad `ParentTimeline` establecida. Esto impedirá que sucedan situaciones que tienen líneas de tiempo parientes heredadas conflictivas. Las ramas no animadas, y no cambiables de un `Animatable` pueden tener un valor `StatusOfNectUse` de `UnChangeable`, y pueden ser hechas inmodificables sobre el uso. Las propiedades sobre una clase `Modifier`, tales como `From`, `To`, o `By` permanecen modificables o cambiables durante el tiempo de vida de ese `Modifier`.

Los `Modifiers` son cambiables durante su tiempo de vida, de modo que `MakeUnChangeable` lanzará una excepción sobre estas clases. Para un `Animatable` que usualmente contiene una animación, `MakeUnChangeable` lanzará una excepción.

Si el usuario firma para notificaciones `Changed` en una clase `Animatable`, el usuario recibirá notificaciones para los cambios causados sea por los cambios de propiedad o por la naturaleza de la animación. Esto es el usuario recibirá notificaciones cambiadas cuando el `Timelines` asociados con las animaciones usadas por el `Animatable` son `Seeked` o movidas hacia delante a medida que ellas están sobre cada cuadro que está siendo presentado.

En el caso de propiedades animadas independientemente (por ejemplo Opacity) o Animatables (por ejemplo SolidColorBrush), las notificaciones Changed enviadas a cualquier usuario que ha suministrado un manejador ocurrirán en la velocidad de cuadro roscado UI no en la velocidad de cuadro compositor. El valor exacto de una animación en este caso no esta garantizado exactamente lo que es en la pantalla, aunque los valores deben ser cercanos.

Si la animación es dependiente o dependiente MIL es posible obtener un valor que cace con lo que estará en la pantalla aunque no es posible actualmente decir que notificación corresponde a el paso de conversión y por lo tanto cual refleja el valor que va a ser visualizado. Si el árbol de temporización es cambiado durante el paso de conversión, como puede ocurrir frecuentemente, es posible para el usuario recibir múltiples notificaciones, y por lo tanto aun menos probable que el usuario conozca cual corresponde al valor eventual en la pantalla.

Un modificador es un objeto que implementa un método GetValue que toma como una entrada un objeto, llamado el "valor de base", o un cierto tipo, y regresa otro objeto del mismo tipo que el de la entrada. El valor de la salida depende tanto de la entrada como del estado interno del modificador. En particular, esto significa que al llamar GetValue más de una

vez con la misma entrada no se garantiza que se regresa la misma salida. La animación grafica ocurre cuando un método modificador `GetValue` es llamado una vez por cuadro produciendo un nuevo valor para cada cuadro.

En un caso general no hay garantías acerca del valor regresado del método `GetValue`, y cada vez que el método es llamado puede retornar un valor diferente. Los objetos que consumen los modificadores pueden asumir que este es el caso y llamar al modificador repetidamente, como en el siguiente ejemplo:

```
class MyObject
{
    private Modifier myModifier;
    private object myBaseValue;

    public DoSomeWork()
    {
        object currentValue = myModifier.GetValue(myBaseValue);

        DoSomethingWithCurrentValue(currentValue);
        PostCallbackToDoSomeWork(); // Causes this method to be called again
    }
}
```

En la práctica, sin embargo, puede haber veces cuando un modificador espera en la misma salida dada la misma entrada,

dependiendo de su estado interno. Un modificador se dice que está "cambiando" cuando está en un periodo en el cual el valor de retorno del GetValue puede ser diferente en cada invocación. No está "cambiando" cuando el valor de retorno del GetValue es el mismo en cada invocación. Si un modificador "no está cambiando" un usuario de ese modificador puede guardar de manera segura el valor de retorno del método GetValue, y quizás evitar evaluar el método GetValue repetida e innecesariamente, como en el siguiente ejemplo:

```
class MyObject
{
    private Modifier myModifier;
    private object myBaseValue;

    public Initialize()
    {
        myModifier.ChangeBegun += new EventHandler(this.OnChangeBegun);
    }
}
```

```
}

public DoSomeWork()
{
    object currentValue = myModifier.GetValue(myBaseValue);

    DoSomethingWithCurrentValue(currentValue);
    if(myModifier.IsChanging)
    {
        PostCallbackToDoSomeWork(); // Do more work shortly
    }
    else
    {
        // Do nothing, since the modifier isn't going to change
        // any time soon. If it does start to change, the delegate
        // above will restart our processing loop.
    }
}

public OnChangeBegun()
{
    // The modifier is changing again, so start doing work on
    // a schedule again
    PostCallbackToDoSomeWork();
}
}
```

Una clase Modifier abstracta es implementada, de la cual los modificadores requieren ser heredados. Esta clase suministra

implementaciones por defecto para todos excepto los métodos GetValue y GetUniqueInstance:

```
public abstract class System.Windows.Media.Animation.Modifier
{
    public virtual Timeline ParentTimeline { get; set; }
    public virtual bool IsChanging { get; }
    public virtual bool IsOverridingBaseValue { get; }
    Modifier abstract GetUniqueModifierInstance(Timeline defaultParentTimeline);
    object abstract GetValue(object baseValue);
    public virtual bool UsesBaseValue { get; }

    // Event notifications
    public virtual event EventHandler Changed { add; remove; }
}
```

La siguiente tabla resume las semánticas para la clase Modifier:

Método, propiedad o evento	Significado
Changed	Surge para el modificador siempre que su estado interno cambia.
ParentTimeline	La línea de tiempo que es pariente de cualesquier líneas de tiempo auto emparentadas en este modificador. Si esta propiedad está establecida, cualesquier líneas de tiempo auto emparentadas en este modificador son reemparentadas a la nueva línea de tiempo pariente.
GetUniqueInstance	Regresa un caso de este modificador

que puede mantener su propio estado de tiempo de corrida separadamente de otros casos. Si este modificador contiene líneas de tiempo auto emparentadas, el caso regresado tiene aquellas líneas de tiempo emparentadas a la línea de tiempo pasada como un parámetro.

GetValue Calcula el valor de salida actual de este modificador, con base en el valor de base pasado como un argumento y el estado interno del modificador. Cuando la propiedad **IsOverridingBaseValue** es falsa, esta función esta garantizada para regresar el valor de base.

IsChaging Verdad si el modificador está cambiando actualmente, falso si no está en un periodo de no cambio. Está, marca es usada mejor en conjunto con los eventos **ChangeBegun** y **ChangeEnded**. Si esta marca o bandera es verdad, el valor **IsOverridingBaseValue** también requiere ser verdad.

IsOverridingBaseValue Verdad si el valor de retorno del método **GetValue** estás actualmente afectado por un modificador. Cuando este valor es falso, **GetValue** está garantizado para que regrese el mismo objeto que ha sido pasado como un argumento. Note que un modificador puede pasar por alto el valor base pero no cambiar.

UsesBaseValue Verdad si el valor de retorno del **GetValue** depende del valor de base. Si esta propiedad es falsa esto significa que el modificador ignora el valor base también. Si el modificador está en una lista, esta

propiedad permite una optimización en donde solamente un subconjunto de modificadores requieren ser evaluados en algunos casos.

En adición, un conjunto de clase de tipo específico está implementado para heredar de Modifier pero expone versiones tipo seguro de los métodos de interfaz. Los ejemplos siguientes muestran la clase FloatModifier:

```
public abstract class System.Windows.Media.Animation.FloatModifier : Modifier
{
    // Type-specific methods
    public sealed override object GetValue(object baseValue)
    {
        return GetValue((float)baseValue);
    }
    public abstract float GetValue(float baseValue);
}
```

Un modificador temporizado es un modificador cuyo comportamiento está controlado por lo menos parcialmente por un objeto Timeline. El modificador anteriormente mencionado rige y aplica, pero en adición un modificador temporizado implementa la interfaz ITimingControl para exponer el control de la línea de tiempo del modificador. No hay una clase TimeModifier abstracta. En su lugar, la clases tipo específico se heredan de las clases Modifier tipo específico. El siguiente ejemplo muestra la clase FloatTimedModifier:


```

public abstract class System.Windows.Media.Animation.FloatTimedModifier :
    FloatModifier, ITimingControl
{
    protected FloatTimedModifier(FloatTimedModifier example);

    // FloatModifier methods, properties and events
    public override Timeline ParentTimeline { get; set; }
    public override bool IsChanging { get; }
    public override bool IsOverridingBaseValue { get; }
    public override FloatModifier GetUniqueInstance(Timeline defaultParentTimeline);
    public override event EventHandler Changed { add; remove; }

    // ITimingControl methods, properties and events
    double Acceleration { get; set; }
    bool AutoReverse { get; set; }
    TimeSyncValue Begin { get; set; }
    double Deceleration { get; set; }
    Time Duration { get; set; }
    TimeSyncValue End { get; set; }
    TimeEndSync EndSync { get; set; }
    TimeFill Fill { get; set; }
    TimeFill FillDefault { get; set; }
    Timeline ParentTimeline { get; set; }
    double RepeatCount { get; set; }
    Time RepeatDuration { get; set; }
    TimeRestart Restart { get; set; }
    TimeRestart RestartDefault { get; set; }
    double Speed { get; set; }
    int CurrentRepeat { get; }
    Time CurrentTime { get; }

```

```

bool IsForwardProgressing { get; }
bool IsPaused { get; }
bool IsReversed { get; }
double Progress { get; }
void BeginIn(Time offset);
void EndIn(Time offset);
void Pause();
void Resume();
void Seek(Time offset, TimeSeekOrigin origin);
event EventHandler ChangeBegan { add; remove; }
event EventHandler ChangeEnded { add; remove; }
event EventHandler Paused { add; remove; }
event EventHandler Repeated { add; remove; }
event EventHandler Resumed { add; remove; }
event EventHandler Reversed { add; remove; }
event EventHandler Searched { add; remove; }

// Data
protected Timeline Timeline;
}

```

Note que la interfaz `Modifier` y `ITimingControl` tienen algunos métodos similares, propiedades y eventos similares. Un `TimedModifier` expone una implementación única para estos. Un `TimedModifier` es libre de implementar `ITimingControl` por medio de dirigir todas las llamadas al `Timeline` que controla, aunque no requiere hacer eso. La implementación por defecto de `ITimingControl` suministrada por el `Timed Modifier` tipo específico dirige las llamadas al `Timeline` de control.

Una animación es un modificador temporizado que implementa una función de animación particular.

```
public sealed class System.Windows.Media.Animation.FloatAnimation :  
    FloatTimedModifier  
{
```

```
    public FloatAnimation(float from, float to, Time duration);  
    public FloatAnimation(float from, float to, Time duration, TimeFill fill);  
    public FloatAnimation(float to, Time duration);  
    public FloatAnimation(float to, Time duration, TimeFill fill);  
  
    // All FloatTimedModifier methods, properties and events are inherited,  
    // plus the following are added:  
  
    public float By { get; set; }  
    public float From { get; set; }  
    public InterpolationMethod InterpolationMethod { get; set; }  
    public bool IsAccumulating { get; set; }  
    public KeySplineEnumerator KeySplines { get; set; }  
    public KeyTimeEnumerator KeyTimes { get; set; }  
    public FloatKeyValueEnumerator KeyValues { get; set; }  
    public float To { get; set; }  
}
```

Una colección de animación es una lista de objetos de animación (que se heredan del <Type>Modifier) en donde la salida del método GetValue del primer objeto es usado como el

18/2X

parámetro de valor base para el método GetValue en el segundo objeto, etc. Para flexibilidad, los objetos contenidos en una colección de animación son realmente del tipo Modifier tipo especificado. La colección en su totalidad soporta un método GetValue que parece similar al IModifier.GetValue de hecho, las colecciones de animación soportan la mayoría de las interfaces IModifier pero no implementan realmente el IModifier porque ellas no soportan la propiedad "UsesBaseValue" (está propiedad es siempre asumida como "verdad" para la colección en su totalidad).

400

```

public sealed class System.Windows.Media.Animation.FloatAnimationCollection :
    ICollection
{
    public Timeline DefaultParentTimeline { get; set; }
    public bool IsChanging { get; }

```

```

    public bool IsOverridingBaseValue { get; }
    public FloatAnimationCollection GetUniqueInstance(Timeline
        defaultParentTimeline);
    public float GetValue(float baseValue);

    // Event notifications
    public event EventHandler ChangeBegan { add; remove; }
    public event EventHandler Changed { add; remove; }
    public event EventHandler ChangeEnded { add; remove; }

    // ICollection MPEs supported
    public FloatModifier this[int index] { get; }
}

```

Los eventos disparados desde las colecciones de animación están enlazados.

Las animaciones de trayecto son una especialización de la clase `TimeMatrixModifier`. Una `MatrixModifier` puede ser usada junto con una `MatrixTransform`. Una `MatrixTransform` tiene una propiedad `Matrix`, y una propiedad `MatrixAnimations` y ya que un `PathAnimation` es una `MatrixModifier`, está puede ser usada como una `MatrixAnimation`.

329

```

public sealed class System.Windows.Media.Animation.PathAnimation :
    TimedMatrixModifier
{
    public PathGeometry PathGeometry { get; set; }
    public bool DoesRotateWithTangent { get; set; }
}

```

Método, propiedad o evento
Geometry

Significado

Esta puede ser cualquier geometría. Para elipses, un punto de partida apropiado para el progreso 0 se selecciona. Se la geometría tiene muchas sub geometrías, cada uno de sus trayectos puede ser movido a lo largo a su vez orden para ser definido al interior de la geometría.

DoesRotateWithTangent

Si está propiedad se establece en falso, el objeto se moverá a lo largo del trayecto de geometría sin ninguna rotación. Si se establece en verdad, el objeto rotará para cazar con la tangente del trayecto en cualquier localización dada.

Uso de marca:

830

```

<Canvas>
  <Canvas.TransformEffect>
    <MatrixTransform> <!-- default Matrix base value is identity -->
      <MatrixTransform.Matrix>
        <PathAnimation Begin="0" Duration="10" DoesRotateWithTangent="true"/>
          <PathAnimation.Geometry>
            <PathGeometry> ... </PathGeometry>
          </PathAnimation.Geometry>
        </PathAnimation>
      </MatrixTransform.Matrix>
    </MatrixTransform>
  </Canvas.TransformEffect>
</Canvas>

```

Cada recurso, método u objeto que puede ser animado sigue un número de reglas, incluyendo aquellas que implementan la interfaz Animatable. Para cada propiedad animable (o parámetro) llamado "Foo" o tipo "Bar", hay otra propiedad (o parámetro) llamado "FooAnimations", o tipo "BarAnimationCollection". Siempre que la animación es deseable, las colecciones de animación son usadas. Los objetos Modifier o Animation básicos no son usados directamente por que esto precluye la composición de la animación. Los recursos pueden ser animados agregando colecciones de animación a propiedades individuales. El siguiente ejemplo muestra como crear un SolidColorBrush con un color animado:

```

ColorAnimation anim = new ColorAnimation ();
animBuilder.From = new Color(1.0f, 0.0f, 0.0f, 0.0f);
animBuilder.Begin = new Time(0);
animBuilder.To = new Color(1.0f, 1.0f, 0.0f, 0.0f);
animBuilder.Duration = new Time(1000);
animBuilder.AutoReverse = true;
animBuilder.RepeatDuration = Time.Indefinite;

SolidColorBrush brush = new SolidColorBrush ();
brush.Color = new Color(1.0f, 0.0f, 0.0f, 0.0f);
brush.ColorAnimations = anim;

```

Los recursos Animate pueden ser usados en las operaciones de conversión o como valores para las propiedades Element.

Una operación de conversión puede ser animada por medio de agregar colecciones de animación al método de contexto de dibujo que llama o usando recursos animados. El siguiente ejemplo muestra como empujar un valor de opacidad animado en un contexto de dibujo:

```

FloatAnimation anim = new FloatAnimation ();
anim.From = 0.0f;
anim.Begin = Time.Immediately;
anim.To = 1.0f;
anim.Duration = new Time(1000);
anim.Fill = TimeFill.Freeze;

myDrawingContext.PushOpacity(0.0f, anim);

```


Los elementos pueden ser animados agregando colecciones de animación a las propiedades `Element`. El siguiente ejemplo muestra como animar el ancho de un botón en C#:

```
LengthAnimation anim = new LengthAnimation ();  
anim.From = new Length(50);  
anim.Begin = Time.Immediately;
```

```
anim.To = new Length(100);  
anim.Duration = new Time(1000);  
anim.Acceleration = 0.2;  
anim.Deceleration = 0.2;  
anim.Fill = TimeFill.Freeze;  
  
myButton.Width = new Length(50);  
myButton.WidthAnimations = anim;
```

La siguiente muestra el mismo ejemplo en XAML:

```
<Button ID="myButton" Width="50">  
  <Button.Width>  
    <LengthAnimationCollection>  
      <LengthAnimation  
        From="50"  
        Begin="Immediately"  
        To="100"  
        Duration="1"  
        Acceleration="0.2"  
        Deceleration="0.2"  
        Fill="Freeze"  
      />  
    </LengthAnimationCollection>  
  </Button.Width>  
</Button>
```

Siempre que una animación (o recurso animado) es usado, la animación (o recurso) es clonado (de una manera suave y eficiente) para suministrar el destino con una línea de tiempo independientemente controlable única. Un efecto colateral de este comportamiento es que la animación original no es parte de una escena visual, y por lo tanto no responde a las llamadas de control por la interfaz `ITimingControl`. Para lograr este efecto, el código llamado usa primero una animación y luego lee en la animación de nuevo. El valor que es leído de nuevo puede entonces ser colocado en memoria cache y usado para control de temporización. El siguiente ejemplo muestra un patrón que codifica la intención de animaciones de control que pueden continuar:

```
private FloatAnimation myOpacityAnimation;

public void Initialize()
{
    FloatAnimation anim = new FloatAnimation ();

    // Set the Begin property to Indefinite because we want to start
    // this animation interactively, not automatically.
    anim.Begin = Time.Indefinitely;
    anim.From = 1.0f; // Fully opaque
    anim.Duration = new Time(500); // half a second
    anim.To = 0.5f; // Half transparent
    anim.AutoReverse = true;

    // Animate the opacity of some element we own
    myElement.Opacity = 1.0f;
    myElement.OpacityAnimations = anim;

    // ERROR: The following line doesn't have the intended result:
    // myOpacityAnimation = animation;
    //
    // This line caches an animation "template", not the actual animation
    // that controls the opacity of the element.

    // This caches the right animation -- the one that is actually in use:
    myOpacityAnimation = (FloatAnimation)myElement.OpacityAnimations[0];
}

...

public void OnSomeEvent()
{
```

```
// Whenever we detect some event, "blink" the element
myOpacityAnimation.BeginIn(0);
}
```

El usuario crea una nueva clase usando AnimationEffect como una clase base para implementar un AnimationEffect. El usuario también requerirá crear un constructor para sus AnimationEffect.

```
public abstract class AnimationEffect : Changeable
{
    protected AnimationEffect(AnimationEffect effect);
    public abstract AnimationEffect GetUniqueInstance(Timeline
        defaultParent(Timeline);

    protected void Invalidate();
    protected void InvalidatePassive();
    protected bool IsInvalid { get; }
    protected Element Element { get; }

    protected virtual void AttachImpl();
    protected virtual void DetachImpl();
    protected virtual void PreLayoutReadImpl();
    protected virtual void PreLayoutWriteImpl();
    protected virtual void PostLayoutReadImpl();
}
```

Método, propiedad Significado

o evento

Invalidate	El usuario llamará esto cuando el quiere que sus AnimatioEffect colocados en la lista de AnimationEffects sea procesado durante el siguiente RenderQueueItem y quiere asegurarse de que un RenderQueueItem es programado. Lista de animaciones inválidas es reestablecida al comienzo del RederQueueItem.
InvalidatePassive	El usuario llamará está cuando quiere que su AnimationEffect este colocado en la lista de AnimationEffect para ser procesada durante el siguiente RederQueueItem, pero no quiere que suceda que un RenderQueueItem sea programado.
IsInvalid	Retorna verdadero si la animación está actualmente en la lista de AnimationEffects que va a ser procesada durante el siguiente RenderQueueItem. Podría ser verdadero porque Invalidate ha sido llamado.
Element	Este es el Element al cual el AnimationEffect está sujetado. Si el AnimationEffect sujetado a un Element, este lanzará una excepción. El usuario no hace ningún establecimiento hasta que

OnAttach ha sido llamado.

AttachImpl

Cuando un AnimationEffect está sujeto a un elemento, este es clonado automáticamente y el nuevo clon es agregado a la colección de AnimatioEffects sobre el elemento y ha llamado el OnAttach. En este punto, la propiedad protegida Element en el AnimationEffect será establecida.

Si el usuario ha agregado un AnimationEffect a una colección, solamente el nuevo AnimationEffect habrá llamado el OnAttach. No hay garantías de que cuando OnAttach es llamado el elemento tendrá sus propiedades de marcación establecidas o que el hijo del Element estará en su lugar. Un AnimationEffect es clonado. Aunque el Element podría ser pasado al AnimationEffect cuando se llama cada función, este no puede pasar a manejadores de evento para los eventos desde otros elementos que estarán en donde el AnimationEffect lo requiere más. El AnimationEffect puede establecer

manejadores de eventos sobre otros elementos pero todavía requerirá conocer que esta asignado a este Element.

DetachImpl Este llamará sobre un AnimationEffect cuando se ha liberado de un Element.

PreLayoutReadImpl Este será llamado sobre un AnimationEffect si está sucio antes de que se corra el diseño en el RenderQueueItem. Este es el tiempo cuando el AnimationEffect debe leer los valores que requiere. La razón de leer y escribir son separadas es que leer causa que el diseño corra inmediatamente y si cada AnimationEffect lee y escribe a su vez este ralentizará el proceso completo.

PreLayoutWriteImpl Este será llamado sobre un AnimationEffect si esta sucio antes de que se corra el diseño en el RenderQueueItem. Aunque no se garantiza el orden en el cual AnimationEffect será procesado, si garantizamos que todos los AnimationEffect sucios habrán llamado OnPreLayoutread Antes de ser llamado.

PostlayoutReadImpl Este será llamado sobre un AnimationEffect si esta sucio después de

10371

que se corre el diseño en el RenderQueueItem si la bandera IsAlwaysDirty no está establecida, la bandera sucia en este AnimationEffect habrá sido establecida en falso y está habrá sido removida de la lista AnimationEffects`a ser procesados durante el siguiente RenderQueueItem. Si el AnimatioEffect llama SetDirty en este método, este efectivamente mantendrá su suciedad durante el procesamiento durante el siguiente RenderQueueItem. Si el AnimationEffect tendría que permanecer sucio, es mucho más eficiente establecer la bandera IsAlwaysDirty.

TIPOS PRIMITIVOS

La unidad de longitud base en el MIL es un doble, por lo cual los otros tipos primitivos y APIs están basados en dobles. Generalmente estos dobles son evaluados como unidades de usuario que inicialmente son iguales a 1/96 de una pulgada. Para los colores, cada uno de los canales de colores es representado por un float, y no por un doble. Para medidas de ángulos, los valores doble están en grados. Cuando un float o

820

un doble es evaluado como una medida de tiempo, este es asumido en segundos.

La estructura Time representa un punto particular en el tiempo o una extensión de tiempo. Adicional, un valor especial de tiempo llamado "indefinido" representa un punto en el tiempo infinito en el futuro o una extensión de tiempo infinitamente grande. Los valores de Time son designados para ser usados en un sistema de propiedades, como un valor especial llamado "no especificado" pueden ser usados para limpiar una propiedad, o para indicar explícitamente que la propiedad no es asignada. Los valores de Time son internamente almacenados como contadores enteros:

sig

time:

(full-clock-value | partial-clock-value | timecount-value | special-value)

full-clock-value:

hours ":" minutes ":" seconds ("." fraction)?

partial-clock-value:

minutes ":" seconds ("." fraction)?

timecount-value:

timecount ("." fraction)? (metric)?

special-value:

("Indefinite" | "Unspecified")

metric:

"h" | "min" | "s" | "ms"

hours:

digit+

minutes:

2digit

seconds:

2digit

fraction:

digit+

timecount:

digit+

2digit:

digit digit

digit:

"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Adicionalmente a la gramática de arriba, note que "minutos" y "segundos" necesitan ser especificados en el rango "00" a "59" para ser considerados válidos. También, si el formato "timecount-value" es usado sin unidades, el valor es asumido en segundos. Los siguientes son ejemplos de valores Time y sus significados:

Time	Valor
02:30:03	2 horas, 30 minutos y 3 segundos
50:00:10.25	50 horas, 10 segundos y 250 milisegundos
02:33	2 minutos y 33 segundos
00:10.5	10 segundos y 500 milisegundos
3.2h	3 horas y 12 minutos
45min	45 minutos
30s	30 segundos
5.45ms	5.45 milisegundos
12.467	12 segundos y 467 milisegundos
1d	1 día

La estructura Time es utilizada para almacenar un solo valor Time:

```
public struct System.Windows.Media.Animation.Time : IComparable
{
    public Time(int milliseconds);

    public bool IsFinite { get; }
    public static Time Abs(Time t);
    public int CompareTo(Time other);
    public override bool Equals(object obj);
    public override int GetHashCode();
    public static Time Max(Time a, Time b);
    public static Time Min(Time a, Time b);
    public override string ToString();
    public static Time operator +(Time a, Time b);
    public static Time operator -(Time t);
}
```

Done

```

public static Time operator -(Time a, Time b);
public static Time operator *(double s, Time t);
public static Time operator *(Time t, double s);
public static Time operator /(Time t, double s);
public static double operator /(Time a, Time b);
public static Time operator %(Time a, Time b);
public static bool operator ==(int n, Time time);
public static bool operator ==(Time time, int n);
public static bool operator ==(Time a, Time b);
public static bool operator !=(int n, Time time);
public static bool operator !=(Time time, int n);
public static bool operator !=(Time a, Time b);
public static bool operator >=(Time a, Time b);
public static bool operator <=(Time a, Time b);
public static bool operator >(Time a, Time b);
public static bool operator <(Time a, Time b);

public static readonly Time Immediately;
public static readonly Time Indefinite;
public static readonly int MaxValue;
public static readonly int MinValue;
public static readonly Time Unspecified;
}

```

Otros tipos básicos son asignados adelante, en los cuales la siguiente notación es utilizada:

- *: 0 o más
- +: 1 o más
- ?: 0 o 1
- {n}: n veces
- (): agrupación
- |: alternativas separadas
- Doble comillas para encerrar literales

805

wsp:

whitespace+

digit:

"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

hex-digit:

digit | "a" | "A" | "b" | "B" | "c" | "C" | "d" | "D" | "e" | "E" | "f" | "F"

digit-sequence:

digit+

sign:

"-" | "+"

exponent:

("e" | "E") sign? digit-sequence

fractional-constant:

(digit-sequence? "." digit-sequence) | (digit-sequence ".")

floating-point-constant:

(fractional-constant exponent?) | (digit-sequence exponent)

integer-constant:

digit-sequence

comma:

" , "

comma-wsp:

get

```

( wsp comma? wsp* ) | ( comma wsp* )

number:
( sign? ( integer-constant | floating-point-constant ) ) | "Unset"

number-wsp:
number wsp*

coordinate:
number-wsp

coordinate-pair:
coordinate comma-wsp coordinate

coordinate-pairs:
( coordinate-pair comma-wsp )* coordinate-pair

filename:
wsp* ( "" valid-filename-char+ "" | valid-filename-char-no-whitespace ) wsp*

```

Sintaxis de marcas para colores:

```

numeric-color:
"#" ( hex-digit{3} | hex-digit{4} | hex-digit{6} | hex-digit{8} )

color:
numeric-color | [a-zA-Z]+

```

El objeto Colors contiene miembros estáticos, comprende colores muy conocidos como Rojo y Azul:

get

```
public sealed class Colors
{
    public static Color AliceBlue { get; }
    public static Color AntiqueWhite { get; }
    public static Color Aqua { get; }
    public static Color Aquamarine { get; }
    public static Color Azure { get; }
    public static Color Beige { get; }
    public static Color Bisque { get; }
    public static Color Black { get; }
    public static Color BlanchedAlmond { get; }
    public static Color Blue { get; }
    public static Color BlueViolet { get; }
    public static Color Brown { get; }
    public static Color BurlyWood { get; }
    public static Color CadetBlue { get; }
    public static Color Chartreuse { get; }
    public static Color Chocolate { get; }
    public static Color Coral { get; }
    public static Color CornflowerBlue { get; }
    public static Color Cornsilk { get; }
    public static Color Crimson { get; }
    public static Color Cyan { get; }
    public static Color DarkBlue { get; }
    public static Color DarkCyan { get; }
    public static Color DarkGoldenrod { get; }
    public static Color DarkGray { get; }
    public static Color DarkGreen { get; }
    public static Color DarkKhaki { get; }
    public static Color DarkMagenta { get; }
    public static Color DarkOliveGreen { get; }
    public static Color DarkOrange { get; }
```



```
public static Color DarkOrchid { get; }
public static Color DarkRed { get; }
public static Color DarkSalmon { get; }
public static Color DarkSeaGreen { get; }
public static Color DarkSlateBlue { get; }
public static Color DarkSlateGray { get; }
public static Color DarkTurquoise { get; }
public static Color DarkViolet { get; }
public static Color DeepPink { get; }
public static Color DeepSkyBlue { get; }
public static Color DimGray { get; }
public static Color DodgerBlue { get; }
public static Color Firebrick { get; }
public static Color FloralWhite { get; }
public static Color ForestGreen { get; }
public static Color Fuchsia { get; }
public static Color Gainsboro { get; }
public static Color GhostWhite { get; }
public static Color Gold { get; }
public static Color Goldenrod { get; }
public static Color Gray { get; }
public static Color Green { get; }
public static Color GreenYellow { get; }
public static Color Honeydew { get; }
public static Color HotPink { get; }
public static Color IndianRed { get; }
public static Color Indigo { get; }
public static Color Ivory { get; }
public static Color Khaki { get; }
public static Color Lavender { get; }
public static Color LavenderBlush { get; }
public static Color LawnGreen { get; }
```

```
public static Color LemonChiffon { get; }
public static Color LightBlue { get; }
public static Color LightCoral { get; }
public static Color LightCyan { get; }
public static Color LightGoldenrodYellow { get; }
public static Color LightGray { get; }
public static Color LightGreen { get; }
public static Color LightPink { get; }
public static Color LightSalmon { get; }
public static Color LightSeaGreen { get; }
public static Color LightSkyBlue { get; }
public static Color LightSlateGray { get; }
public static Color LightSteelBlue { get; }
public static Color LightYellow { get; }
public static Color Lime { get; }
public static Color LimeGreen { get; }
public static Color Linen { get; }
public static Color Magenta { get; }
public static Color Maroon { get; }
public static Color MediumAquaMarine { get; }
public static Color MediumBlue { get; }
public static Color MediumOrchid { get; }
public static Color MediumPurple { get; }
public static Color MediumSeaGreen { get; }
public static Color MediumSlateBlue { get; }
public static Color MediumSpringGreen { get; }
public static Color MediumTurquoise { get; }
public static Color MediumVioletRed { get; }
public static Color MidnightBlue { get; }
public static Color MintCream { get; }
public static Color MistyRose { get; }
public static Color Moccasin { get; }
```

```
public static Color NavajoWhite { get; }
public static Color Navy { get; }
public static Color OldLace { get; }
public static Color Olive { get; }
public static Color OliveDrab { get; }
public static Color Orange { get; }
public static Color OrangeRed { get; }
public static Color Orchid { get; }
public static Color PaleGoldenrod { get; }
public static Color PaleGreen { get; }
public static Color PaleTurquoise { get; }
public static Color PaleVioletRed { get; }
public static Color PapayaWhip { get; }
public static Color PeachPuff { get; }
public static Color Peru { get; }
public static Color Pink { get; }
public static Color Plum { get; }
public static Color PowderBlue { get; }
public static Color Purple { get; }
public static Color Red { get; }
public static Color RosyBrown { get; }
public static Color RoyalBlue { get; }
public static Color SaddleBrown { get; }
public static Color Salmon { get; }
public static Color SandyBrown { get; }
public static Color SeaGreen { get; }
public static Color SeaShell { get; }
public static Color Sienna { get; }
public static Color Silver { get; }
public static Color SkyBlue { get; }
public static Color SlateBlue { get; }
public static Color SlateGray { get; }
```

```

public static Color Snow { get; }
public static Color SpringGreen { get; }
public static Color SteelBlue { get; }
public static Color Tan { get; }
public static Color Teal { get; }
public static Color Thistle { get; }
public static Color Tomato { get; }
public static Color Transparent { get; }
public static Color Turquoise { get; }
public static Color Violet { get; }
public static Color Wheat { get; }
public static Color White { get; }
public static Color WhiteSmoke { get; }
public static Color Yellow { get; }
public static Color YellowGreen { get; }
}

```

La estructura punto es asignada como sigue:

```

public struct System.Windows.Point
{
    public Point(); // initializes to 0,0
    public Point(double x, double y);

    public static Boolean Equals(Point point1, Point point2)
    public virtual Boolean Equals(Object o)

    public Double X { get; set; }
    public Double Y { get; set; }

    public virtual Int32 GetHashCode()

    public static Point operator+(Point point, Vector vector)

```

```

public static Boolean operator==(Point point1, Point point2)
public static Boolean operator!=(Point point1, Point point2)
public static Point operator*(Point point, Matrix matrix)
public static Point operator-(Point point, Vector vector)
public static Vector operator-(Point point1, Point point2)

// This will take the absolute value of X and Y, since Size cannot be negative
public static explicit operator Size(Point point) //
public static explicit operator Vector(Point point)

public static Point Add(Point point, Vector vector)
public static Point Multiply(Point point, Matrix matrix)
public void Offset(Double dx, Double dy)
public static Point Subtract(Point point, Vector vector)
public static Vector Subtract(Point point1, Point point2)
public virtual String ToString()
)

```

Sintaxis de marcas para un objeto punto:

```

point:
    coordinate-pair

```

Un objeto Vector es asignado como sigue:

```

public struct System.Windows.Vector
{
    public Vector(); // initializes to 0,0
    public Vector(double x, double y);

    public double X { get; set; }
    public double Y { get; set; }
}

```

```

public static Boolean Equals(Vector vector1, Vector vector2)
public virtual Boolean Equals(Object obj)

public Double Length { get; }
public Double LengthSquared { get; }
public Double X { get; set; }
public Double Y { get; set; }

public virtual Int32 GetHashCode()

public static Point operator+ (Vector vector, Point point)
public static Vector operator+ (Vector vector1, Vector vector2)
public static Vector operator/ (Vector vector, Double scalar)
public static Boolean operator==(Vector vector1, Vector vector2)
public static Boolean operator!= (Vector vector1, Vector vector2)
    // return the dot product: vector1.X*vector2.X + vector1.Y*vector2.Y
public static Double operator*(Vector vector1, Vector vector2)
public static Vector operator* (Double scalar, Vector vector)
public static Vector operator* (Vector vector, Double scalar)
public static Vector operator* (Vector vector, Matrix matrix)
public static Vector operator-(Vector vector1, Vector vector2)
public static Vector operator- (Vector vector)

    // This will take the absolute value of X and Y, since Size cannot be negative
public static explicit operator Point (Vector vector)
public static explicit operator Size (Vector vector)

public static Vector Subtract(Vector vector1, Vector vector2)
public static Double Multiply(Vector vector1, Vector vector2)
public static Vector Multiply(Double scalar, Vector vector)
public static Vector Multiply(Vector vector, Double scalar)
public static Vector Multiply(Vector vector, Matrix matrix)

```

```

public static Vector Divide(Vector vector, Double scalar)
public static Point Add(Vector vector, Point point)
public static Vector Add(Vector vector1, Vector vector2)

public Void Normalize()
    // return the angle required to rotate v1 into v2, in degrees
    // This will return a value between (-180, 180] degrees
public static Double AngleBetween(Vector vector1, Vector vector2)
public static Double CrossProduct(Vector vector1, Vector vector2)

    // return the Determinant: vector1.X*vector2.Y - vector1.Y*vector2.X
public static Double Determinant(Vector vector1, Vector vector2)

public virtual String ToString()

}

```

Sintaxis de marca para un objeto Vector:

size:

coordinate-pair

Note que tamaño no es un Vector, y así este no puede ser adicionado, restado o transformado. También el tamaño no puede tener un ancho o alto negativo. Un ArgumentException puede ser lanzado si se trata de asignar una de estas.

427

Con respecto a los rectángulos, el nombre `Rect` es utilizado en lugar de `Rectangle` para que no colisione con el elemento `Rectangle`. Mientras que es posible especificar un ancho o alto negativo, esta no es una manera estándar de normalizar el rectángulo y muchas operaciones en los rectángulos pueden dar resultados no intuitivos. El Ancho y el Alto de los `Rect` no pueden ser asignados a valores no negativos. Si tales valores son pasados al constructor, el resultado podrá ser normalizado. Si el Ancho o el Alto es asignado a negativo, esto podrá resultar en un `ArgumentException`.

Un `Rect` contiene puntos dentro de éste, todos los puntos tienden en sus esquinas. Así, los ángulos de un `Rect` son inclusivos. Como consecuencia, un `Rect` de Ancho 0 o Altura 0 no es vacío, como este contiene un conjunto de puntos encontrados en un segmento de línea unidimensional representados por un `Rect`. Si el Alto y el Ancho son 0, entonces el `Rect` contiene un `Point` en `Location`. Esto significa que un `Rect` verdaderamente vacío es un valor especial, y puede ser accedido vía la propiedad estática `EmptyRect`. Las propiedades `Alto` y `Ancho` de `EmptyRect` podrán retornar infinito negativo (solo en el caso donde ambas sean negativas), las propiedades `X` y `Y` podrán retornar infinito positivo, y ninguna de esas propiedades será modificable. Esto es para asegurar que no tiene un ancho "vacío" pero altura normal, o viceversa.

856


```
public struct System.Windows.Rect
{
    public Rect (); // Sets all values to 0
    public Rect (Point location, Size size);
    public Rect (double x, double y, double width, double height);

    // Both of these will normalize the rectangle
    public Rect (Point point1, Point point2);
    public Rect (Point point, Vector vector);

    public static Rect FromLTRB(double left, double top, double right, double
bottom);

    public static Rect Empty { get; } // returns the Empty Rect
    public bool IsEmpty { get; }

    public static Rect Infinite( get; }

    public Point Location { get; set; }
    public Size Size { get; set; }
```

```

public double X { get; set; }
public double Y { get; set; }
public double Width { get; set; }
public double Height { get; set; }

public double Left { get; } // Alias to X
public double Top { get; } // Alias to Y
public double Right { get; } // X + Width
public double Bottom { get; } // Y + Height

public Point TopLeft { get; }
public Point TopRight { get; }
public Point BottomLeft { get; }
public Point BottomRight { get; }
public Point Center { get; }

public bool Contains(Point point);
public bool Contains(Rect rect);
public bool Contains(double x, double y);

public bool IntersectsWith(Rect rect);

public void Intersect(Rect rect);
public static Rect Intersect(Rect rect1, Rect rect2);

public void Union(Rect rect);
public static Rect Union(Rect rect1, Rect rect2);
public void Union(Point point);
public static Rect Union(Rect rect, Point point);

...

public void Offset(Vector offset);
public void Offset(double dx, double dy);

```

```

public static Rect Offset(Rect rect, Vector offset);
public static Rect Offset(Rect rect, double dx, double dy);

public void Inflate(Size size);

// If -width > Width*2 or -height > Height*2, the Rect becomes Empty
public void Inflate(double width, double height);

public static Rect Inflate(Rect rect, double width, double height);
public static Rect Inflate(Rect rect, Size size);

// Transforms 4 corners and sets new axis aligned rect.
public void Transform(Matrix matrix);
public static Rect Transform(Rect rect, Matrix matrix);

public static Boolean Equals(Rect rectangle1, Rect rectangle2)
public virtual Boolean Equals(Object o)
public virtual Int32 GetHashCode()

public static Boolean operator==(Rect rectangle1, Rect rectangle2)
public static Boolean operator!=(Rect rectangle1, Rect rectangle2)
public virtual String ToString()
}

```

Como se mostró, el rect suministra un número de métodos. IsEmpty retorna verdadero si la instancia es igual al Empty Rect. El método FromLTRB esencialmente retorna un rectángulo inicializado con (izquierdo, arriba, derecha-izquierda, abajo-arriba). El método Contains retorna verdadero si el Rect no es Empty, $p.x \geq r.x$, $p.x \leq r.x + r.width$, $p.y \geq r.y$ y

$p.y \leq r.y + r.height$. Esto significa que este método puede retornar falso si el rectángulo tiene un ancho o alto negativo.

`IntersectsWith` retorna verdadero si ningún `Rect` es `Empty`, $r1.x \leq r2.x + r2.width$, $r2.x \leq r1.x + r1.width$, $r1.y \leq r2.y + r2.height$, y $r2.y \leq r1.y + r1.height$. La intersección de dos rectángulos es computada tomando el máximo de las dimensiones izquierda y arriba y el mínimo de las dimensiones derecha y abajo. Si ambos `Rect` son `Empty`, el `Empty Rect` es retornado.

La unión de dos rectángulos es computada tomando el mínimo de las dimensiones izquierda y arriba y el máximo de las dimensiones derecha y abajo. Si ambos `Rect` son `Empty`, el otro `Rect` es retornado.

Para el método `Offset`, la compensación es simplemente adicionada a la posición del rectángulo. Este método no tiene efecto si el `Rect` es `Empty`.

Para `Inflate`, la cantidad de inflación es simplemente aplicada igualmente a todos los lados. Los ajustes son hechos, comprendiendo, $r.x = r.x - s.width$; $r.y = r.y - s.height$; $r.width = r.width + 2*s.width$ y $r.height = r.height + 2*s.height$. Este método no tiene efecto si el `Rect` es `Empty`, por lo que un `Empty Rect` no tiene posición.

La siguiente sintaxis de marcas especifica el alto y ancho, x, y. Debería notarse que es equivalente a lo que hace el convertidor para System.Drawing.Rectangle.

rectangle:

(coordinate comma-wsp){3} coordinate | "Empty"

Las matrices para computaciones de 2D son representadas como matrices 3x3. El MIL utiliza la sintaxis fila-vector. El mil solo permite transformaciones afines, y así solo requiere solamente seis valores, en lugar de una matriz completa 3x3. Esto es nombrado y definido como sigue.

$$\begin{bmatrix} m11 & m12 & 0 \\ m21 & m22 & 0 \\ offsetX & offsetY & 1 \end{bmatrix}$$

Cuando una matriz es multiplicada por un punto, esta transforma el punto de un nuevo sistema de coordenadas a un sistema de coordenadas anterior:

$$\begin{bmatrix} x_{newCoordSys} & y_{newCoordSys} & 1 \end{bmatrix} \cdot \begin{bmatrix} m11 & m12 & 0 \\ m21 & m22 & 0 \\ offsetX & offsetY & 1 \end{bmatrix} = \begin{bmatrix} x_{oldCoordSys} & y_{oldCoordSys} & 1 \end{bmatrix}$$

Las transformaciones pueden ser anidadas a cualquier nivel. Cuando una nueva transformación es aplicada esta es igual a una pre-multiplicación de ésta en la matriz de transformación actual:

$$\begin{bmatrix} x_{newCenter} & y_{newCenter} & 1 \end{bmatrix} \cdot \begin{bmatrix} m11 & m12 & 0 \\ m21 & m22 & 0 \\ offsetX & offsetY & 1 \end{bmatrix} = \begin{bmatrix} m11 & m12 & 0 \\ m21 & m22 & 0 \\ offsetX & offsetY & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{oldCenter} & y_{oldCenter} & 1 \end{bmatrix}$$

Muchos sitios en el API no toman una matriz directamente. En cambio, la clase Transform soporta animación en forma profunda.

```
public struct System.Windows.Media.Matrix
{
    // Construction and setting
    public Matrix(); // defaults to identity

    public Matrix(
        double m11, double m12,
        double m21, double m22,
        double offsetX, double offsetY);

    // Identity
    public static Matrix Identity { get; }
    public void SetIdentity();
    public bool IsIdentity { get; }

    // Unset support
    public static Matrix Unset { get; } // all values are NaN
    public bool IsSet { get; } // false if any NaN

    // Math operations
    public void Prepend(Matrix matrix); // "this" becomes: matrix * this
    public void Append(Matrix matrix); // "this" becomes: this * matrix
}
```

43A

```

// Matrix operations
public void Rotate(double angle); // Appends a rotate
public void RotatePrepend(double angle); // Prepends a rotate

public void RotateAt(double angle, Point center); // Appends a rotate
public void RotateAtPrepend(double angle, Point center); // Prepends a rotate

public void Scale(Double scaleX, Double scaleY); // Appends a scale
public void ScalePrepend(Double scaleX, Double scaleY); // Prepends a scale

public void ScaleAt(Double scaleX, Double scaleY, Point point); // Appends a
    scale
public void ScaleAtPrepend(Double scaleX, Double scaleY, Point point);
    // Prepends a
    scale

public void Skew(Double skewX, Double skewY); // Appends a skew, in
    degrees
public void SkewPrepend(Double skewX, Double skewY); // Prepends a skew

public void Translate(Double offsetX, Double offsetY); // Appends a translation
public void TranslatePrepend(Double offsetX, Double offsetY); // Prepends a
    translation

public static operator * (Matrix matrix1, Matrix matrix2);

// Transformation services
public Point Transform(Point point);
public void Transform(Point[] points);

// Since this is a vector ignores the offset parts of the matrix
public Vector Transform(Vector vector);

```



```

public void TransformVectors(Vector[] vectors);

// Inversion
public double Determinant { get; }
public bool HasInverse { get; }
public Matrix Inverse { get; } // Throws ArgumentException
    InvalidOperationException if !HasInverse

// Individual members
public double M11 { get; set; }
public double M12 { get; set; }
public double M21 { get; set; }
public double M22 { get; set; }
public double OffsetX { get; set; }
public double OffsetY { get; set; }
};

```

Para la sintaxis de marcas, el orden es M11, M12, M21, M22, OffsetX, OffsetY:

```

matrix:
    ( coordinate comma-wsp ){5} coordinate | "Identity"

```

ELEMENTOS Y VISUALES TRI-DIMENSIONALES

Esta sección es dirigida hacia efectos tri-dimensionales, primariamente para suministrar directamente efectos tri-dimensionales por medio de la capa de integración media para

las aplicaciones a usar. Tales efectos son integrados con distanciamiento, impresión, composición de escritorio, y participan limpiamente en la arquitectura MIL y modelo de elemento, y la posibilidad de hacer composición de medios a través de MIL. Por ejemplo, visualización de negocio tiene algunos requerimientos para la presentación tri-dimensional e interactividad. Esto puede requerir una resolución probada en miles o decenas de miles de objetos, y puede resultar en visualizaciones interactivas que pueden ver alguna cosa como la imagen mostrada en la FIG. 41.

Las características descritas aquí son generalmente direccionadas hacia representación en tiempo de ejecución e interacción en lugar de la funcionalidad que es suministrada vía herramientas de modelaje. La funcionalidad de modelaje que es suministrada es mas orientada para situaciones donde la animación o la construcción en tiempo de ejecución será necesaria para ser expuesta en tiempo de ejecución. Por ejemplo, el moldeado de texto 3D puede ser suministrado, aún pensando que esta es una característica real de modelado, puesto que las aplicaciones pueden desear juntar datos de texto, de tal modo que se requiera de esto en tiempo de ejecución.

Como se describió anteriormente, Visuals en el sistema MIL representa el árbol de composición 2D que últimamente tomó

representación. Note que `UiElements` y `Controls` son `Visuals`. Adicionalmente, los `Visuals` tienen una colección de hijos `Visuals` que son representados en el frente (en términos de algoritmo de pintor) del contenido propio de `Visual`. Como también se describió anteriormente, hay una variedad de subclases `Visual` concretas, que incluye `RetainedVisual`, `SurfaceVisual`, `HwndVisual`, y así sucesivamente. Un `RetainedVisual` representa un conjunto de instrucciones de dibujo 2D que el usuario puede crear directamente en el `Visual`, o lo hace como resultado de recibir un llamado `OnRender()` desde el sistema de esquema.

Como puede entenderse, 3D se ajusta naturalmente en el mundo visual 2D. Para finalizar, un `Retained3DVisual` es suministrado, tal que, como esta imagen 2D, representa una lista de instrucciones de dibujado que son imperativamente llenadas al abrir un contexto de dibujo, y representarlo en el contexto de dibujo. Esto efectivamente resultará en la construcción de una escena gráfica, o un meta archivo, o una lista de instrucciones en el `Visual` mismo.

Un `Retained3DVisual` es esencialmente un conjunto de datos tri-dimensionales (instrucciones de representación/ escena gráfica/ meta archivo) incluyendo luces, una cámara para definir la proyección 2D de la escena, un puerto visual de dos dimensiones rectangular en el espacio de coordenadas

locales para mapear la proyección hacia otros parámetros de ambiente, tales como interruptores antialiasing, interruptores niebla, y demás.

Note que la capacidad bajo demanda `OnRender()` no es suministrada por el `Retained3DVisual`. Esta funcionalidad existe para 2D, para ayudar a mantener la cantidad de datos dibujados en el mundo 2D. Como 2D, la representación ocurre vía un `DrawingContext` donde "modo de percepción inmediato" los llamados son realizados. Por ejemplo, en 2D, lo siguiente puede ser un contexto de dibujo:

```
DrawingContext ctx = ...;  
ctx.DrawRectangle(...);  
ctx.PushTransform(...);
```

```
ctx.DrawGeometry(...);  
ctx.PushTransform(...);  
ctx.DrawEllipse(...);  
ctx.Pop();  
ctx.Pop();
```

Por consistencia con 2D, un modelo similar es suministrado en 3D que se ve como:

440

```

DrawingContext3 ctx = ...;
ctx.DrawMesh(mesh, material);
ctx.PushTransform(transform3);
ctx.DrawMesh(...);
ctx.PushTransform(...);
ctx.DrawMesh(...);
ctx.Pop();
ctx.Pop();

```

Note que este modelo de representación trabaja bien para ambos un visual Retained Mode 3D (donde las "instrucciones" son simplemente guardadas), y un visual Immediate Mode 3D (donde la representación sucede directamente, y la cámara necesita ser estabilizada en el frente). En efecto, en el caso del modo retenido, que sucede internamente es una jerarquía de modelaje 3D siendo construida y retenida. Alternativamente, en el caso del modo inmediato, no están sucediendo cosas, las instrucciones son en sí mismas emitidas directamente, y una pila de contexto (para transformaciones, por ejemplo) es en sí misma mantenida.

El siguiente es un ejemplo que muestra programación con el API Visual 3D. Este ejemplo crea un Retained3DVisual, obtiene un contexto de dibujo para representarlo en él, representa primitivas y luces en asigna una cámara, y adiciona el visual al hijo visual de un control:

809

```

// Create a 3D visual
Retained3DVisual visual3 = new Retained3DVisual();

// Render into it
using (Drawing3DContext ctx = visual3.Models.RenderOpen())

```

```

{
    // Render meshes and lights into the geometry
    ctx.DrawMesh(mesh, material);
    ctx.PushTransform(transform3);
    ctx.DrawMesh(...);
    ctx.PushTransform(secondTransform3);
    ctx.AddLight(...);
    ctx.DrawMesh(...);
    ctx.Pop();
    ctx.Pop();
}

// Establish ambient properties on the visual
visual3.Camera = new ProjectionCamera(...);

// Add it to the compositing children of some control called myControl
((IVisual)myControl).AddChild(visual3);

```

La integración de 2D en 3D es una característica significativa, y permite escenarios muy interesantes de tener UI 2D interactuando en un mundo 3D, mapeados en superficies 3D, y así sucesivamente. 2D se integra en 3D de maneras separadas. Un camino es como una textura en una primitiva 3D.

Cualquier Brush Avalon 2D (como las especificaciones más complejas de material) puede ser utilizado en textura 3D. Un caso especial de esto es el VisualBrush el cual puede albergar un visual arbitrario (y así Controles, o Aplicación UI entera). Mecanismos son suministrados para resolver pruebas exitosas y pruebas adicionales en el visual para aplicaciones que deseen hacerlas y, en ciertos casos, la resolución de pruebas se hará tal que el Visual pueda pasarlas. Note que aunque el Visual en sí mismo no es un mapa de bits, Visual-texturing puede ser implementada a través de técnicas sofisticadas, tales como tomar la geometría 2D en el Visual y llevarla a 3D y así tener vectores gráficos 3D en lugar de mapas de bits 2D usados como texturas.

Otra forma en que 2D se integra en 3D es como una representación de instrucciones en un contexto 3D. Es también deseable habilitar simplemente un visual 2D representado, pantalla alineada, y no escalada sin algunos efectos de perspectiva, en una profundidad z particular, por ejemplo, para apilar una esfera, un visual 2D, y un cubo mientras todavía todos compartan el mismo z-búfer, de modo que permita, por ejemplo, una esfera pasar a través de una pieza de UI 2D. Esto será expuesto por una instrucción "DrawVisual" en el Drawing3DContext.

Como se describió, la Animación puede operar intuitivamente, y la tecnología de animación 2D es directamente transferible a 3D. Una excepción es racionalizando las líneas de tiempo.

Lo anterior muestra un estilo de representación imperativo de uso donde las instrucciones de dibujo son emitidas al contexto. Este no es un uso declarativo, y así esta aproximación imperativa no es apropiada para unas marcas declarativas.

Debido a que con el mecanismo anterior no hay un modelo 3D explícitamente construido externamente, (igual internamente lo es), un programador no puede direccional un modelo 3D o enumerar su contenido. Como resultado, no hay un lugar en el cual escribir "efectos" que tomen un modelo como entrada y generen un nuevo modelo.

Para suministrar una forma declarativa de construir y usar "recursos" 3D como en 2D con Brushes, Pens, Geometry, Paths, y demás, un número de tipos es suministrado tal que permita a los usuarios construir en un flujo de instrucciones 3D y que el objeto construido pueda ser asignado en un Retained3DVisual en lugar de usar el contexto.

Por ejemplo, el anterior código de ejemplo basado en Drawing3DContext puede ser rescrito como:


```

// Create a 3D visual
Retained3DVisual visual3 = new Retained3DVisual();

visual3.Models.Add(new MeshPrimitive3D(mesh, material));

Model3DCollection innerGroup1 = new Model3DCollection();
innerGroup1.Transform = transform3;
innerGroup1.Add(new MeshPrimitive3D(mesh, material));

Model3DCollection innerGroup2 = new Model3DCollection();
innerGroup2.Transform = secondTransform3;
innerGroup2.Add(new Light(...));
innerGroup2.Add(new MeshPrimitive3D(...));

```

```

innerGroup1.Add(innerGroup2);

visual3.Models.Add(innerGroup1);

// Everything else is the same as before...

// Establish ambient properties on the visual
visual3.Camera = new ProjectionCamera(...);

// Add it to the compositing children of some control called myControl
((IVisual)myControl).AddChild(visual3);

```

Aquí, un modelo es construido, y luego asignado en elRetained3DVisual. Los pares Push Transform/Pop son

reemplazados por la construcción de un `Model3DCollection` el cual en sí mismo tiene una transformación y modelos bajo éste. Suministrando a ambos la aproximación de modelo y la aproximación imperativa basada en contexto se suministra una solución para las marcas declarativas a nivel elemento, enumeración visual, efectos gráficos de escena y posibilidad de modificar el contenido visual, y esto hecho de una manera arquitectónica.

La raíz del árbol modelo de clase es `Model3D`, el cual representa un modelo tri-dimensional que puede ser adjuntado al `Reatined3DVisual`. Finalmente, luces, meshes, flujo de archivo `.X` (este puede venir de un archivo, un recurso, memoria, y demás), grupos de modelos, y visuales 2D posicionadas en 3D son todos modelos. Así, se tiene la siguiente jerarquía:

- **Model3D**
 - **Model3DCollection** – is a collection of **Model3D**
 - **Primitive3D**
 - **MeshPrimitive3D(mesh, material, hitTestID)**
 - **ImportedPrimitive3D(stream, hitTestID)** (for `.x` files)
 - **Light** --
 - **AmbientLight**
 - **SpecularLight**

- DirectionalLight
- PointLight
 - Spotlight
- VisualModel3D – has a Visual and a Point3 and a hitTestID

La clase Model3D en si misma soporta un número de operaciones, incluyendo tomar el cubo circunscrito 3D, Tomar y asignar el Transform del Model3D, Tomar y asignar otras propiedades a nivel "nodo", como modo sombreado y Tomar y asignar el hitTestObject. Note que las pruebas iniciadas 3D explícitas no son expuestas dentro del escenario 3D. En otras palabras, no hay un API para proyectar un rayo dirigido arbitrariamente en una escena 3D para ver que prueba esto. La funcionalidad de prueba puede ser accedida ejecutando unas pruebas contra la proyección 2D en el Retained3DVisual.

Incorporar 3D propone un desafío cuando este viene a racionalizar el sistema de coordenadas 2D con el sistema de coordenadas 3D. En 2D, el sistema de coordenadas tiene el origen en el superior izquierdo, +x a la derecha, y +y abajo.

El sistema de coordenadas 3D puede ser bien manejado, donde +z sale de la pantalla, +x está a la derecha (el cual es igual que en 2D), y +y es arriba (el cual es diferente de 2D). Esto es causado por un sistema de coordenadas manual en un contexto de gráficos 3D que tienden a crear una gran

B28

cantidad de errores de programación innecesarios. Note que el hardware moderno tiene la posibilidad de bajar sombreados por vértice, así haciendo posible usar cualquier convención de sistema de coordenadas deseado.

Otro aspecto de la racionalización del sistema de coordenadas es el espacio u,v donde las texturas permanecen. No hay una convención particular por defecto para el espacio u,v , pero los modelos con "+v bajando" permitirán más mapeos directos de UI 2D a través de texturas que se modelan con "+v subiendo" (para todas las texturas parecidas mostradas al lado de abajo). En el evento donde el modelo a ser texturado no tenga la apropiada rejilla u,v orientada, la propiedad Transform en el Brushes de Material puede ser usada para contrarrestar.

Un número de experimentos ha sido llevado a cabo para ver de manera más viable maneras de tomar el texto para verlo atractivo en 3D; (note que esto no necesariamente implica "Texto 3D" con ancho, alto y profundidad). Para finalizar, una aplicación puede construir Visuals 2D teniendo el texto (y otros elementos 2D arbitrarios) a ser usado en 3D, asegurando que el Visual se haya sobre-ejemplificado (por ejemplo, por 4x-8x), y luego se use este Visual como un Material en un plano o un objeto arbitrario 3D. Si la textura

B7b

resultante es anisotrópica, el resultado es un texto atractivo.

Note que los tipos declarativos especificados en esta sección (por ejemplo, los vectores, puntos, luces, meshes, materiales, primitivas, transformadas, y demás) son todos fácilmente describibles a través de marcas basadas en XAML utilizando mecanismos de descripción de tipo de clase XAML CLR estándar. Los tipos pueden tener TypeConverters, pero si no, ellos son especificables en marcas a través de mecanismos de descripción de propiedades simples y complejas estándar que ofrece XAML. Note que las especificaciones TypeConverter son para interpretar representaciones basadas en cadenas puras de tipos como ellos aparecen en XAML. Así, por ejemplo, para un vector 3D que es utilizado en un ScaleTransform3D, la marca XAML puede ser:

```
<ScaleTransform3D scaleVector="1,2,8" />
```

Note que "ScaleTransform3D" y el "scaleVector" son analizados y entendidos por el analizador genérico XAML, pero el "1,2,8" es alimentado al TypeConverter de Vector3D y supone generar un Vector3D (puesto que es el tipo de propiedad "ScaleVector" de ScaleTransform3D). Además, note que mientras que no sea listado explícitamente por cada tipo, estos tipos tienen los

siguientes métodos (mostrado aquí para Vector3D, pero aplicable a los demás):

```
public static bool operator == (Vector3D vector1, Vector3D vector2)
public static bool Equals(Vector3D vector1, Vector3D vector2)
public static bool operator != (Vector3D vector1, Vector3D vector2)
public override bool Equals(object o)
public override int GetHashCode()
public override string ToString()
```

También, cualquier tipo que derive de Changeable (directa o indirectamente) necesitará tener el método "public new MyType Copy()" en este. Estos tipos primitivos simplemente existen en soporte de otros tipos de escritos en esta sección. Donde quiera que sea posible, ellos reflejan los tipos primitivos utilizados en 2D, y esta semejanza es una meta de diseño para estos tipos.

El objeto Point3D es un análogo directo al tipo de Punto 2D System.Windows.Point:

1
A50

```

public struct System.Windows.Media3D.Point3D
{
    public Point3D(); // initializes to 0,0,0
    public Point3D(double x, double y, double z);

    public double X { get; set; }
    public double Y { get; set; }
    public double Z { get; set; }

    public void Offset(double dx, double dy, double dz);

    public static Point3D operator +(Point3D point, Vector3D vector);
    public static Point3D operator -(Point3D point, Vector3D vector);
    public static Vector3D operator -(Point3D point1, Point3D point2);
    public static Point3D operator *(Point3D point, Matrix3D matrix);
    public static Point3D operator *(Point3D point, Transform3D transform);

    public static explicit operator Vector3D(Point3D point);

    // Explicit promotion of a 3D point to a 4D point. W coord becomes 1.
    public static explicit operator Point4D(Point3D point);
}

```

coordinate:

double-number-representation

comma-wsp:

one comma with any amount of whitespace before or after

7829

```

coordinate-triple:
    (coordinate comma-wsp){2} coordinate

point3D:
    coordinate-triple

```

Vector3D es un análogo directo al tipo Vector 2D
System.Windows.Vector:

```

public struct System.Windows.Media3D.Vector3D
{
    public Vector3D(); // initializes to 0,0,0
    public Vector3D(double x, double y, double z);

    public double X { get; set; }
    public double Y { get; set; }
    public double Z { get; set; }

    public double Length { get; }
    public double LengthSquared { get; }

    public void Normalize(); // make the Vector3D unit length

    public static Vector3D operator -(Vector3D vector);
    public static Vector3D operator +(Vector3D vector1, Vector3D vector2);
    public static Vector3D operator -(Vector3D vector1, Vector3D vector2);
    public static Point3D operator +(Vector3D vector, Point3D point);
    public static Point3D operator -(Vector3D vector, Point3D point);
    public static Vector3D operator *(Vector3D vector, double scalar);
    public static Vector3D operator *(double scalar, Vector3D vector);
    public static Vector3D operator /(Vector3D vector, double scalar);
    public static Vector3D operator *(Vector3D vector, Matrix3D matrix);
}

```



```

public static Vector3D operator *(Vector3D vector, Transform3D transform);

// return the dot product: vector1.X*vector2.X + vector1.Y*vector2.Y
public static double DotProduct(Vector3D vector1, Vector3D vector2);

// return the normal vector between the two input vectors by
// computing the cross product.
public static Vector3D CrossProduct(Vector3D vector1, Vector3D vector2);

// return the angle required to rotate v1 into v2, in degrees
// This will return a value between (-180, 180] degrees
public static double AngleBetween(Vector3D vector1, Vector3D vector2);

public static explicit operator Point3D(Vector3D vector);}

// Explicit promotion of a 3D vector to a 4D point. W coord becomes 0.
public static explicit operator Point4D(Vector3D point);

```

point3D:

coordinate-triple

Point4D agrega en una cuarta componente, w, a un punto 3D, y es usado para transformar a través de la Matriz3D no afín. No hay un Vector4D, como la componente "w" de 1 trasladado a un Point3D, y una componente "w" de 0 trasladada a un Vector3D:

```

public struct System.Windows.Media3D.Point4D
{
    public Point4D(); // initializes to 0,0,0,0
    public Point4D(double x, double y, double z, double w);

    public double X { get; set; }

```

```

    public double Y { get; set; }
    public double Z { get; set; }
    public double W { get; set; }

    public static Point4D operator -(Point4D point1, Point4D point2);
    public static Point4D operator +(Point4D point1, Point4D point2);
    public static Point4D operator *(double scalar, Point4D point);
    public static Point4D operator *(Point4D point, double scalar);
    public static Point4D operator *(Point4D point, Matrix3D matrix);
    public static Point4D operator *(Point4D point, Transform3D transform);
}

```

point4D:

coordinate-quad

Quaternions son entidades 3D definidas que representan rotación en tres dimensiones. Su poder viene de la capacidad de interpolar (y así animar) entre quaternions para alcanzar interpolación suave, confiable. El mecanismo de interpolación particular es conocido como Spherical Linear Interpolation.

Los Quaternions pueden ser contruidos de diferentes especificaciones de sus componentes (x, y, z, w), o como una representación eje/ángulo. La primera representación puede resultar en quaternions desnormalizados, para los cuales ciertas operaciones no tienen sentido (por ejemplo, extraer un eje y un ángulo).

Las componentes de un Quaternion no pueden ser asignadas hasta que el Quaternion es construido, ya que es potencialmente ambiguo el hacerlo así, por ejemplo, asignar el Angulo a un Quaternion no normalizado no tiene sentido.

```
public struct System.Windows.Media3D.Quaternion
{
    public Quaternion(); // initializes to 0,0,0,0
    ..
    // Non-normalized quaternions are allowed
    public Quaternion(double x, double y, double z, double w);
}
```

```

// allow construction through axis and angle
public Quaternion(Vector3D axisOfRotation, double angleInDegrees);

// fundamental Quaternion components
public double X { get; }
public double Y { get; }
public double Z { get; }
public double W { get; }

// axis/angle access. Will raise an exception if the quaternion
// is not normalized.
public Vector3D Axis { get; }
public double Angle { get; } // in degrees, just like everything else

// Magnitude of 1? Only normalized quaternions can be used in
// RotateTransform3D's.
public bool IsNormalized { get; }

public Quaternion Conjugate(); // return conjugate of the quaternion
public Quaternion Inverse(); // return the inverse of the quaternion
public Quaternion Normalize(); // return a normalized quaternion

public static Quaternion operator +(Quaternion left, Quaternion right);
public static Quaternion operator -(Quaternion left, Quaternion right);
public static Quaternion operator *(Quaternion left, Quaternion right);

// smoothly interpolate between two quaternions
public static Quaternion Slerp(Quaternion left, Quaternion right, double t);
}

```

quaternion:

<code>coordinate-quad </code>	<code>// x,y,z,w representation</code>
<code>(" coordinate-triple ") coordinate</code>	<code>// axis,angle representation</code>

Matrix3D es un 3D análogo a System.Windows.Matrix. Como Matriz, muchos API no toman Matrix3D, pero toman Transform3D, el cual soporta animación en forma profunda. Matrices para computaciones 3D son representadas como una matriz 4x4. El MIL utiliza la sintaxis fila-vector:

<code>m11</code>	<code>m12</code>	<code>m13</code>	<code>m14</code>
<code>m21</code>	<code>m22</code>	<code>m23</code>	<code>m24</code>
<code>m31</code>	<code>m32</code>	<code>m33</code>	<code>m34</code>
<code>offsetX</code>	<code>offsetY</code>	<code>offsetZ</code>	<code>m44</code>

Cuando una matriz es multiplicada con un punto, ésta transforma el punto de un nuevo sistema de coordenadas al sistema de coordenadas anterior.

Las transformaciones pueden ser anidadas a cualquier nivel. Siempre que una nueva transformación es aplicada esto es lo mismo que una pre-multiplicación de ésta en la matriz de transformación actual:

```
public struct System.Windows.Media.Media3D.Matrix3D
{
    // Construction and setting
    public Matrix(); // defaults to identity

    public Matrix(
        double m11, double m12, double m13, double m14,
        double m21, double m22, double m23, double m24,
        double m31, double m32, double m33, double m34,
        double offsetX, double offsetY, double offsetZ, double m44);

    // Identity
    public static Matrix3D Identity { get; }
    public void SetIdentity();
    public bool IsIdentity { get; }
```

```

// Math operations
public void Prepend(Matrix3D matrix); // "this" becomes: matrix * this
public void Append(Matrix3D matrix); // "this" becomes: this * matrix

// Rotations – Quaternion versions. If you want axis/angle rotation,
// build the quaternion out of axis/angle.
public void Rotate(Quaternion quaternion);
public void RotatePrepend(Quaternion quaternion);
public void RotateAt(Quaternion quaternion, Point3D center);
public void RotateAtPrepend(Quaternion quaternion, Point3D center);

public void Scale(Vector3D scalingVector);
public void ScalePrepend(Vector3D scalingVector);
public void ScaleAt(Vector3D scalingVector, Point3D point);
public void ScaleAtPrepend(Vector3D scalingVector, Point3D point);

public void Skew(Vector3D skewVector); // Appends a skew, in degrees
public void SkewPrepend(Vector3D skewVector);
public void SkewAt(Vector3D skewVector, Point3D point);
public void SkewAtPrepend(Vector3D skewVector, Point3D point);

public void Translate(Vector3D offset); // Appends a translation
public void TranslatePrepend(Vector3D offset); // Prepends a translation

public static Matrix3D operator * (Matrix3D matrix1, Matrix3D matrix2);

// Transformation services. Those that operate on Vector3D and Point3D
// raise an exception if IsAffine == false.
public Point3D Transform(Point3D point);
public void Transform(Point3D[] points);
public Point4D Transform(Point4D point);

```

El ImageEffect es la clase base para los efectos de formación de imágenes con base en cuadrícula. Un ImageEffect puede ser visualizado como una colección de 0 o más entradas y 0 o más salidas. Las entradas y salidas a un ImageEffect son todas del tipo ImageSource. Un ImageEffect es típicamente inicializado con sus entradas y sus propiedades y luego sus salidas son usadas sea para dibujar parte de una escena o como cuadros para un ImageEncoder. Los efectos de construcción incluida incluyen (pero no se limitan a) lo siguiente: ImageEffectBlur, ImageEffectFlipRotate, ImageEffectGammaCorrect, ImageEffectGlow, ImageEffectGrayscale, ImageEffectNegate, ImageEffectSharpen, ImageEffectTint.

TRANSFORMACIÓN

La clase Transform de objetos, representada en la figura 7, puede ser usada para puesta en escala, giro, traslación y vector de inclinación y gráficos de tramado. Las clases Transform derivadas suministran una utilización amigable y semántica de enumeración. La jerarquía de clase de transformada difiere de la estructura de matriz que es una clase y soporta animación y semánticas de enumeración:

TransformCollection (semánticas de enumeración, etc.)

TransformCollection.AddScale (...)

Handwritten signature or mark

Animate MatrixTransform

La Transform es una clase de base abstracta:

```
[TypeConverter(typeof(TransformConverter))]  
public abstract class Transform : Animatable, IDisposeable  
{  
    internal Transform();  
    public new Transform Copy();  
  
    public static MatrixTransform CreateMatrix(Matrix matrix);  
    public static TranslateTransform CreateTranslation(double x, double y);  
    public static RotateTransform CreateRotation(double angle);  
    public static RotateTransform CreateRotation(double angle, Point center);  
    public static ScaleTransform CreateScale(double scaleX, double scaleY);  
    public static ScaleTransform CreateScale(  
        double scaleX,  
        double scaleY,  
        Point center);  
    public static SkewTransform CreateSkew(double angleX, double angleY);  
    public static SkewTransform CreateSkew(  
        double angleX,  
        double angleY,  
        Point center);  
  
    // Identity transformation  
    public static Transform Identity { get; }  
  
    public abstract Matrix Value { get; }  
}
```

Una TransformCollection es una colección de objetos Transform cuyo valor es la multiplicación de la matriz de los valores Transform individuales. El valor está compuesto de izquierda a derecha, y caza el primero y el último de los elementos de la colección:

```
public sealed class TransformCollection : Transform, IAddChild, IList
{
    public TransformCollection();
    public TransformCollection(int capacity);
    public TransformCollection(params Transform[] transforms);
    public TransformCollection(ICollection transformCollection);
    public new TransformCollection Copy();

    // IEnumerable
    public IEnumerator GetEnumerator();

    // ICollection
    public int Count { get; }
    public bool IsSynchronized { get; }
```

```
public object SyncRoot();
public void CopyTo(Array array, int index);
```

```
// IList
```

```
public bool IsFixedSize { get; }
bool IList.IsReadOnly { get; }
object IList.this[int index] { get; set; }
int IList.Add(object value);
public void Clear();
bool IList.Contains(object value);
int IList.IndexOf(object item);
void IList.Insert(int index, object value);
void IList.Remove(object value);
public void RemoveAt(int index);
```

```
// Extra functions
```

```
public int Capacity { get; set; }
```

```
public Transform this[int index] { get; set; }
```

```
public int Add(Transform transform);
```

```
public bool Contains(Transform transform);
public int IndexOf(Transform transform);
public int IndexOf(Transform transform, int startIndex);
public int IndexOf(Transform transform, int startIndex, int count);
public int LastIndexOf(Transform transform);
public int LastIndexOf(Transform transform, int startIndex);
public int LastIndexOf(Transform transform, int startIndex, int count);
public void Insert(int index, Transform transform);
public void Remove(Transform transform);
public void AddRange(TransformCollection transformCollection);
public void SetRange(int index, TransformCollection transformCollection);
public void InsertRange(int index, TransformCollection transformCollection);
```

```

public void RemoveRange(int index, int count);
public TransformCollection GetRange(int index, int count);

// IAddChild
void IAddChild.AddChild(Object o);
void IAddChild.AddText(string s);

public override Matrix Value { get; }

public void AddMatrix(
    double m11,
    double m12,
    double m21,
    double m22,
    double offsetX,
    double offsetY);
public void AddMatrix(Matrix matrix);
public void AddTranslate(double translateX, double translateY);
public void AddRotate(double angle);
public void AddRotate(double angle, Point center);
public void AddScale(double scaleX, double scaleY);
public void AddScale(double scaleX, double scaleY, Point center);
public void AddSkew(double angleX, double angleY);
public void AddSkew(double angleX, double angleY, Point center);

public Transform GetOptimizedTransform();
}

```

Un RotateTransform define una rotación por un ángulo alrededor de un punto centro específico (por omisión es 0,0). El ángulo es especificado en grados. Una matriz estática de

representación para el ángulo de rotación alrededor del punto x, y es como sigue.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\text{angle}) & \sin(\text{angle}) & 0 \\ -\sin(\text{angle}) & \cos(\text{angle}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{bmatrix}$$

```

public sealed class RotateTransform : Transform
{
    public RotateTransform();
    public RotateTransform(double angle);
    public RotateTransform(
        double angle,
        Point center);
    public RotateTransform(
        double angle,
        DoubleAnimationCollection angle,
        Point center,
        PointAnimationCollection center);
    public new RotateTransform Copy();

    [Animations("AngleAnimations")]
    public double Angle { get; set; }
    public DoubleAnimationCollection AngleAnimations { get; set; }

    [Animations("CenterAnimations")]
    public Point Center { get; set; }
    public PointAnimationCollection CenterAnimations { get; set; }

    public override Matrix Value { get; }
}

```

La TranslateTransform define un eje de translación alineada en la dirección x y y. La representación de la matriz estática para la translación por desplazamiento dx, dy es como sigue.

```
public sealed class System.Windows.Media.TranslateTransform : Transform
{
    public TranslateTransform();
    public TranslateTransform(
        double offsetx,
        double offsety);
    public TranslateTransform(
        double offsetx, DoubleAnimationCollection offsetx,
        double offsety, DoubleAnimationCollection offsety);
    public new TranslateTransform Copy();

    [Animations("XAnimations")]
    public double X { get; set; }
    public DoubleAnimationCollection XAnimations { get; set; }

    [Animations("YAnimations")]
    public double Y { get; set; }
    public DoubleAnimationCollection YAnimations { get; set; }

    public override Matrix Value { get; }
}
```

La ScaleTransform define una escala en la dirección x y y alrededor de un punto centro (por defecto es 0,0). La

representación de la matriz estática para la escala s_x , s_y alrededor del punto x , y es cómo sigue.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x & -y & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x & y & 1 \end{bmatrix}$$

```

public sealed class ScaleTransform : Transform
{
    public ScaleTransform();
    public ScaleTransform(
        double scaleX,
        double scaleY);
    public ScaleTransform(
        double scaleX,
        double scaleY,
        Point center);
    public ScaleTransform(
        double scaleX,
        DoubleAnimationCollection scaleXAnimations,
        double scaleY,
        DoubleAnimationCollection scaleYAnimations,
        Point center,
        PointAnimationCollection center);
    public new ScaleTransform Copy();

    [Animations("XAnimations")]
    public double ScaleX { get; set; }
    public DoubleAnimationCollection ScaleXAnimations { get; set; }

    [Animations("YAnimations")]
    public double ScaleY { get; set; }
    public DoubleAnimationCollection ScaleYAnimations { get; set; }

    [Animations("CenterAnimations")]

```

```

public Point Center { get; set; }
public PointAnimationCollection CenterAnimations { get; set; }

public override Matrix Value { get; }
}

```

El SkewTransform define una inclinación por un grado de ángulo a lo largo de la dirección x y y. el ángulo de inclinación está en grados. La representación de la matriz estática para la inclinación por el ángulo es como sigue.

$$\begin{bmatrix} 1 & \tan(\text{angleY}) & 0 \\ \tan(\text{angleX}) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```

public sealed class SkewTransform : Transform
{
    public SkewTransform();
    public SkewTransform(
        double angleX,
        double angleY);
    public SkewTransform(
        double angleX,
        double angleY,
        Point center);
    public SkewTransform(
        double angleX,
        DoubleAnimationCollection angleXAnimations,
        double angleY,
        DoubleAnimationCollection angleYAnimations,
        Point center,
        PointAnimationCollection pointAnimations);
    public new SkewTransform Copy();
}

```



```

[Animations("AngleXAnimations")]
public double AngleX { get; set; }
public DoubleAnimationCollection AngleXAnimations { get; set; }

[Animations("AngleYAnimations")]
public double AngleY { get; set; }
public DoubleAnimationCollection AngleYAnimations { get; set; }

[Animations("CenterAnimations")]
public Point Center { get; set; }
public PointAnimationCollection CenterAnimations { get; set; }

public override Matrix Value { get; }
}

```

La MatrixTransform define una transformación por medio de su representación matemática:

```

public sealed class MatrixTransform : Transform
{
    public MatrixTransform();
    public MatrixTransform(
        double m11,
        double m12,
        double m21,
        double m22,
        double offsetX,
        double offsetY);
    public MatrixTransform(Matrix matrix);
    public new MatrixTransform Copy();

    public Matrix Matrix { get; set; }
}

```

```
public override Matrix Value { get; }
}
```

Cuando una propiedad tipo Transform es especificada en una marcación, el sistema de propiedad usa el convertidor tipo Transform para convertir la representación de la expresión en el objeto derivado Transform apropiado. No hay actualmente manera de describir propiedades animadas usando esta sintaxis.

La sintaxis es gráficos de vectores y la construcción Transform correspondientes se resume como sigue, en donde los parámetros marcados por "<" representan parámetros opcionales:

- Matriz (m11 m12 m21 m22 offsetX offsetY)
- ° AddMatrix(m11 m12 m21 m22 offsetX offsetY)
- translate (tx<ty>)
- ° AddTranslate (tx, ty)
- ° Si ty no esta especificado se asume que es 0.
- scale (sx <sy>)
- ° AddScale (sx, sy)

° Si sy no esta especificado se asume que es igual a sx.

• rotate (angle <cx> <cy>)

° AddRotate (angle, point (cx, cy))

° Si cx, cy no esta especificado se asume que es 0,0.

• skewX (angle)

° AddSkew (angle,0)

• skewY (angle)

° AddSkew (0, angle)

transform-list:

wsp* transforms? wsp*

transforms:

transform

| transform comma-wsp+ transforms

transform:

matrix

| translate

| scale

| rotate

| skewX

| skewY

matrix:

"matrix" wsp* "(" wsp*

number comma-wsp

number comma-wsp

number comma-wsp

number comma-wsp

number comma-wsp

number wsp* ")"

translate:

"translate" wsp* "(" wsp* number (comma-wsp number)? wsp* ")"

scale:

"scale" wsp* "(" wsp* number (comma-wsp number)? wsp* ")"

rotate:

"rotate" wsp* "(" wsp* number (comma-wsp number comma-wsp number)? wsp*
")"

skewX:

"skewX" wsp* "(" wsp* number wsp* ")"

APL

skewY:

"skewY" wsp* "(" wsp* number wsp* ")"

EFFECTOS

Los efectos suministran un medio para alterar los contenidos visuales de una escena de una manera de conversión céntrica. Por ejemplo, ImageEffects (efectos bitmap con base en cuadrículas) operan sobre la representación compuesta completamente con base en imagen de una porción de una escena. Los efectos son quebrados en diversos tipos incluyendo ImageEffects, BlendModes y VectorEffects.

Los ImageEffects pueden ser usados en una escena en el modo retenido por medio de aplicarlos a un subgráfico o un elemento, o este puede ser usado en una tubería de imagen sola. En general, el ImageEffect tiene 0 o más entradas y 0 o más salidas, que son del tipo ImageSource. En la tubería de imagen de modo inmediato, no necesita haber una salida, como el ImageEffect puede estar en la superficie de otras propiedades que describen atributos de su entrada. Por ejemplo, un ImageEffect puede emitir una información de histograma de color o una información de detección de cara. En la escena de modo retenido, hay una entrada meta adicional

762

que suministra acceso a los contenidos convertidos del subgráfico al cual el efecto es aplicado.

/// La clase ImageEffect es la clase base para todos los efectos de formación de imágenes (manchado, escala de grises, etc.)

///

/// Es posible para un efecto no tener una entrada pero un efecto debe siempre tener por lo menos una salida. Las implementaciones por defecto de cosas asumen esto. Si un efecto derivado

/// va a ser usado con Output/Outputs asegúrese de que por lo menos uno este allí. El sistema de clase abstracta pública. Windows.Media.ImageEffect: Changeable

{

/// Este constructor permite que una clase derivada especifique el numero de entradas y

/// Salidas para este efecto, y esta clase entonces maneja las matrices de entrada y salida

/// Incluyendo validación de índices.

/// Estos valores por defecto a uno y uno. Si el efecto desea tener un número

/// Variable de entradas o salidas, este puede pasar de -1 para cada una (o ambas) cuentas,

/// Si las colecciones de entrada y salida permiten esto.

/// Finalmente, estos métodos son todos virtuales, así las clases derivadas pueden escoger no

/// Delegar de nuevo a la clase base, en cuyo caso no hay extra costo incurrido. De ImageEffect protegidos (int inputCount, int outputCount);

/// Este constructor permite que una clase derivada especifique el número de entradas y

/// Salidas para este efecto, y esta clase manejará entonces las matrices de entrada y salida

/// Incluyendo las marcas de validación.

/// Estos valores por omisión en uno y uno. Si el efecto lo desea tener o desea tener un número

/// Variable de entradas o salidas, este puede pasar -1 para cada uno (o ambos) cuentas,

/// Y las colecciones de entrada y salida lo permitirán.

/// Finalmente, estos métodos son todos virtuales, de modo que las clases derivadas pueden escoger no

/// Delegar de nuevo a la clase base, en cuyo caso no hay extra costo incurrido.

ImageEffect protegido (int inputCount, outputCount, double scaleX, double scaleY);

/// Esta es la primera entrada, y esta un alias para las entradas [0]

///

/// Advertencia de desempeño:

/// Si la entrada de los efectos no esta en un formato que

/// Los efectos soporte el efecto se convertirá la

/// Entrada en un formato trabajable para usted.

/// Public virtual ImageSource Input {get; set;}

/// Esta es una colección de entradas

Public virtual ImageSourceCollection input {get;}

/// Esta es una primera salida, y es un alias para las salidas [0]

Public virtual ImageSource Output {get;}


```
/// Esta es una colección de salidas

Public virtual ImageEffectSourceCollection Output {get;}

Protected internal virtual ImageSource GetOutput (int
outputIndex);

/// Estos valores contienen la escala horizontal y vertical
aplicada a esta fuente.

/// Hay ocasiones cuando un efecto requiere operar a una
resolución diferente

/// O a un espacio de coordenadas diferentes que al actual,
espacio de coordenadas lógico.

/// Así, estas propiedades permiten al consumidor mapiar
entre un espacio local y

/// ImageEffectSource space.

Protected internal virtual double GetScaleX (int
outputIndex);

Protected internal virtual double GetScaleY (int
outputIndex);
```

```
Protected internal virtual PixelFormat GetFormat (int  
outputIndex);
```

```
/// Ancho, en píxeles, de la imagen.
```

```
Protected internal virtual GetPixelWidth (int outputIndex);
```

```
/// Altura, en píxeles, de la imagen.
```

```
Protected internal virtual GetPixelHeight (int outputIndex);
```

```
/// DPI horizontal de la imagen.
```

```
Protected internal virtual double GetDpiX (int outputIndex);
```

```
///DPI vertical de la imagen
```

```
Protected internal virtual double GetDPI (int outputIndex);
```

```
/// Obtenga una paleta para una salida particular
```

```
Protected internal virtual ImagePalett GetPalette (int  
outputIndex);
```

```
/// Implementación del método abstracto de ImageSource
```

///

/// Realmente no hace nada. Si usted no quiere iniciarlo en (0,0)

/// En la entrada, entonces sourceRect se inicia en el punto que usted quiere.

```
Protected internal void Copy (int outputIndex, INtegerRect
sourceRect, PixelFormat pixelformat, byte [] pixeles, int
stride, int pixelOffset);
    }
}
```

BlendModes son una forma específica de efectos con base en imagen. Estos pueden ser aplicados a la escena de modo retenido generalmente de la misma manera que los ImageEffects. Así hay una propiedad Element ("BlendMode") lo mismo que una propiedad BlendMode sobre Visual, un método PushBlendMode sobre visual, un método PushBlendMode sobre IDrawingContext y una propiedad BlendMode sobre Brush. Los modos Blend llevan a cabo una combinación de los colores fuente y destino, así como la fuente es compuesta. Ejemplos de modos Blend influyen Multiply, Add, y así sucesivamente. Los VectorEffects son otro tipo de efectos.

Como se describió en las generalidades, los BlendModes describen una operación que controla la manera en la cual una imagen esta compuesta dentro de otra imagen o dentro de una escena.

Los BlendModes pueden ser aplicados a la escena y a las Brushes. Cada BlendMode describe una manera de combinar un píxel fuente y un píxel destino, y es aplicado a cada píxel están siendo compuestos. Los BlendModes son aplicados después de que una fuente es puesta en escala o de otra manera transformada, y después de que cualesquier efectos son aplicados (incluyendo Opacidad). Note que cuando las operaciones BlendMode son aplicadas, la fuente y destino están en formato alfa pre multiplicado. Para especificar un BlendMode, el programador puede usar uno de los BlendModes especificados en la clase estática BlendModes, o puede establecer explícitamente los multiplicadores de fuente y destino. Debido a que en una implementación los multiplicadores no son extensibles y no tienen parámetros, se representan por un enumerador:

```
/// The four values associated with each BlendModeMultiplier are  
/// multiplied by the corresponding channel in the color to which  
/// the BlendModeMultiplier is applied.  
/// The factors are multiplied by the Alpha, Red, Green and Blue  
/// channels, respectively  
/// where in-gamut values range from [0..1].  
public enum BlendModeMultipliers  
{
```

```

/// Zero - (0,0,0,0)
Zero,
/// One - (1,1,1,1)
One,
/// SourceColor - (Alphasource, Redsource, Greensource, Bluesource)
SourceColor,
/// InverseSourceColor - (1-Alphasource, 1-Redsource, 1-Greensource,
/// 1-Bluesource)
InverseSourceColor,
/// SourceAlpha - (Alphasource, Alphasource, Alphasource, Alphasource)
SourceAlpha,
/// InverseSourceAlpha - (1-Alphasource, 1-Alphasource, 1-Alphasource,
/// 1-Alphasource)
InverseSourceAlpha,
/// DestinationColor - (Alphadestination, Reddestination, Greendestination,
/// Bluedestination)
DestinationColor,
/// InverseDestinationColor -
/// (1-Alphadestination, 1-Reddestination, 1-Greendestination,
/// 1-Bluedestination)
InverseDestinationColor,
/// DestinationAlpha - (Alphadestination, Alphadestination,
/// Alphadestination, Alphadestination)
DestinationAlpha,
/// InverseDestinationAlpha -
/// (1-Alphadestination, 1-Alphadestination, 1-Alphadestination,
/// 1-Alphadestination)
InverseDestinationAlpha,
/// SourceAlphaSaturate - f = min(Alphasource, 1 - Alphadestination),
/// (1, f, f, f)
SourceAlphaSaturate
}

```

```

/// This class represents a blend operation between a source and
/// destination surface
public sealed class BlendMode: Changeable
{
    // Constructs a custom BlendMode from the provided
    // BlendModeMultipliers
    public BlendMode(BlendModeMultiplier sourceMultiplier,
        BlendModeMultiplier destinationMultiplier);

    public new BlendMode Copy();

    public BlendModeMultiplier SourceMultiplier { get; }
    public BlendModeMultiplier DestinationMultiplier { get; }
}

/// This is a collection of well-known BlendMode instances
public sealed class BlendModes
{
    /// Normal – Source is One, Destination is InverseSourceAlpha
    public static Normal {get;}

    /// SourceCopy – Source is One, Destination is Zero
    public static SourceCopy {get;}
}

```

ENSAYO DE GOLPE

El ensayo de golpe es usado para escoger visuales en una escena. Algunos escenarios de alto nivel incluyen la selección lazo y la selección de banda de caucho, la navegación por teclado (usada para encontrar el siguiente elemento para cambiar el foco), determinar el foco del ratón en el árbol de elementos, seleccionar elementos que se sobreponen con transparencia (tal como imágenes), el ensayo de golpe "de burbuja pensante" y seleccionar un golpe de carácter en el texto.

En general, el ensayo por golpe suministra consistencia a través del núcleo, la estructura de trabajo y los controles, y opera iniciando desde la parte superior del árbol de control, y regresando un control o un conjunto de controles por medio de un punto o geometría. Un control puede definir si ha sido golpeado o no con servicios de soporte incluyendo geometría convertida, caja de límites, geometría de fuera de banda (región de golpe) opacidad de la imagen o máscara, y su propia lógica. El control puede regresar un dato relacionado con un golpe específico sobre el golpe (por ejemplo, línea, posición de carácter, etc.).

El mecanismo del ensayo de golpe puede filtrar los resultados de ensayo de golpe de una manera eficiente.

Adicionalmente, el mecanismo de ensayo de golpe suministra flexibilidad para extender a otros tipos de visuales y resolver en sub primitivos dentro del Visual, por ejemplo un Retained3DVisual es un ejemplo de esta.

El caminado del ensayo de golpe es una derecha profunda o una izquierda profunda sobre el árbol visual. Hay tres participantes, el probador de golpe, el caminador, y el control barra visual. El probador de golpe implementa dos llamadas, una para manejar el caminador, y la otra para finalizar el caminado de manera temprana sobre ciertos visuales de golpe. El control implementa un método virtual para definir que es golpe. El caminador es una parte fijada del sistema y camina sobre el árbol visual con base en el comportamiento de llamadas, esencialmente preguntando a cada control si el control ha sido un golpe. Los golpes son reportados por medio de las llamadas en orden z, y de manera superior a inferior.

Internamente, el ensayo de hit comprende entonces una caminada sobre el árbol visual. Cuando desciende, el probador de hit visualiza el filtrado en términos de relaciones del nivel de elemento, por ejemplo, un cuadro con formas, un panel con un cuadro interior. Cuando un hit ocurre, el

probador de hit puede o continuar procesando hits adicionales (si hay alguno) o detenerse.

La lógica de flujo de control de esta perspectiva de caminador de golpes es el siguiente pseudo código:

- For each visual, starting at root:
 - If hit point inside coalesced children hit bounds then (eg, `HitTestBounds.Contains(HitPoint)`)
 - Call hit tester delegate (e.g., `HitTestFilterDelegate`)
 - If null or returns `Continue` then (e.g., `HitTestFilterBehavior.Continue`)
 - For each child visual
 - Transform hit point to its local space
 - Hit test against child visual (eg, `Visual.HitTest(...)`)
 - If child returns `Stop` then return (e.g., `HitTestResultBehavior.Stop`)
 - If hit point inside visual (eg, `Visual.HitTestCore(...)`)
 - Call hit tester result delegate (`HitTestResultDelegate`)
 - Either continue or stop (e.g., `HitTestResultBehavior.Continue`)

El probador de golpe usa métodos públicos para iniciar el ensayo de golpe y suministrar delegados para controlar el comportamiento. El comportamiento por defecto es para probar contra todos los visuales y regresar sobre el primer golpe. Si no resulta un delegado dado, una excepción es lanzada.

El control decide su lógica de ensayo de golpe por medio pasando por el encima del HitTextCore para punto y geometría. Cuando un ensayo de golpe es iniciado, el caminador de árbol visual interno llama el HitTestCore, en efecto preguntando al control si es un golpe. El HitTestBounds refleja límites estrechos de la región de golpe y es usado para utilizar el caminador. El comportamiento visual por defecto es para ensayar contra la conversión de la caja de límites de contenido. Los límites del golpe por defecto es para convertir el contenido de la caja de límites.

```
public class Visual : DependencyObject, IDisposable, IVisual
{
    // Ignore non hit testing stuff

    // HitTestCore is implemented by visual author to hit against its
    // content only.
    protected virtual PointHitTestResult HitTestCore(
        PointHitTestParameters point);
    protected virtual GeometryHitTestResult HitTestCore(
        GeometryHitTestParameters geometry);

    protected virtual Rect HitTestBounds { get; }
```

```
}

public delegate HitTestFilterBehavior HitTestFilterDelegate(Visual visual);

public delegate HitTestResultBehavior HitTestResultDelegate(HitTestResult result)

public interface IVisual
{
    // Ignore non hit testing stuff

    // Return top most visual hit
    public PointHitTestResult HitTest(Point point);

    // HitTest is called to initiates a hit test against a visual tree.
    void HitTest(
        HitTestFilterDelegate filterHitDelegate,
        HitTestResultDelegate resultHitDelegate,
        HitTestParameters hitTestParams);
}
```

El probador de golpe inicia el ensayo de golpe pasando por un punto de golpe o geometría de golpe y parámetros adicionales del HitTestParameters. La clase esta provista principalmente simplificar el diseño y permitir la extensibilidad del mismo. Las solicitudes de ensayo de golpe especiales pueden derivarse de esta clase para pasar información adicional a controles interesados. Cada control implementa un HitTestCore específico contra un punto y una geometría. Se espera que los controles respeten en los parámetros de ensayo de golpe cuando implementan su lógica HitTestCore.

```
public abstract class HitTestParameters
{
    // Common hit testing parameters
    internal HitTestParameters();
}
```

```
public class PointHitTestParameters : HitTestParameters
{
    public PointHitTestParameters(Point point);

    Point HitPoint { get; }
}

// Returns detailed intersection information between
// two geometry objects.
enum IntersectionDetail
{
    // Intersection detail is empty or not requested.
    EmptyOrNotCalculated,

    // First geometry is fully inside second geometry.
    FullyInside,

    // Second geometry fully contains the first geometry.
    FullyContains,

    // First and second geometry edges intersect.
    Intersects
}

public class GeometryHitTestParameters : HitTestParameters
{
    Public GeometryHitTestParameters(
        Geometry geometry,
        bool computeIntersectionDetail);

    public Geometry HitGeometry { get; }
```

```

public bool ComputeIntersectionDetail { get; }
}

```

El control regresa un dato específico por medio de derivar del `HitTestResult`. Por ejemplo, un control de texto puede querer regresar el golpe de la posición del carácter. El `PointHitTestResult` contiene el punto en el espacio de la coordenada local. El `GeometryHitTestResult` contiene la geometría de espacio de coordenada local (del ensayo de golpe original). Las funciones de la transformación visual pueden mapear la localización del golpe aun en un espacio ancestral.

```

public abstract class HitTestResult
{
    internal HitTestResult(Visual visual);

    public Visual Visual { get; }
}

public class PointHitTestResult : HitTestResult
{
    public PointHitTestResult(Visual visual, Point point);

    public Point Point { get; }
}

public class GeometryHitTestResult : HitTestResult

{
    public GeometryHitTestResult(
        Visual visual,
        Geometry geometry,
        IntersectionDetail intersectionDetail);

    public Geometry Geometry { get; }
}

```

```

// This is only set if the GeometryHitTestParameters.IntersectionDetail
public IntersectionDetail IntersectionDetail { get; }
}

```

Para ilustrar el uso de delegados, considere un probador de golpe que quiere el primer golpe más superior usando delegados anónimos:

```

public Visual HitTest(Visual visual, Point pt)
{
    Visual visualHit = null;

    visual.HitTest(
        new PointHitTestParameters(pt),
        null,
        HitTestResultDelegate(HitTestResult htr) {
            visualHit = htr.Visual;
            return HitTestResultBehavior.Stop;
        }
    );
    return visualHit;
}

```

Otro ejemplo es un probador de golpe que quiere regresar todos los visuales que fueron golpeados:



```

public Visual[] HitTest(Visual visual, Point pt)
{
    ArrayList visualsHit = new ArrayList();

    visual.HitTest(
        new PointHitTestParameters(pt),
        null,

```

```

        HitTestResultDelegate(HitTestResult htr) {
            visualsHit.Add(htr.Visual);
            return HitTestResultBehavior.Continue;
        }
    };
}

```

El probador de golpe usa enumeradores para controlar el filtrado del ensayo de golpe y el comportamiento resultante:

```

public enum HitTestFilterBehavior
{
    //
    ContinueSkipChildren,
    ContinueSkipVisualAndChildren,
    ContinueSkipVisual,
    Continue,
    Stop
}

```

El enumerador HitTestFilerBehavior controla el comportamiento de filtrado en que se especifica los ensayos de golpe

SkipChildren en esta visual, pero no sus visuales hijos. El SkipVisualAndChildren no especifica el ensayar por golpe los visuales hijos. SkipVisual especifica no hacer ensayo de golpe de la visual, pero ensayar de golpe cualesquier visuales de hijos. Continue especifica ensayar de golpe esta visual y sus visuales hijos. Stop especifica no ensayar de golpe ninguna visual en el árbol visual y regresar al llamador.

El enumerador HitTestResultBehavior controla el comportamiento del ensayo de golpe:

```
public enum HitTestResultBehavior
{
    Stop,
    Continue ...
}
```

Stop especifica regresar a la entrada de ensayo de golpe, pasando por alto cualesquier filtro adicional u operaciones de ensayo de golpe. Continue especifica ensayar de golpe contra la siguiente visual.

Aunque los identificadores de ensayo de golpe pueden ser usados para marcar el contenido especifico para una

identificación de golpe positiva, el desempeño fue pobre por que tal modelo rompe la corriente de conversión, agregado a un encabezamiento de caminado y fue difícil manejar cuando ensayar por golpe. Con la combinación de elementos y visuales en un tipo unificado, el nivel básico de granularidad es la visual misma, y los controles pueden estructurarse ellos mismos para obtener el nivel de granularidad que se desea.

El autor del control escribe la lógica para el golpe por medio de pasar por alto el HitTestCore y hace su propia computación y/o usa los servicios descritos más adelante.

Los siguientes son algunos ejemplos que demuestran la potencia de estos servicios.

Un primer ejemplo demuestra un control que tiene una propiedad pública HitRegion que representa la región sensible a golpe del control. Note que la región de golpe no requiere cazar con los contenidos convertidos, y puede ser optimizada por algunas aplicaciones. Si la región de golpe no esta establecida (`_hitRegio=null`) el control difiere a los servicios de implementación de base para determinar un golpe.

```
public class HitRegionControl : Control           // derives from visual.
{
    private Geometry _hitRegion;

    public Geometry HitRegion
    {
        get
        {
            return _hitRegion;
        }
        set
        {
            _hitRegion = value;
        }
    }
}
```

```

protected virtual PointHitTestResult HitTestCore(PointHitTestParameters
    htParams)
{
    bool IsHit = (_hitRegion != null) ?
        _hitRegion.DoesContain(htParams.Point) : IsHitRenderContents(htParams);

    return IsHit ? new PointHitTestResult(this, htParams.Point) : null;
}

protected virtual GeometryHitTestResult HitTestCore(
    GeometryHitTestParameters htParams)
{
    IntersectionDetail intersectDetail = (_hitRegion != null) ?
        _hitRegion.DoesContain(
            htParams.Geometry,
            htParams.ComputeIntersectionDetail) : HitTestRenderContents(htParams);

    return (intersectDetail != IntersectionDetail.Empty) ?
        new GeometryHitTestResult(
            this,
            htParams.Geometry,
            intersectDetail) : null;
}

protected virtual Rect HitTestBounds
{
    get
    {
        return (_hitRegion != null) ? _hitRegion.Bounds : GetContentBoundingBox()
    }
}
}

```

Para sobrepasar el comportamiento IsHit use servicios de soporte adicionales.

Las clases de Geometry llevan a cabo un ensayo de golpe contra su área interior:

```
public abstract class Geometry : Changeable
{
    public virtual bool DoesContain(Point point);

    public virtual bool DoesContain(Geometry geometry);
    public virtual IntersectionDetail DoesContainWithDetail(Geometry geometry);
}
```

La visual suministra funciones protegidas al ensayo de golpe contra los contenidos convertidos (o así mismos). Si una visual es retenida, esta dispara una validación de contenido. Este ayudador o auxiliar examina la corriente de instrucción de dibujo almacenada sobre la visual, una instrucción a cada momento, ensayando por golpe el punto o geometría para cada uno con la geometría convertida.

```
public class Visual : DependencyObject, IDisposable, IVisual
{
    protected Rect VisualDescendantBounds { get; }
    protected Rect VisualContentBounds { get; }
}
```

El código regresara si el píxel de imagen en el punto esta por encima del umbral alfa. El punto esta en espacio visual y la transformación es para visualizar espacio en donde el ensayo con base en píxel ocurre.

Public class ImageData: ImageSource

```
{
    public virtual bool HitTestImageContents(
        HitTestParameters htParams,
        Transform transform);
}
```

ANIMACIÓN

El sistema de animación esta compuesto de dos conjuntos principales de componentes, particularmente, un motor de control de temporización y un conjunto de objetos de animación. El motor de temporización es un servicio que puede ser usado por cualesquier objetos que exhiban comportamientos variables en el tiempo, en los ejemplos principales son animaciones y audio u objetos de medios de video. Los objetote de animación implementan un conjunto de funciones que mapean espacios de tiempo en otros tipos de datos, que

son entonces usados como entradas en otros objetos de mayor nivel.

La animación gráfica es lograda por medio de asociar una colección de animación con una operación de conversión. Por ejemplo, el método `IdrawingContex.DrawingLine`, un lápiz y dos puntos de extremos. Uno de los puntos de extremo puede estar asociado con una colección de objetos `PointAnimation`, en cuyo caso la línea se moverá durante el tiempo. Similarmente, el lápiz puede tener una colección asociada de objetos `ColorAnimation`. En tales casos, cada animación usada en una operación de conversión puede ser corrida sobre un reloj separado, algunas veces llamado una "línea de tiempo". Una vez que un primitivo animado es dibujado, el sistema de conversión asume el redibujo de la escena a intervalos regulares. Cada vez un cuadro es convertido al valor actual de las animaciones involucradas en la escena es computado, con base en el tiempo que ha transcurrido (en la mayoría de los casos medido por el reloj del sistema), y cuando los primitivos animados son redibujados.

La programación de animaciones requiere un entendimiento tanto de los objetos de animación suministrados por el sistema lo mismo que del motor de temporización que maneja esas animaciones. Los siguientes términos son usados en diversos lugares de esta sección.

Un modelo Timming en el cual objetos temporizados participan en un sistema de temporización jerárquica en donde las líneas de tiempo individuales tienen atributos que definen su comportamiento con relación a su línea de tiempo pariente o, para líneas de tiempo de nivel superior, en relación a un "documento" raíz (o "página" o "cuadro") en la línea de tiempo. Los atributos de Timming son un conjunto de parámetros que definen el comportamiento de tiempo de un objeto. Los atributos Timming son descriptivos exclusivamente y no tienen estado de corrida en tiempo. En adición, los atributos de temporización son inmutables.

Una línea de tiempo es un caso de una entidad de temporización que mantiene un estado de corrida en tiempo de acuerdo con un conjunto de atributos de temporización. Una línea de tiempo define el concepto de "ahora" para un objeto temporizado. Un árbol de temporización es una estructura de datos que contiene un conjunto de líneas de tiempo dispuestas de una manera jerárquica. La relación entre las líneas de tiempo esta definida por un conjunto de reglas de herencia y por atributos de temporización asociados con cada línea de tiempo.

Un objeto temporizado es un objeto que exhibe un comportamiento variable en el tiempo. La descripción de

comportamiento de tiempo de un objeto temporizado es especificada por un conjunto de atributos de temporización mientras que su estado de temporización en tiempo de corrida es mantenido por una o más líneas de tiempo. Una función de animación es una función que toma un valor base de un tipo de dato particular como una entrada y produce un valor del mismo tipo como su salida. Una función de animación puede o no tomar otros parámetros de entrada implícitos o explícitos, tales como el valor de tiempo actual de una TimeLine. A esto respecto, una función de animación puede no ser constante, en que la misma entrada puede producir diferentes salidas a diferentes tiempos.

Un modificador es un objeto que implementa una función de animación y es usado para modificar el valor de una propiedad de un elemento, algún otro objeto complejo, o un parámetro para una llamada de conversión. Un modificador temporizado es un modificador que esta asociado con una TimeLine, y cuya función de animación depende explícitamente del estado de tiempo de corrida de aquella TimeLine. Una animación es un modificador temporizado que implementa un cierto conjunto conocido de funciones de animación.

Una colección de animación es una colección de modificadores que procesa el mismo tipo de datos. Una colección de animación encadena la salida de un modificador a la entrada

de otro, creando una tubería de modificación. Ya que la colección completa toma una entrada y produce una salida, la colección misma se comporta como un modificador. Las líneas de tiempo pueden ser pensadas como cronómetros que controlan los procesos variables en el tiempo, de modo que la producción de un corto de video o de una animación. Los tiempos que son especificados en los atributos de una línea de tiempo están relacionados con algo. En la mayoría de los casos ellos están relacionados con una línea de tiempo pariente, pero para las líneas de tiempo en la raíz del árbol los valores están relacionados a "documentos de tiempo" en donde el tiempo del documento es una línea de tiempo implícita que inicia cuando la aplicación es lanzada o cuando una página o cuadro está siendo navegada. El reloj en una línea de tiempo esta expuesto en dos maneras: como un desplazamiento del punto de inicio o como una proporción de progreso entre 0 y 1. Lo último es simplemente la proporción del tiempo real a la duración.

La línea de tiempo más simple tiene un tiempo de inicio y una duración. Por ejemplo, una línea de tiempo con un tiempo de inicio de 3 segundos y una duración de 5 segundos "inicia" 3 segundos después del tiempo de referencia $t=0$ (por defecto, el momento en que la aplicación es cargada), y "termina" 5 segundos después, durante estos 5 segundos la línea de tiempo se dice que esta "on" si esta línea de tiempo controla una

animación, esta animación es cambiada (por ejemplo se mueve) durante este tiempo pero está estática antes y después. La figura 29 muestra una línea de tiempo con un tiempo de inicio de tres y una duración de 5.

Una línea de tiempo también puede ser programada para repetir su comportamiento. Esta repetición puede ser especificada como un conteo de hiperacción, o una duración de repetición. La línea de tiempo pasa a través de muchas corridas de inicio a fin como sea necesario para llenar la cuenta o duración solicitada. Si la cuenta repetida no es un valor entero, la última interacción es interrumpida en la mitad. La figura 30 muestra una línea de tiempo con Begin=3, Duration=5 y RepeatDuration=17 (significando que la animación se repetirá cada 5 segundos, hasta que se completan 17 segundos después del tiempo de inicio o 20 segundos).

El tiempo de inicio para una línea de tiempo es relativo normalmente a su línea de tiempo pariente (o al tiempo del documento), pero un tiempo de inicio también puede ser especificado con relación a otro inicio o final de una línea de tiempo. En tal situación cada inicio (o fin) en la línea de tiempo fuente causa un inicio correspondiente que es programado para la línea de tiempo objetivo. La figura 31 muestra una línea con un tiempo de inicio 3 segundos después de otra línea de tiempo.

Cuando una línea de tiempo alcanza el punto final es inmediatamente cambiada a "off". Este tiempo, el objeto temporizado que esta controla cesa de tener un efecto sobre la presentación. Por ejemplo, si el objeto temporizado es una animación entonces cuando la línea de tiempo de control alcanza el punto final la animación es removida, es decir se revierte de nuevo a su valor base. Ha casos, sin embargo, en donde es deseable tener el estado estable final de una animación por medio de congelar el último valor. En otras palabras, la línea de tiempo progresa desde 0 a 1 entre los puntos de inicio y final, pero después del punto final permanece "on" con un progreso de 1. Esto es llamado un comportamiento de "llenado". La figura 32 representa una línea de tiempo con Begin=3, Duration=5 y Fill= freeze.

El tiempo fluye linealmente desde un valor de progreso de 0 hasta un valor de progreso de 1, desde el punto de vista de una línea de tiempo. Sin embargo, entre el pasaje del tiempo al interior de una línea de tiempo y al interior de su pariente puede ser alterado desde la correlación directa por defecto, en que el tiempo puede ser relanzado en una línea de tiempo de modo que este parece fluir hacia atrás, la velocidad a la cual el tiempo pasa puede ser acelerada o desacelerada por un factor multiplicativo, y/o en la curva de progreso puede ser morfeado de modo que en lugar de

progresar linealmente de 0 a 1, es que acelera desde un estado quieto al punto de inicio a una velocidad máxima de progreso, y entonces desacelera hacia un estado quieto en el punto final esto produce un efecto "ease-in, ease-out" para cualesquiera animaciones controladas por esta línea de tiempo.

Más particularmente, la curve progreso-tiempo es lineal por defecto. Cuando esta curva lineal es usada para controlar ciertas animaciones, el usuario percibe un efecto "sacudidas" en los puntos de inicio y finales porque la animación inicia y finaliza súbitamente. Para estos casos, una línea de tiempo puede ser programada para acelerar el pasaje del tiempo desde un estado quieto a una velocidad máxima usando una curva de aceleración suave. Similarmente, el tiempo puede ser programado para desacelerar hacia 0 cerca del punto final. Los efectos de aceleración y desaceleración son especificados como el porcentaje de la duración gastada en las fases de aceleración o desaceleración. Los dos valores son positivos y su suma no debe exceder uno. La figura 33 muestra una línea de tiempo con `Begin=3`, `Duration=10`, `Accleration=0,2` y `Deceleration=0,4`.

Una manipulación directa del tiempo involucra programar una línea de tiempo que vaya desde un valor de progreso de 0 a 1 y luego de nuevo a 0. En este caso, la línea de tiempo esta

305

activa durante el doble de la duración especificada, una vez para la parte "hacia delante" y una vez para la parte "hacia atrás". La figura 34 muestra una línea de tiempo con `Begin=3`, `Duration=5` y `AutoReverse=True`.

El pasaje aparente del tiempo por una línea de tiempo puede ser más rápido o más lento que aquella para su pariente, por un factor constante. Por defecto este factor es 1, lo que significa que el tiempo en una línea de tiempo y su pariente pasa a la misma velocidad. Si en su lugar este valor es mayor de uno, entonces el tiempo para la línea de tiempo progresa a una velocidad mayor o más rápida que la de su pariente. Por ejemplo, un factor de tres hace que la línea de tiempo vaya entre los puntos de inicio y final tres veces más rápido que la duración especificada. De manera alterna, si el factor está entre 0 y 1 entonces el tiempo pasa a una velocidad más lenta. Si el factor es negativo el tiempo en la línea de tiempo siempre parece moverse hacia atrás con relación a su pariente. Note que el tiempo de inicio mismo es un desplazamiento en el cuadro de referencia de aquella línea de tiempo pariente. Como un resultado, mientras la duración de la línea de tiempo este afectada por el factor velocidad, el tiempo de inicio no lo esta. La figura 35 muestra una línea de tiempo con `Begin=3`, `Duration=5` y `Speed=0,5`.

205

Las líneas de tiempo pueden ser organizadas en una estructura de árbol. Cada documento, cuadro o ventana tiene alguna línea de tiempo "raíz" implícita, lo que puede pensarse como que representa el mundo real, el tiempo de reloj de pared. Sin embargo, el tiempo $t=0$ para la línea de tiempo raíz es el tiempo cuando esa línea de tiempo es creada, esto es, cuando un documento es cargado, un cuadro se está navegando o una ventana está abierta.

Dada la naturaleza jerárquica del sistema de temporización, tiene sentido referirse al pasaje del tiempo como una ocurrencia en uno a tres cuadros de referencia. Un cuadro simple de referencia es el cuadro de referencia experimentado por una línea de tiempo individual. En este cuadro de referencia, el valor de progreso de una línea de tiempo es siempre 0 en $t=0$ y 1 en $t=d$, en donde d es una duración simple de una línea de tiempo. La duración de una línea de tiempo es siempre especificada en el cuadro simple de referencia. El cuadro de referencia de la línea de tiempo pariente es el cuadro simple de referencia para la línea de tiempo que es la pariente de cualquier línea de tiempo dada. Por ejemplo, el tiempo Begin para una línea de tiempo es siempre especificado en el cuadro de referencia de la línea de tiempo pariente. El cuadro global de referencia es el cuadro simple de referencia de la línea de tiempo raíz. En que el cuadro de referencia, el tiempo $t=5s$ ocurre 5 segundos

después de que la línea de tiempo es creada, y una dura exactamente 10s del mundo real.

En adición, las diversas reglas de control de la temporización aplican a sub árboles de temporización, incluyendo aquel si una línea de tiempo está activa, su línea de tiempo pariente también requiere estar activa. Inversamente, si una línea de tiempo no está activa, entonces ninguno de sus hijos puede estar activo y ninguno puede iniciarse. Si una línea de tiempo esta pausada explícitamente (por medio de una llamada del método `ITimingontrol.Pause`) entonces sus hijos están pausados implícitamente. Cuando esa línea de tiempo es reasumida, cualquiera de sus hijos que no fueron pausados explícitamente es también reasumido. Si una línea de tiempo es iniciada (por cualquiera de una variedad de razones, incluyendo el cruce de un punto de repetición) entonces sus hijos son reposicionados.

Una línea de tiempo puede ser emparentada explícitamente a otra línea de tiempo, en el fin cuyo caso la forma del árbol de temporización es explícito y claro. En muchos casos, sin embargo, es útil dejar que el sistema emparente una línea de tiempo automáticamente, con base de algún pariente de temporización por defecto. Una línea de tiempo que no esta emparentada explícitamente se dice que esta auto emparentada, y su línea de tiempo pariente efectiva depende en como la

línea de tiempo es usada. Dos tipos de auto emparentamiento a un partiente visual o emparentamiento a una raíz.

El pariente visual de una línea de tiempo está determinado implícitamente por como la línea de tiempo es usada. Por ejemplo, si la línea de tiempo controla una animación de color que, a su vez anima una brocha usada como fondo de algún visual V, entonces el V es el "pariente visual" de esa línea de tiempo. Si ese visual tiene una línea de defecto asociada, esa línea de tiempo es la pariente de nuestra línea de tiempo original en este ejemplo. De otra manera, el pariente del visual es examinado, recursivamente. La raíz de un árbol visual esta siempre asociada implícitamente con la línea de tiempo raíz, de modo que si visual está en el árbol visual entonces cualesquier líneas de tiempo auto emparentadas dentro de este están garantizadas que están emparentadas en algún lado en el árbol temporizar. Sin embargo, si el Visual no está todavía en el árbol Visual entonces sus líneas de tiempo permanecen por fuera del árbol de temporización hasta tal momento en que el Visual este insertado dentro de un arco.

La línea de tiempo "raíz" por defecto está también definida por el emparentamiento visual, excepto que, en este caso, el pariente Visual más cercano que tiene una línea de tiempo no sea usado necesariamente. En su lugar, con el emparentamiento

raíz de una línea de tiempo siempre está asociado con el Visual más alto en el árbol (que puede ser un objeto Frame o Window, y el Visual raíz asociado con un VisualManager).

Una vez una línea de tiempo está emparentada automáticamente, puede requerir ser reemparentada si cualquier cosa sucede que cambie la línea de tiempo pariente por defecto implíta. Por ejemplo, si el pariente visual inmediato de la línea de tiempo no tiene inicialmente una línea de tiempo defecto de su propiedad pero entonces una es establecida entonces la línea de tiempo requiere ser reemparentada. Este reemparentamiento ocurre automáticamente. El auto emparentamiento y el reemparentamiento son implementados por la interfaz IAIMatable, descrita más abajo.

Las líneas de tiempo y los objetos temporizados comparten un número de comportamientos en común, por ejemplo, una animación puede ser pausada o reiniciada, y una lista de animaciones puede estar activa o inactiva. Para mantener la consistencia, los objetos temporizados implementan uno o más interfaces que le permiten el acceso a los métodos de temporización y de sus propiedades.

La interfaz ITimingControl está implementada por objetos temporizados que pueden ser controlados en el tiempo de corrida:

```
public interface System.Windows.Media.Animation.ITimingControl
{
    // Timing attributes
    double Acceleration { get; set; }
    bool AutoReverse { get; set; }
    TimeSyncValue Begin { get; set; }
    double Deceleration { get; set; }
    Time Duration { get; set; }
    TimeSyncValue End { get; set; }
    TimeEndSync EndSync { get; set; }
    TimeFill Fill { get; set; }
    TimeFill FillDefault { get; set; }
    Timeline ParentTimeline { get; set; }
    double RepeatCount { get; set; }
    Time RepeatDuration { get; set; }
```

```
TimeRestart Restart { get; set; }
TimeRestart RestartDefault { get; set; }
double Speed { get; set; }

// Run-time timing state
int CurrentRepeat { get; }
Time CurrentTime { get; }
void Disable();
void Enable();
bool IsChanging { get; }
bool IsEnabled { get; }
bool IsForwardProgressing { get; }
bool IsOverridingBaseValue { get; }
bool IsPaused { get; }
bool IsReversed { get; }
double Progress { get; }

// Run-time timing control
void BeginIn(Time offset);
void Disable();
void Enable();
void EndIn(Time offset);
void Pause();
void Resume();
void Seek(Time offset, TimeSeekOrigin origin);
void OverrideSpeed(double speed);

// Event notifications
event EventHandler Begun { add; remove; }
event EventHandler Changed { add; remove; }
event EventHandler Ended { add; remove; }
event EventHandler Paused { add; remove; }
```

```

event EventHandler Repeated { add; remove; }
event EventHandler Resumed { add; remove; }
event EventHandler Reversed { add; remove; }
event EventHandler Sought { add; remove; }
}

```

La siguiente tabla resume las semánticas de la interfaz ITimingControl:

Method, property or Event	Significado
Acceleration	Un valor entre 0 y 1 que representa la fracción de la duración simple lapsada en la fase de tiempo de aceleración. La suma de este atributo y el atributo Deceleration puede no exceder de 1.
AutoReverse	Si este atributo es "True" entonces la línea de tiempo progresa desde el inicio al final y luego inmediatamente progresa hacia atrás desde el final al inicio. En este caso, la línea de tiempo puede estar activa el doble de la cantidad de tiempo especificada para el atributo Duration.
Begin	El tiempo en el cual esta línea de tiempo debe empezar. Por defecto este tiempo está relacionado con el tiempo de inicio de la línea de tiempo pariente, pero el desplazamiento puede también ser especificado con relación a algún otro en tiempo de inicio o fin de otra línea de tiempo. En este último caso, la otra línea de tiempo está emparentada a la misma de tiempo que esa.
BeginIn	Dispara un inicio interactivo y el punto especificado en el tiempo en el futuro o pasado. El parámetro esta en el cuadro de referencia de está línea de tiempo pariente de

	esta línea de tiempo. Si la línea de tiempo pariente no esta activa, este método no tiene efecto.
Begun	Surge siempre que el objeto ingresa a un periodo en el cual su estado interno esta cambiando continuamente.
Changed	Surge por el modificador siempre que su estado interno cambia.
Ended	Surge siempre que el objeto deja un periodo en el cual su estado interno esta cambiando continuamente.
CurrentRepeat	La interacción actual de la línea de tiempo, si esta se repite. La primera interacción es la interacción 1. Si <code>IsOverrindingBaseValue</code> es falso está propiedad regresa a 0.
CurrentTime	El tiempo local actual para esta línea de tiempo. Si el <code>IsOverrinfingBaseValue</code> es falso esta propiedad regresa a <code>Time.Unispecified</code>
Deceleration	Un valor entre 0 y 1 que representa la fracción de la duración simple gastada en la fase de tiempo de desaceleración. La suma de este atributo y del atributo <code>Acceleration</code> puede no exceder de 1.
Disable	Desase esta línea de tiempo, efectivamente removiéndola del árbol de temporización.
Duration	La duración de un periodo único desde el inicio al fin.
Enable	Permite esta línea de tiempo, efectivamente insertandola dentro del árbol de temporización. Este método no tiene efecto si esta es una línea de tiempo auto emparentada y un pariente por defecto no ha sido especificado.
End	El tiempo final máximo para esta línea de tiempo. Si este valor es menor que la suma de la propiedades <code>Begin</code> y <code>Duration</code> , entonces el periodo de activación es cortado por este tributo. En adición, todos los inicios (programados o interactivos) más haya del tiempo especificado por este atributo son ignorados.

5A

EndIn	Dispara un final interactivo en el punto especificado en el tiempo en el futuro o el pasado. El parámetro está en el cuadro de referencia de esta línea de tiempo pariente de la 1 Línea de tiempo. Si la línea de tiempo pariente no está activa, este método no tiene efecto.
EndSync	Este atributo es usado para definir la duración implícita de una línea de tiempo, la cual es usada si el atributo Duration no esta establecido explícitamente. La duración implícita de una línea de tiempo puede estar definida por un objeto temporizado que esta controla o por otras líneas de tiempo que pueden estar emparentadas a esta.
Fill	El comportamiento de la línea de tiempo después de que el tiempo de fin pasa. Por defecto, la línea de tiempo, está solamente "on" desde el inicio al fin, pero si este atributo se establece en "freeze" entonces la línea de tiempo permanece on pasado el tiempo final. En este cas, el valor de progreso después del tiempo de fin es igual a lo que fue en el tiempo de fin. Los valores posibles son Remove (defecto global), Freeze, Hola, Transition y Auto.
FillDefault	El valor por defecto para el atributo Fill. Si el atributo Fill no está especificado entonces este atributo es usado para determinar el comportamiento de llenado. En adición, este valor por defecto es heredado por líneas de tiempo emparentadas a esta, a menos que ellas tengan su propio conjunto de atributos FillDefault. Los valores posibles son iguales que para el atributo Fill.
IsChanging	Verdad si la línea de tiempo está activa, falso de otra manera.
IsEnabled	Verdead si la línea de tiempo es parte de un sub árbol de temporización, falso de otra manera. Si esta propiedad es verdad esto no

203

	garantiza que el subárbol esa línea de tiempo sea parte o está ella misma autorizada.
IsForwardProgressing	Verdad si el progreso en esta línea de tiempo se mueve de 0 a 1, según se compare con el tiempo del reloj de pared. Esta propiedad toma en cuenta el efecto de estar anidada en líneas de tiempo potencialmente reversadas. Si IsOverridingBaseValue es falso esta propiedad regresa al mismo valor que aquél que la línea de tiempo pariente de esta línea de tiempo regresaría.
IsOverridingBaseValue	Verdad si la línea de tiempo está activa o esta en un periodo de llenado.
IsPaused	Verdad si la línea de tiempo está activa, pero en un estado pausado.
IsReverse	Verdad si la línea de tiempo está en un periodo reversado, como se ve del cuadro de referencia local propio de la línea de tiempo. Esta propiedad no toma en cuenta el efecto de estar aliado en líneas de tiempo potencialmente reversadas. Si IsOverridingBaseValue es falso esta propiedad regresa a falso.
ParentTimeline	La línea de tiempo que es pariente temporizado de esta línea de tiempo. Esta puede ser una referencia para cualquier otra línea de tiempo, o uno de dos valores de referencia especiales: Timeline.VisualParent o Timeline.RootTimeline . Si esta propiedad está establecida para TimelineVisualParent entonces esta línea de tiempo es autoemparentada en el uso a la línea de tiempo asociada con el Visual en el cual es usado (si el Visual no tiene un DefaultTimeline asociado, entonces el Visual pariente es inspeccionado, recursivamente). Si este está establecido para el Timeline:RootTimeline entonces esta línea de tiempo es autoemparentada en el uso a la "raíz" del árbol de temporización.
Pause	Pausa esta línea de tiempo y todas sus líneas de tiempo hijos. Si esta línea de tiempo no

	está activa este método no tiene efecto.
Paused	Surgió para la línea de tiempo siempre que está o uno de sus descendientes ha sido pausado.
Progress	El valor de progreso actual de la línea de tiempo. Si <code>IsOverridingBaseValue</code> es falso está propiedad regresa a 0. en todos los casos, el valor de regreso de esta propiedad es siempre un valor entre 0 y 1, inclusive.
RepeatCount	El número de veces de un periodo de inicio hasta el fin debería ser repetido. Esto puede ser un valor fraccional, lo mismo que el valor especial <code>float.PositiveInfinity</code> para indicar que la línea de tiempo debe repetirse siempre. Si ambos de estos atributos y el atributo <code>RepeatDuration</code> están especificados, la duración activa total es el mínimo de dos.
RepeatDuration	La longitud de tiempo para el cual un periodo de inicio a fin debería ser repetido. Esto puede implicar, una cuenta de repetición fraccional, o puede ser un valor especial <code>Time.Indefinite</code> para indicar que la línea de tiempo debe repetirse para siempre. Si ambos de estos atributos y el atributo <code>RepeatCount</code> están especificados, la duración activa total es el mínimo de dos.
Repeated	Surgió para la línea de tiempo siempre que está se repite su duración.
Restart	El comportamiento de la línea de tiempo cuando un segundo tiempo de inicio (o posterior) es alcanzado por defecto, un tiempo de inicio interrumpe cualquier periodo activo y regresa de nuevo al tiempo <code>t=0</code> para la línea de tiempo, pero si este atributo está establecido en <code>WhenNotActive</code> entonces un tiempo de inicio interrumpiría un periodo activo es ignorado. Los valores posibles son <code>Always</code> , <code>WhenNotActive</code> y <code>Never</code> .
RestartDefault	El valor por defecto para el atributo <code>Restart</code> . Si el atributo <code>Restart</code> no está especificado entonces este atributo es usado para determinar

306

	el comportamiento de reinicio. En adición, este valor por defecto es heredado por las líneas de tiempo emparentadas a está, a menos que ellas tengan su propio atributo RestartDefault establecido. Los valores posibles son los mismos que para el atributo Restart.
Resume	Reasumen está línea de tiempo y todas sus líneas de tiempo hijos. Si está línea de tiempo no está activa y pausada este método no tiene efecto.
Resumed	Survió para la línea de tiempo siempre que es reasumida.
Reversed	Survió para la línea de tiempo cuando la dirección del tiempo cambia.
SEC	Cambia el tiempo actual para está línea de tiempo, la cual puede tener un efecto sobre todas las líneas de tiempo hijo. Si está línea de tiempo no está activa este método no tiene efecto.
Seeked	Survió para la línea de tiempo siempre que su tiempo cambia como un resultado de una operación de búsqueda.
Speed	La velocidad relativa a la cual el tiempo debe pasar para esta línea de tiempo, comparado con su línea de tiempo pariente. Por ejemplo, un valor de 1 significa velocidad normal, mientras que un valor de 2 significa que el tiempo que transcurre es 2 veces más rápido (y por lo tanto, la duración percivida finaliza solamente la mitad de lo especificado por el atributo Duration). Este valor puede ser negativo, en cuyo caso el tiempo fluye para atrás en está línea de tiempo, desde los tiempos de fin a el tiempo de inicio, a medida que la línea tiempo pariente fue invertida.

Las escenas de gráficos pueden ser animadas por medio de especificar parámetros animados para algunas operaciones de conversión, o por medio de agregar animaciones a ciertas

propiedades de elementos. Una animación es una función que toma algún conjunto arbitrario de entradas (por lo menos una de las cuales generalmente es una línea de tiempo) y produce una salida del tipo correcto para ser pasada a una operación de conversión. Por ejemplo, un `PointAnimation` convierte un valor de progreso de línea de tiempo en un tipo de valor `Point`. Al mismo tiempo, varias operaciones de conversión que toman uno o más valores `Point` como parámetros pueden también recibir un `PointAnimation` en lugar de un `Point`, en cuyo caso la función de animación es evaluada en cada cuadro para calcular el `Point` que se usa en ese cuadro.

Las animaciones son agrupadas en colecciones. Una colección de animación trabaja como una tubería, tomando como una entrada el valor base de una propiedad y produciendo como su salida el valor actual que debería ser usado para esa propiedad. La colección encadena 0 o más objetos de animación, cada uno soporta semánticas similares de toma de un valor de entrada y produce una salida de tipo similar. La tubería es evaluada a intervalos regulares, y la salida es usada en operaciones de conversión, produciendo el efecto de animación.

Ya que los valores que pueden ser animados tienen diversos tipos, hay también diversas animaciones tipificadas diferentemente. Sin embargo, todas las animaciones siguen un

patrón común y todas implementan un conjunto de interfaces comunes. Los objetos de animación están organizados en tres grupos de clases, particularmente modificadores, modificadores temporizados y animaciones.

Una animación directa interpola un valor entre los puntos de inicio y de fin. Cuando ambos los puntos de inicio y fin son especificados, el valor base es ignorado para el tiempo en que la animación está "on". Cuando la animación está "off", el valor de la propiedad puede revertirse al valor base. Note que una animación está "on" siempre que su línea de tiempo asociada está "on". Por lo tanto, un from-to-animation puede estar hecho para pasar por alto permanentemente el valor de base estableciendo el atributo de temporización Fill para "Freeze". La figura 36 muestra un punto animado en Y con From=10 y To=70.

Si solamente el punto de inicio o de fin está especificado, pero no ambos, el valor de base de la propiedad es usado para el valor del otro punto. Mientras esto parece redundante con el ejemplo previo, hay una diferencia clave en que este caso, el valor de base no es ignorado, sino compuesto con la animación. Esto puede producir efectos interesantes si el valor de base es cambiante (porque la propiedad está siendo modificada por otro proceso) o si la animación está encadenada a otra animación.

570

Otra manera para especificar la función de animación es especificar un delta del valor base. Esto es similar conceptualmente a una animación from-to que interpola desde el valor base al valor base más el delta. Sin embargo, en este caso tanto los puntos de inicio como de fin están compuestos con el valor base.

Si la línea de tiempo asociada con una animación es establecida para ser repetida, la animación corre desde el inicio al fin varias veces. La figura 37 muestra un punto animado en y con Form=10, By=60 y RepeatCount=2. En lugar de repetir la misma trayectoria en cada interacción, una animación puede ser programada para acumular el efecto de cada interacción en esencia componiéndose con ella misma. La figura 38 muestra un punto animado en y con From=10, By=60, RepeatCount=2 y IsAccumulating=True.

Aunque el comportamiento por defecto de una animación from-to es ignorar el valor base de la propiedad animada, este comportamiento puede ser cambiado a un comportamiento aditivo, en donde los valores from y to son ambos deltas del valor base. La siguiente tabla resume los tipos de animación básicos.

610

521

Tipo	Valor de salida
From	El valor "from" en $t=0$ y el valor base en $t=1$
To	El valor base en $t=0$ y el valor "To" en $t=1$
From-To	El valor "From" en $t=0$ y el valor "To" en $t=1$.
By	El valor e base en $t=0$ y la suma del valor de base y el valor "By" en $t=1$.
From-By	El valor "From" en $t=0$ y la suma del valor "From" y "By" en $t=1$.

Con las animaciones básicas, el valor de salida en los puntos de comienzo y fin están especificados, y una interpolación lineal es usada para calcular los valores intermedios. Para funciones de animación más complejas, una lista de valores puede ser especificada en su lugar. Cada valor corresponde a un cuadro clave. En un caso simple, estos cuadros clave ocurren a intervalos regulares. Las animaciones también pueden ser programadas para usar un espaciamiento de pasos entre los cuadros clave. En el método de interpolación por pasos, el espacio entre cada par de cuadros clave es proporcional a la proporción de la "distancia" entre dos valores claves de la "distancia total" cubierta por la animación. Esto es posible para animaciones cuyos tipos tienen un concepto significativo de "distancia" tal como, por ejemplo, animaciones flotantes o de punto. En tal caso, la

get

interpolación entre cuadros clave es lineal. Una tercera opción es no interpolar de ninguna manera, en cuyo caso la función de valor de salida es discreta. La figura 39 muestra un punto animado en y con KeyValues=10, 90, 17 y varios métodos de interpolación.

Para control adicional, el tiempo para cada cuadro clave puede ser especificado explícitamente. La interpolación entre los cuadros clave puede ser lineal o discreta. Los tiempos clave son especificados como porcentajes de la duración de animación total, y deben cubrir el periodo completo. En otras palabras, el primer tiempo clave es 0, y para la interpolación lineal el último tiempo clave es 1. la figura 40 muestra un punto animado en y con KeyValues=10, 90, 50 y KeyTimes=0, . 2, 1.

Para tener un control adicional sobre la interpolación, un conjunto de curvas cúbicas Bezier pueden usados para describir la curva del tiempo usada para la animación. Esto no debería confundirse con una curva Bezier convertida en pantalla; la curva es usada para modificar la forma de la curva de temporización, pero los valores del cuadro clave todavía se interpolan linealmente durante el valor de progreso. Está interpolación estriada agrega un filtro que convierte el valor de progreso lineal 0-1 provisto por la

línea de tiempo asociada con la animación en una curva de progreso no lineal 0-1.

La siguiente tabla contiene una lista de atributos específicos de animación y sus significados. Esta lista es una plantilla seguida por todos los objetos de animaciones. En donde el tipo de un atributo es "<ValueType>", un objeto real expondría el atributo con un tipo que caza de tipo de animación. Por ejemplo, un objeto ColorAnimation describe estos atributos como "Color". En adición a los atributos dados abajo, los objetos de animación soportan los atributos especificados en la interfaz ITimingAttributes.

Atributo	Tipo	Significado
By	<ValueType>	El valor delta al fin de la animación. El valor en el comienzo es o el valor From, si está especificado, o el valor de base de la propiedad.
From	<ValueType>	El valor inicial de la animación.
InterpolationMethod	InterpolationMethod	El método usado par interpolar entre valores clave. Los valores posibles son Discrete, Linear, Paced o Spline.
KeySplines	KeySplineCollection	Un conjunto de puntos de control Bezier asociados con una lista KeyTimes, que define una función cúbica que controla el intervalo de pasos de la

		animación. Esta lista debe contener un elemento menos que la lista KeyTimes. Esta lista es usada solamente si el atributo InterpolationMethod es establecido en Spline.
KeyTimes	KeyTimeCollection	Una lista de valores de tiempo usados para controlar los pasos de la animación. Esta lista debe contener el mismo número de elementos que la lista KeyValues. La lista está ordenada en valores de tiempo incrementales, y el primer valor en esta lista requiere ser 0 y el último 1, a menos que el InterpolationMethod es establecido en Discrete, en cuyo caso el último valor puede ser cualquier cosa menos que o igual a 1.
KeyValues	<ValueType>KeyValueCollection	Una lista de valores para la animación.
To	<ValueType>	El valor al fin de la animación.

La clase Animatable es derivada de la clase Changeable. Está usada como una clase de base por cualquier objeto o colección de objetos que pueden ser animados o que contienen valores animados, tales como un recurso de medios. Los Modifiers, TimedModifiers, y Animations irónicamente se

— 8A

derivan de Changeable en lugar de Animatable por que sus propiedades individuales no son animables.

```
public abstract class Animatable : Changeable
{
    public abstract bool HasAnimations { get; }
    public abstract bool IsAnimating { get; }
    public abstract bool IsOverridingBaseValue { get; }

    public abstract Animatable GetCurrentValue();
    public abstract void SetDefaultParentTimeline(Timeline defaultParentTimeline);
}
```

Método, Propiedad o evento

Significado

HasAnimations

Verdad si el objeto puede variar con el tiempo. En general, esta propiedad es verdadera si el objeto está manteniendo algunas colecciones de animación.

IsAnimating

Verdadero si cualquiera de las animaciones en el objeto están cambiando (ver Modifier.IsChanging).

IsOverridingBaseValue

Verdadero si cualquiera de las animaciones en el objeto están cambiando o en un estado de llenado y por lo tanto activa actualmente y modifica al objeto.

GetCurrentValue

Regresa un objeto que tiene

```

public sealed class CloseSegment : PathSegment
{
    public CloseSegment(bool isStroked);
    public new CloseSegment Copy();
}

public sealed class LineSegment : PathSegment
{
    public LineSegment();
    public LineSegment(Point point, bool isStroked);
    public LineSegment(
        Point point,
        PointAnimationCollection pointAnimations,
        bool isStroked);
    public new LineSegment Copy();

    [Animations("PointAnimations")]
    public Point Point { get; set; }
    public PointAnimationCollection PointAnimations { get; set; }
}

public sealed class BezierSegment : PathSegment
{
    public BezierSegment();
    public BezierSegment(
        Point point1,
        Point point2,
        Point point3,
        bool isStroked); --
    public BezierSegment(
        Point point1, PointAnimationCollection point1Animations,

```

```

    Point point2, PointAnimationCollection point2Animations,
    Point point3, PointAnimationCollection point3Animations,
    bool isStroked);
public new BezierSegment Copy();

[Animations("Point1Animations")]
public Point Point1 { get; set; }
public PointAnimationCollection Point1Animations { get; set; }

[Animations("Point2Animations")]
public Point Point2 { get; set; }
public PointAnimationCollection Point2Animations { get; set; }

[Animations("Point3Animations")]
public Point Point3 { get; set; }
public PointAnimationCollection Point3Animations { get; set; }
}

public sealed class QuadraticBezierSegment : PathSegment
{
    public QuadraticBezierSegment();
    public QuadraticBezierSegment(
        Point point1,
        Point point2,
        bool isStroked);
    public QuadraticBezierSegment(
        Point point1, PointAnimationCollection point1Animations,
        Point point2, PointAnimationCollection point2Animations,
        bool isStroked);
    public new QuadraticBezierSegment Copy();

    [Animations("Point1Animations")]

```

```

public Point Point1 { get; set; }
public PointAnimationCollection Point1Animations { get; set; }

[Animations("Point2Animations")]
public Point Point2 { get; set; }
public PointAnimationCollection Point2Animations { get; set; }
}

public sealed class PolyLineSegment : PathSegment
{
    public PolyLineSegment();
    public PolyLineSegment(Point[] points, bool isStroked);
    public PolyLineSegment(ICollection points, bool isStroked);
    public new PolyLineSegment Copy();

    public void AddPoint(Point point);
    public PointCollection Points { get; set; }
}

public sealed class PolyBezierSegment : PathSegment
{
    public PolyBezierSegment();
    public PolyBezierSegment(Point[] points, bool isStroked);
    public PolyBezierSegment(ICollection points, bool isStroked);
    public new PolyBezierSegment Copy();

    public void AddPoint(Point point);
    public PointCollection Points { get; set; }
}

public sealed class PolyQuadraticBezierSegment : PathSegment
{

```

```

public PolyQuadraticBezierSegment(Point[] points, bool isStroked);
public PolyQuadraticBezierSegment(ICollection points, bool isStroked);
public new PolyQuadraticBezierSegment Copy();

public void AddPoint(Point point);
public PointCollection Points { get; set; }
}

public sealed class ArcSegment : PathSegment
{
    public ArcSegment();
    public ArcSegment(
        Point point,
        Size arcSize,
        double xRotation,
        bool largeArcFlag,
        bool sweepFlag,
        bool isStroked);
    public ArcSegment(
        Point point,
        PointAnimationCollection pointAnimations,
        Size arcSize,
        SizeAnimationCollection arcSizeAnimations,
        double xRotation,
        DoubleAnimationCollection xRotationAnimations,
        bool largeArcFlag,
        bool sweepFlag,
        bool isStroked);
    public new ArcSegment Copy();

    [Animations("PointAnimations")]
    public Point Point { get; set; }
}

```

```

public PointAnimationCollection PointAnimations { get; set; }

[Animations("SizeAnimations")]
public Size Size { get; set; }
public SizeAnimationCollection SizeAnimations { get; set; }

[Animations("XRotationAnimations")]
public double XRotation { get; set; }
public DoubleAnimationCollection XRotationAnimations { get; set; }

public bool LargeArc { get; set; }
public bool SweepFlag { get; set; }
}

```

Cada uno de los objetos PathSegment que son agregados a un PathFigure también tienen la propiedad PathSegment.IsStroked. si un PathSegment tiene esta propiedad establecida en verdadero, entonces el PathSegment particular contribuirá al área golpeada del PathGeometry cuando es convertida con un Pen. Esto también aplica al ensayo por golpe y a una ampliación explícita del PathGeometry. El comportamiento particular cuando se conmuta entre secciones PathSegment golpeadas y no golpeadas en un PathFigure es el mismo que si el Pen especificado en punteado, en que el punto de punteado apropiados pueden ser aplicados a los extremos del PathSegment.

El siguiente ejemplo demuestra la marca que usa un PathGeometry, un PathFigure, y un PathSegments:

```
<Path Fill="#4000FF00" Stroke="Yellow" >
```

```
<Path.Data>
```

```
<PathGeometry Transform="translate(225 25)" FillRule="EvenOdd">
```

```
<PathGeometry.Figures>
```

```
<PathFigureCollection>
```

```
<PathFigure IsFilled="True">
```

```
<PathFigure.Segments>
```

```
<PathSegmentCollection>
```

```
<StartSegment Point="0 0" IsStroked="False" />
```

```
<LineSegment Point="100 0" />
```

```
<BezierSegment Point1="125 25" Point2="125 75"
```

```
Point3="100 100" />
```

```
<QuadraticBezierSegment Point1="50 50" Point2="0 100" />
```

```
<ArcSegment Point="100 150" Size="100 100"
```

```
XRotation="45" LargeArc="False" SweepFlag="True" />
```

```
<PolyLineSegment Points="100 175 0 175" />
```

```
<PolyBezierSegment Points="50 225 50 275 0 300 50 325 50  
375 0 400" />
```

```
<PolyQuadraticBezierSegment Points="50 450 0 500 50 550 0  
600" />
```

```
<CloseSegment IsStroked="True" />
```

```
</PathSegmentCollection>
```

```
</PathFigure.Segments>
```

```
</PathFigure>
```

```
</PathFigureCollection>
```

```
</PathGeometry.Figures>
```

```
</PathGeometry>
```

```
</Path.Data>
```

```
</Path>
```


Un RectangleGeometry define un rectángulo o un objeto de geometría de rectángulo de esquinas redondeadas. El radio X y el radio Y se refieren a los ejes alineados en longitud radial de las esquinas redondeadas:

```
public sealed class RectangleGeometry : Geometry
{
    public RectangleGeometry();
    public RectangleGeometry(Rect rect);
    public RectangleGeometry(
        Rect rect,
        double radiusX,
```

```

    double radiusY);
public RectangleGeometry(
    Rect rect, RectAnimationCollection rectAnimations,
    double radiusX, DoubleAnimationCollection radiusXAnimations,
    double radiusY, DoubleAnimationCollection radiusYAnimations);
public RectangleGeometry(
    Rect rect, RectAnimationCollection rectAnimations,
    double radiusX, DoubleAnimationCollection radiusXAnimations,
    double radiusY, DoubleAnimationCollection radiusYAnimations,
    Transform);
public new RectangleGeometry Copy();

[Animation("RectAnimations")]
public Rect Rect { get; set; }
public RectAnimationCollection RectAnimations { get; set; }

[Animation("RadiusXAnimations")]
[DefaultValue(0.0)]
public double RadiusX { get; set; }
public DoubleAnimationCollection RadiusXAnimations { get; set; }

[Animation("RadiusYAnimations")]
[DefaultValue(0.0)]
public double RadiusY { get; set; }
public DoubleAnimationCollection RadiusYAnimations { get; set; }
}

```

El siguiente ejemplo demuestra la marca que usa un
RectangleGeometry:

```
<Path Fill="W4000FP00">
```

```
<Path.Data>
```

```
<RectangleGeometry Rect="125 125 75 75" RadiusX="10"
RadiusY="5"></RectangleGeometry>
```

```
</Path.Data>
```

```
</Path>
```

Una `EllipseGeometry` define un área elíptica dado los ejes alineados radialmente de longitudes X y Y:

```
public sealed class EllipseGeometry : Geometry
{
    public EllipseGeometry();
    public EllipseGeometry(Rectangle rect);
    public EllipseGeometry(
        Point center,
        double radiusX,
        double radiusY);
    public EllipseGeometry(
        Point center, PointAnimationCollection centerAnimations,
        double radiusX, DoubleAnimationCollection radiusXAnimations,
        double radiusY, DoubleAnimationCollection radiusYAnimations);
    public EllipseGeometry(
        Point center, PointAnimationCollection centerAnimations,
        double radiusX, DoubleAnimationCollection radiusXAnimations,
        double radiusY, DoubleAnimationCollection radiusYAnimations,
        Transform transform);
    public new EllipseGeometry Copy();

    [Animation("CenterAnimations")]
    public Point Center { get; set; }
    public PointAnimationCollection CenterAnimations { get; set; }

    [Animation("RadiusXAnimations")]
    public double RadiusX { get; set; }
    public DoubleAnimationCollection RadiusXAnimations { get; set; }
```

```

[Animation("RadiusYAnimations")]
public double RadiusY { get; set; }
public DoubleAnimationCollection RadiusYAnimations { get; set; }
}

```

El siguiente ejemplo demuestra la marca que usa un EllipseGeometry:

```

<Path Fill="#4000FF00">
  <Path.Data>
    <EllipseGeometry Center="50 300" RadiusX="50"
      RadiusY="75"></EllipseGeometry>
  </Path.Data>
</Path>

```

Un LineGeometry define un segmento de línea entre dos puntos y por lo tanto no contiene área para llenar:

```

public sealed class LineGeometry : Geometry
{
    public LineGeometry();
    public LineGeometry(Point pt1, Point pt2);
    public LineGeometry(
        Point startPoint,
        PointAnimationCollection startPointAnimations,
        Point endPoint,
        PointAnimationCollection endPointAnimations);
    public LineGeometry(
        Point startPoint,
        PointAnimationCollection startPointAnimations,
        Point endPoint,
        PointAnimationCollection endPointAnimations,
        Transform transform);
    public new LineGeometry Copy();
}

```

```

[Animation("Point1Animations")]
public Point StartPoint { get; set; }
public PointAnimationCollection StartPointAnimations { get; set; }

[Animation("Point2Animations")]
public Point Point2 { get; set; }
public PointAnimationCollection Point2Animations { get; set; }
}

```

El siguiente ejemplo demuestra la marca que usa un LineGeometry:

537

```

<Path Stroke="#4000FF00"
  StrokeThickness="20"
  StrokeStartLineCap="Flat"
  StrokeEndLineCap="Triangle"
  StrokeDashCap="Round"
  StrokeLineJoin="Bevel"
  StrokeMiterLimit="100"
  StrokeDashOffset="0"
  StrokeDashArray="1.0 2.0">
  <Path.Data>
    <LineGeometry StartPoint="125 25" EndPoint="175 75"></LineGeometry>
  </Path.Data>
</Path>

```

FORMACIÓN DE IMÁGENES

ImageSource es una clase abstracta, que comprende un bloque de construcción básico para el entubado de formación de imágenes. Un ImageSource representa conceptualmente un conjunto constante único de píxeles en un cierto tamaño y resolución. Por ejemplo, un ImageSource puede ser un cuadro único en un archivo de imágenes que un Decoder suministraría, o puede ser los resultados de una transformación que opera sobre cierto ImageSource de su propiedad. Un ImageSource no es multicuadro o animado. Un ImageSource es cambiable, no por que sus propias propiedades puedan ser cambiadas, sino por

722

que las propiedades de sus subclases pueden potencialmente ser cambiadas.

Por razones de desempeño, el soporte de los ImageSources que suministran acceso no administrado a la imagen, que usan la interfaz IMILBitmapSource. Si la subclase de ImageSource no la suministra, entonces la clase de base ImageSource lo hace (usando una clase de envoltura).

```
namespace System.Windows.Media
{
    public abstract class ImageSource : Changeable
    {
        /// Native format of the image's data.
        /// If the ImageSource is directly readable, this is the format the
        /// pixels will be in when they are read.
        public abstract PixelFormat Format { get; }

        /// Whether the ImageSource can convert its data to the specified format.
        /// If not, a format converter could be used to do the conversion.
        /// Note: for best performance, ImageSources should provide support for
        /// PixelFormat32bppPARGB.
        public virtual bool CanConvertTo(PixelFormat targetPixelFormat)

        /// Width, in pixels, of the image.
        public abstract int PixelWidth { get; }

        /// Height, in pixels, of the image.
        public abstract int PixelHeight { get; }

        /// Horizontal DPI of the image.
        public abstract double DpiX { get; }

        /// Vertical DPI of the image.
        public abstract double DpiY { get; }
```

/// Get the width of the image in measure units (96ths of an inch).

public double Width { get; }

/// Get the width of the image in measure units (96ths of an inch).

public double Height { get; }

/// Get the Palette for the image, if there is one.

ImagePalette Palette { get; }

/// Copy the pixel data from the image into the array of pixels that has the

/// specified stride, starting at the pixelOffset (specified in number of

/// pixels from the beginning). The pixels should be copied into

/// the specified pixelFormat. To find out if the pixelFormat is supported,

/// call CanConvertTo first.

public void Copy(PixelFormat pixelFormat, byte[] pixels,

int stride, int pixelOffset)

/// Copy a rect of pixel data from the image into the array of pixels that has

/// the specified stride, starting at the pixelOffset (specified in number of

/// pixels from the beginning). The pixels should be copied into

/// the specified pixelFormat. To find out if the pixelFormat is supported,

/// call CanConvertTo first. An empty rect (one with a width and/or height

/// of 0), means to ignore the rect and copy the entire image.

public abstract void Copy(IntegerRect sourceRect, PixelFormat pixelFormat,

byte[] pixels, int stride, int pixelOffset);

/// If there is an embedded thumbnail, return it.

/// Otherwise, return null. This method does NOT create a

/// thumbnail for images that don't already have one.

public virtual ImageSource EmbeddedThumbnail { get; set; }

/// Get a thumbnail of the image.

Handwritten signature


```

/// This method always returns a thumbnail that is in approximately the
/// same aspect ratio as the original image.
///
/// If there is an embedded thumbnail, this method returns that
/// thumbnail, scaled to the specified size. If there is not an embedded
/// thumbnail, the image itself is scaled to the specified size.
///
/// The specified width and height of the thumbnail must both be > 0.
/// We choose whether to use the specified width value or the specified
/// height value by determining which will give the largest possible
/// thumbnail, while maintaining the aspect ratio and staying less than
/// or equal to both the width and height values.
///
/// Therefore, to specify that you wish to use only the width value, you
/// would use Int32.MaxValue for the height value.
public virtual ImageSource GetThumbnail(ImageSizeOptions sizeOptions);

/// Provides access to this image's metadata.
public virtual ImageMetaData MetaData { get; set; }

        /// Returns an IMILBitmapSource for this ImageSource.
        unsafe internal IntPtr BitmapSource { get; }
    }
}

```

ImageData es una subclase de ImageSource. El ImageData implementa ImageSource para diferentes y diversas fuentes de imágenes:

HICON**HBITMAP****Pixel data in managed memory.****Pixel data in unmanaged memory****Images in a System.IO.Stream (requiring a decoder)****Another ImageSource**

ImageData suministra servicios, que incluyen capturar la imagen decodificada en la memoria del sistema, hacer crecer la imagen hasta una fuente rectangular especificada (en una memoria) y dimensionar la imagen hasta un ancho y altura de decodificación especificada (en la memoria). Para los escenarios de codificación de imagen el ImageData permite la especificación de que decodificador usar o descubrir el codec automático, con base en la corriente de entrada y el tipo mismo. El ImageData no soporta cargar una imagen directamente de un URI. El cargador debería ser usado para mapear el URI a una corriente que pueda ser usada para construir un ImageData. Una vez que el ImageData es construido, sus únicas propiedades cambiables son su miniatura en bebida, sus meta datos, y sus datos de píxeles. Las otras propiedades se consideran inmutables.

El dato de pixel del ImageData puede ser cambiado, incluyendo una de dos maneras: (1) obtener un Drawing Context para el

gzb

ImageData y emitir, comandos por vía del Drawing Context para dibujar sobre la imagen, o (2) usar el ImageData como un destino (RenderTarget) para un VisualManager y emitir un comando para convertir un árbol visual (una escena) en ImageData. En cualquier caso, el dibujo es hecho en una imagen en la memoria, lo que significa que el dato de píxel requiere primero ser decodificado y almacenado en la memoria. Solamente la imagen en la memoria cached es cambiado, el archivo de imagen en si mismo no es afectado (a menos que el ImageData sea codificado posteriormente al archivo de imagen usando un ImageEncoder).

```
namespace System.Windows.Media
```

```
{
```

```
/// ImageData can be constructed with or without a cache.
```

```
/// If constructed without a cache, the image will be decoded every time it is
```

```
/// used. The advantage of this approach is that the cache is not stored in
```

```
/// memory and that the decoding and filtering can be optimal for the size that
```

```
/// it is being drawn. The disadvantage is that the decoding has to be done with
```

```
/// every redraw. The other approach is to cache the decoded image. There are
```

```
/// several options for creating the decoded, cached image. It can be created at
```

```
/// a specific size and with a specific source rect (area of interest from the
```

```
/// source image). There are no options if the image is not cached -- it is
```

```
/// decoded with no source cropping. If the image is cached, the cached version
```

**/// of the image becomes the source image, and the original image source will
/// be discarded.**

public class ImageData : ImageSource

{

/// Construct an ImageData from a Stream.

/// The caller owns the stream and is responsible for closing it.

public ImageData(System.IO.Stream imageStream);

/// Construct an ImageData from a Stream.

/// The caller owns the stream and is responsible for closing it.

public ImageData(System.IO.Stream imageStream,

CodecInfo decoderInfo, // identifies the codec to be used (or null)

bool createCache, // if false, sourceRect and sizeOptions are ignored

IntegerRect sourceRect,

ImageSizeOptions sizeOptions

);

/// Construct an ImageData from an array of pixels.

unsafe public ImageData(

int pixelWidth,

int pixelHeight,

double dpiX,

double dpiY,

PixelFormat pixelFormat,

ImagePalette imagePalette,

byte[] pixels,

int stride,

IntegerRect sourceRect,

ImageSizeOptions sizeOptions

);

/// Construct an ImageData from an array of pixels in unmanaged memory

/// (e.g. a DibSection).

```
public ImageData(  
    int pixelWidth,  
    int pixelHeight,  
    double dpiX,  
    double dpiY,  
    PixelFormat pixelFormat,  
    ImagePalette imagePalette,  
    IntPtr pixels, // unmanaged array of pixels, e.g. a DibSection  
    int stride,  
    bool createCache, // if false, sourceRect and sizeOptions are ignored  
    IntegerRect sourceRect,  
    ImageSizeOptions sizeOptions  
);
```

/// Construct an ImageData from a HBITMAP.

```
public ImageData(  
    HBITMAP hbitmap,  
    HPALETTE hpalette,  
    IntegerRect sourceRect,  
    ImageSizeOptions sizeOptions  
);
```

/// Construct an ImageData from a HICON.

```
public ImageData(  
    HICON hicon,  
    IntegerRect sourceRect,  
    ImageSizeOptions sizeOptions  
);
```

/// Construct an ImageData from an ImageSource.

```
public ImageData(  
    ImageSource imageSource,
```

```

    bool createCache, // if false, sourceRect and sizeOptions are ignored
    IntegerRect sourceRect,
    ImageSizeOptions sizeOptions
);
/// Get the information about the specific codec that was used
/// to decode the image (if there a codec was required and we
/// have that information).
public CodecInfo DecoderInfo { get; }
/// Get a DrawingContext to draw on the ImageData.
public DrawingContext Open();
}
}

```

El `ImageDecoder` es una clase abstracta que suministra la clase de base para los decodificadores. Esta suministra una manera para determinar como muchos cuadros están en la imagen y para enumerar (o indexar) los cuadros. Como se mencionó anteriormente los cuadros de imágenes son cada uno un `ImageSource`. Los códigos de construcción crean un objeto `ImageData` para cada cuadro solicitado. Los códigos de adición pueden usar una subclase diferente para regresar una `ImageSource` para cada cuadro. El `ImageDecoder` no es un `ImageSource` en si mismo, sino que en su lugar es un contenedor para uno o más `ImageSources`. Note que cada cuadro de la imagen podría potencialmente tener diferentes atributos (tamaño diferente, resolución diferente, etc.).

```
namespace System.Windows.Media
{
    /// ImageDecoder is essentially a container for image frames.
    /// Each image frame is an ImageSource.
    public abstract class ImageDecoder : ICollection, IEnumerable
    {
        /// The number of image frames in this image.
        public int Count { get; }

        --

        /// Whether access to the ICollection is synchronized (thread-safe).
        public bool IsSynchronized { get; }
```

```
/// Gets an object that can be used to synchronize access to the ICollection.
public object SyncRoot { get; }

/// Copies the frames to an Array, starting at a particular Array index.
public void CopyTo(Array array, int index);

/// Returns an enumerator to iterate through the frames of the image.
public IEnumerator GetEnumerator();

/// Returns a codec-specific object that identifies the properties
/// that were used to encode this image (if supported by the codec).
/// These codec-specific properties can be passed back to the associated
/// encoder (if there is one) to get the same type of encoding again.
public virtual object CustomEncoderProperties { get; }

/// The info that identifies this codec (including any associated encoder).
public abstract CodecInfo Info { get; }

/// Get the pixel format for the first frame.
public virtual System.Windows.Media.PixelFormat Format { get; }

/// Get the pixel width for the first frame.
public virtual int PixelWidth { get; }

/// Get the pixel Height for the first frame.
public virtual int PixelHeight { get; }

/// Get the horizontal dpi for the first frame.
public virtual double DpiX { get; }

/// Get the vertical dpi for the first frame.
```



```
public virtual double DpiY { get; }
```

```
/// Get the width of the image in measure units (96ths of an inch).
```

```
public double Width { get; }
```

```
/// Get the height of the image in measure units (96ths of an inch).
```

```
public double Height { get; }
```

```
/// If there is an embedded thumbnail for the first frame, return it.
```

```
/// Otherwise, return null. This method does NOT create a
```

```
/// thumbnail for images that don't already have one.
```

```
public virtual ImageSource EmbeddedThumbnail { get; }
```

```
/// Return an image frame, cropped by the specified sourceRect.
```

```
/// An empty rect (one with a width and/or height of 0),
```

```
/// means to ignore the rect and get the entire frame.
```

```
public virtual ImageSource GetFrame(  
    int frameIndex,  
    bool createCache,  
    IntegerRect sourceRect);
```

```
/// Return an image frame, with all the specified cropping and sizing applied.
```

```
/// An empty source rect (one with a width and/or height of 0),
```

```
/// means to ignore the rect and get the entire frame.
```

```
public virtual ImageSource GetFrame(  
    int frameIndex,  
    bool createCache,  
    IntegerRect sourceRect,  
    ImageSizeOptions sizeOptions);
```

```
/// Indexer for returning a specific frame of the image (at full size).
```

```
/// The index must be in the range: (NumFrames > index >= 0)
```

```

public abstract ImageSource this[int frameIndex] { get; }

/// Conversion from ImageDecoder to ImageSource by returning the first frame.
public static explicit operator ImageSource(ImageDecoder decoder);

/// Provides read-only access to this image's metadata.
public abstract ImageMetadata Metadata { get; }
}
}

```

Un número de decodificadores contruidos allí son suministrados por el MIL, incluyendo ImageDecoderBmp.cs, ImageDecoderGif.cs, ImageDecoderIcon.cs, ImageDecoderJpeg.cs, ImageDecoderPng.cs, ImageDecoderTiff.cs, e ImageDecoderWmp.cs. Cada ImageDecodr implmentado y un constructor único que usa un System.IO.Stream para inicializar el decodificador, como en el siguiente ejemplo:

```

/// If this decoder cannot handle the image stream, it will
/// throw an exception.
public ImageDecoderJpeg(System.IO.Stream imageStream);

```

Un ImageEncoder es una colección de ImageSources (cuadros de imagen), cada uno potencialmente con su propio meta dato y miniatura. También puede haber una miniatura global y meta dato global asociado con el conjunto completo de cuadros. Un código puede también ser escogido para suministrar soporte

para codificar propiedades, usado para determinar como codificar la imagen. La colección de cuadros puede ser guardada (codificada) en cualquier número de corrientes especificadas (en un tiempo). La colección puede ser limpiada o clarificada y luego llenada con una colección diferente y guardada de nuevo.

```
namespace System.Windows.Media
```

```
{
```

```
    /// ImageEncoder collects a set of frames (ImageSource's) with their associated
```

```
    /// thumbnails and metadata and saves them to a specified stream. In addition
```

```
    /// to frame-specific thumbnails and metadata, there can also be an image-wide
```

```
/// (global) thumbnail and metadata, if the codec supports it.  
public abstract class ImageEncoder : IList, IDisposable  
{  
    public void Dispose();  
  
    /// Gets a value indicating whether the IList has a fixed size.  
    public bool IsFixedSize { get; }  
  
    /// Gets a value indicating whether the IList is read-only.  
    public bool IsReadOnly { get { return false; } }  
  
    /// Indexer for a specific frame of the image. The set method is  
    /// a Replace operation for that frame. The object must be an ImageSource.  
    public object IList.this[int frameIndex] { get; set; }  
    public ImageSource this[int frameIndex] { get; set; }  
  
    /// Add a frame to the image.  
    public int IList.Add(object value);  
    public int Add(ImageSource imageFrame);  
  
    /// Reset the encoder to an empty state, with no frames,  
    /// thumbnails, metadata, or encoding properties.  
    public void Clear();  
  
    /// Determines whether the Encoder contains a specific frame.  
    public bool IList.Contains(object value);  
    public bool Contains(ImageSource imageFrame);  
  
    /// Determines the index of the specified frame.  
    public int IList.IndexOf(object value);  
    public int IndexOf(ImageSource imageFrame);
```

726

```
/// This method inserts the frame at specified frame location, with its  
/// associated thumbnail and metadata (if any).  
public void IList.Insert(int frameIndex, object value);  
public void Insert(int frameIndex, ImageSource imageFrame);  
  
/// Removes the first occurrence of the specified frame.  
public void IList.Remove(object value);  
public void Remove(ImageSource imageFrame);  
  
/// Remove the specified frame from the image. Subsequent frames  
/// are moved up in the list.  
public void RemoveAt(int frameIndex);  
  
/// The number of image frames in this image.  
public int Count { get; }  
  
/// Whether access to the ICollection is synchronized (thread-safe).  
public bool IsSynchronized { get; }  
  
/// Gets an object that can be used to synchronize access to the ICollection.  
public object SyncRoot { get; }  
  
/// Copies the frames to an Array, starting at a particular Array index.  
public void CopyTo(Array array, int index);  
  
/// Returns an enumerator to iterate through the frames of the image.  
public IEnumerator GetEnumerator();  
  
/// The info that identifies this codec (including any associated decoder).  
public CodecInfo-Info { get; }  
  
/// A codec-specific object that identifies the properties
```

```

/// used to encode this image.
public object CustomEncoderProperties { get; set; }

/// Set or get the image's global embedded thumbnail, if there is one.
public ImageSource EmbeddedThumbnail { get; set; }

/// Provides access to this image's global metadata.
public ImageMetaData MetaData { get; set; }

/// Whether this codec supports images with more than one frame.
public bool SupportsMultiFrame { get; }

/// Copy the image from the specified decoder, including all the frames,
/// all the thumbnails, and all the metadata, including the per-frame
/// and global data. This does an implicit clear of any data that's already
/// been set in the encoder.
public void CopyImage(ImageDecoder decoder);

/// Save (encode) the image to the specified stream.
public void Save(System.IO.Stream destStream);
}
}

```

Un número de codificadores contruidos están provistos con el MIL, incluyendo el ImageEncoderBmp.cs, ImageEncoderGif.cs, ImageencoderJpeg.cs, ImageEncoderPng.cs, ImageEncoderTiff.cs, e ImageEncoderWmp.cs.

Las `ImageSizeOptions` son usadas para especificar el tamaño de miniaturas y el tamaño de las imágenes en cache. Las opciones incluyen el ancho, alto, sea para preservar la proporción de aspecto de la imagen original, y un ángulo de rotación (multiple de 90°).

```
namespace System.Windows.Media
```

```
{
```

```
    /// Sizing options for an image. The resulting image
```

SSS

```
/// will be scaled based on these options.
public class ImageSizeOptions
{
    /// The rotation to be applied; only multiples of 90 degrees is supported.
    public enum Rotation
    {
        /// Do not rotate
        Rotate0 = 0,

        /// Rotate 90 degrees
        Rotate90 = 90,

        /// Rotate 180 degrees
        Rotate180 = 180,

        /// Rotate 270 degrees
        Rotate270 = 270,
    }

    /// Construct an ImageSizeOptions object.
    /// Still need to set the Width and Height Properties.
    public ImageSizeOptions();

    /// Construct an ImageSizeOptions object.
    public ImageSizeOptions(
        bool preserveAspectRatio,
        int pixelWidth,
        int pixelHeight
    );

    /// Whether or not to preserve the aspect ratio of the original
    /// image. If so, then the PixelWidth and PixelHeight are
```

270


```
/// maximum values for the image size. The resulting image  
/// is only guaranteed to have either its width or its height  
/// match the specified values. For example, if you want to  
/// specify the height, while preserving the aspect ratio for  
/// the width, then set the height to the desired value, and  
/// set the width to Int32.MaxValue.  
///  
/// If we are not to preserve aspect ratio, then both the  
/// specified width and the specified height are used, and  
/// the image will be stretched to fit both those values.  
public bool PreserveAspectRatio { get; set; }  
  
/// PixelWidth of the resulting image. See description of  
/// PreserveAspectRatio for how this value is used.  
///  
/// PixelWidth must be set to a value greater than zero to be valid.  
public int PixelWidth { get; set; }  
  
/// PixelHeight of the resulting image. See description of  
/// PreserveAspectRatio for how this value is used.  
///  
/// PixelHeight must be set to a value greater than zero to be valid.  
public int PixelHeight { get; set; }  
  
/// RotationAngle to rotate the image. Only multiples of 90 are supported.  
public Rotation RotationAngle { get; set; }  
  
/// Whether the size options are valid. To be valid,  
/// both must be greater than zero, and at most one  
/// can be set to Int32.MaxValue.  
public bool IsValid { get; }  
}
```

- 246

Lo siguiente suministra una definición de formato de pixel para imágenes y superficies con base en píxeles:

```

namespace System.Windows.Media
{
    /// Pixel Format Definition for images and pixel-based surfaces
    public struct PixelFormat
    {
        /// Describes order of each channel of pixel data
        public enum ChannelDescription
        {
            /// Undefined channel description
            Undefined = 0,

            /// single channel indexed (palletized) formats
            Index = 1,

            /// single channel gray formats
            Gray = 2,

            /// red-green-blue
            RGB = 3,

            /// blue-green-red
            BGR = 4,

            /// alpha-red-green-blue
            ARGB = 5,

            /// Cyan-magenta-yellow-black

```

```
CMYK = 6
}

/// op_equality - returns whether or not the two pixel formats are equal
public static bool operator ==(PixelFormat left, PixelFormat right);

/// Equals - Returns whether or not the two pixel formats are equal
public static bool Equals(PixelFormat left, PixelFormat right);

/// op_inequality - returns whether or not the two pixel formats are not equal
public static bool operator !=(PixelFormat left, PixelFormat right);

/// Equals - Returns whether or not this is equal to the Object
public override bool Equals(Object obj);

/// GetHashCode - Returns a hash code
public override int GetHashCode();

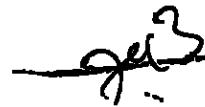
/// The number of bits per pixel for this format.
public int BitsPerPixel { get; }

/// Whether this format is a palletized (indexed) format.
public bool IsPalletized { get; }

/// Whether this format is a set of gray values.
public bool IsGray { get; }

/// Whether this format is CMYK values (cyan, magenta, yellow, black)
public bool IsCMYK { get; }

/// Whether this format is SRGB (Gamma is approximately 2.2)
public bool IsSRGB { get; }
```



```
/// Whether this format is Linear (Gamma is 1.0)  
public bool IsScRGB { get; }  
  
/// Whether this format has premultiplied alpha.  
public bool IsPremultiplied { get; }  
  
/// Describes the data of this format and its order.  
public ChannelDescription Channel { get; }  
  
/// Get the mask to use for obtaining the red channel for this format.  
/// Shift the mask by the leftShift amount to get the mask to use.  
public void GetRedMask(out uint mask, out int leftShift);  
  
/// Get the mask to use for obtaining the green channel for this format.  
/// Shift the mask by the leftShift amount to get the mask to use.  
public void GetGreenMask(out uint mask, out int leftShift);  
  
/// Get the mask to use for obtaining the blue channel for this format.  
/// Shift the mask by the leftShift amount to get the mask to use.  
public void GetBlueMask(out uint mask, out int leftShift);  
  
/// Get the mask to use for obtaining the alpha channel for this format.  
/// Shift the mask by the leftShift amount to get the mask to use.  
public void GetAlphaMask(out uint mask, out int leftShift);  
  
/// Get the masks to use for obtaining each channel for this format.  
/// Shift the masks by the leftShift amount to get the masks to use.  
public void GetRGBMasks(out uint redMask, out int redLeftShift,  
                        out uint greenMask, out int greenLeftShift,  
                        out uint blueMask, out int blueLeftShift);  
  
}
```

Jef

Cada codec (ImageEncoder e ImageDecoder) es requerido para suministrar un CodecInfo que da información acerca del codec, suministra métodos de casos de creación para su decodificador/codificador asociado, y suministra un método para determinar si este codec caza con un CodecFilter administrado.

```
namespace System.Windows.Media
{
    /// Information about a specific codec and a factory for creating
    /// the codec. This is returned from the codec enumerator.
    public abstract class CodecInfo
    {
        /// The number of bytes needed from the image header to
        /// determine if the image is supported by this codec.
        public abstract int RequiredHeaderSize { get; }

        /// Whether the codec supports this image, based on looking
        /// at the first RequiredHeaderSize bytes from the image. The header
        /// must contain at least the first RequiredHeaderSize bytes from the image.
        public abstract bool IsImageSupported(byte[] header);

        /// The Friendly name of a codec
        public abstract string FriendlyName { get; }

        /// Which Mime Types the codec supports.
        public abstract string[] MimeTypes { get; }

        /// Who authored the codec.
        public abstract string CodecAuthor { get; }

        /// The version number of the codec.
    }
}
```

```

public abstract System.Version CodecVersion { get; }

/// Whether there is a decoder associated with this codec.
public abstract bool HasDecoder { get; }

/// Whether there is an encoder associated with this codec.
public abstract bool HasEncoder { get; }

/// Whether this codec is one of the built-in ones.
public bool IsBuiltin { get; }

/// Determine if this codec matches the specified filter.
/// Note: this does NOT check/sniff the bytes in the stream.
public virtual bool MatchesFilter(CodecFilter filter);

/// Get an instance of the decoder associated with this codec (if there is one).
public abstract ImageDecoder CreateDecoderInstance(System.IO.Stream
    imageStream);

/// Get an instance of the encoder associated with this codec (if there is one).
public abstract ImageEncoder CreateEncoderInstance();
}
}

```

Los objetos CodecInfo contruidos allí son suministrados con el MIL, incluyendo CodecInfoBmp.cs, CodecInfoGif.cs, CodecInfoIcon.cs, CodecInfoJpeg.cs, CodecInfoPng.cs, CodecInfoTiff.cs, y CodecInfoWmp.cs.

El CodecFilter es usado por el enumerador de codec para enumerar codec con base en criterios especificados. Los criterios que no son especificados son ignorados cuando se mira por un apareamiento de codec. Por ejemplo, si el

746

MimeType no esta establecido, entonces los codecs con cualquier tipo mimo son considerados.

```
namespace System.Windows.Media
```

```
{
    /// Filter for enumerating codecs. Only those codecs that match the
    /// properties will be enumerated.
    public class CodecFilter
    {
        /// Find a codec that can handle this image stream (by sniffing the stream).
        public System.IO.Stream ImageStream { get; set; }

        /// The Friendly name of a codec
        public string FriendlyName { get; set; }

        /// Does the codec support this Mime Types.
        public string MimeType { get; set; }

        /// Does the codec match this author.
        public string CodecAuthor { get; set; }

        /// Is the codec's version >= this version.
        public System.Version MinVersion { get; set; }

        /// Is the codec's version <= this version.
        public System.Version MaxVersion { get; set; }

        /// Find those codecs that have a matching decoder. Setting this
        /// to false means that we don't filter on whether or not there is
        /// a decoder for that codec.
        public bool HasDecoder { get; set; }

        /// Find those codecs that have a matching encoder. Setting this
        /// to false means that we don't filter on whether or not there is
        /// an encoder for that codec.
        public bool HasEncoder { get; set; }
    }
}
```

563

```

/// Find those codecs that are built-in (not add-ins). Setting this
/// to false means that we don't filter on whether or not the
/// codec is one of the built-in ones.
public bool IsBuiltIn { get; set; }
}
}

```

Cuando el enumerador es construido (sus correctores son internos) esta dado un CodecFilter. Este filtro es usado para determinar que codec enumerar. Solamente aquellos que cacen con el filtro (si hay alguno) son enumerados.

```

namespace System.Windows.Media
{
/// The enumerator for Image frames.
public class ImageCodecEnumerator : IEnumerator
{
/// Reset - resets the position to before the first object in the collection.
/// A call to MoveNext must precede any call to Current after a Reset.
public void Reset();

/// MoveNext - Move to the next object in the collection. Returns false if the
/// enumerator
/// has passed the end of the collection
public bool MoveNext();

/// Current - returns the current object in the collection
public object Current { get; }
}
}

```

7/10

Esta basado en los valores llamados color. Las soluciones del llamado color permitirán que Pantone, Trumatch y otros colores llamados sean soportados con contextos de color. Esto también soportara los parámetros de canal alfa.

Una primera parada de gradiente es especificada usando un contexto de color ("sGay.icc 0,5"). La frase del texto especifica el nombre del archivo del contexto de color, que puede ser un URL o un URI. Esta muestra es para ilustrar el soporte limpio para impresión monocromática sin requerir que los valores RGB sean traducidos primero en valores monocromáticos en el tiempo de conversión de la impresión.

La segunda parada de gradientes especificada usando propiedades sRGB (a="0,8"R="0,2"G="1"B="0,2"). Esta representación es idéntica a como .NET encuadra los colores especificados V1 y WinForms.

La tercer parada de gradiente es usando un contexto de color ("mswopintent8.icm 0,9 0,2 0,1 0,3"). La frase del texto especifica el nombre del archivo del contexto de color, el cual puede ser un URL o URI y puede soportar parámetros de canal alfa. Esta ilustración muestra el soporte CMYK, tal como se requirió para Publisher u otras de tales aplicaciones.

505

Tomados juntos, estos ejemplos suministran una sintaxis muy robusta y limpia para requerimientos del color de soporte. Note que un contexto de color puede ser especificado globalmente y las referencias a color interno requieren conformarse a este contexto global, por ejemplo, para optimización de desempeño.

Como se describió anteriormente, un contexto de color es requerido para objetos de color, independientemente si ellos son basados en vectores o en trama. Aun nivel basto, el contexto de color puede ser pensado como el de un perfil, suministrando una relación entre el espacio de color del objeto de color y el sistema visual humano. El contexto de color suministra la información sobre la relación entre el espacio de color de usuario y el espacio de color sRGB (o color del sistema visual humano). Esto permite que el CMYK de "ida y vuelta" y otra información de color que no ha sido previamente factible de una manera efectiva.

En la práctica hoy, existen literalmente cientos de espacios de color diferentes tales como sRGB, scRGB, AdobeRGB, BruceRGB, AppleRGB, TomRGB, CorbisRGB, JoeRGB, HSV, HSB, XYZ, LAB, LUV, YUV, YCC, CMYK, CcMmYK, CMYKOG, escala de grises livianos, escala de grises de luminiscencia y muchos otros. Muchos de estos espacios de color individuales pueden ser abiertos en clases de espacios tales como espacios RGB que

KCB

son definidos principalmente por tres canales que contienen aproximaciones a la percepción visual de rojo, verde y azul con semánticas de definición que incluyen gama, primarios, punto blanco y precisión de bit. La precisión de bit es necesaria ya que bajas precisiones de bit (tales como 8 bit por canal) típicamente requieren compensaciones grandes y no lineales para lograr eficiencias de codificación razonable.

Suministrar una clase de contexto del color reduce el conjunto de clases de espacio de color posibles a un conjunto mucho más pequeño, tal como escala de grises, RGB, HSV, CMYK, LAB, LCH, CcMmYK, CMYKOG, colores espectrales y espacios de efectos especiales tales como duotono, tritono y cuadtono usados para efectos artísticos. Una reducción adicional es posible combinando espacios que comparten el mismo significado subyacente pero suministran diferentes sistemas coordinados de coordenadas (similar a geometrías rectilíneas y polares). Esto haría del HSV un método superior a la clase de espacio de color RGB y al LCH un método superior a la clase de espacio de color LAB.

También es posible combinar espacios de color de efectos especiales con color espectral e incluir soporte para CcMmYK y CMYKOG y tener el valor de color en justamente este contexto como un arreglo dinámico ya que solamente usuarios avanzados usaran esta característica. Una reducción adicional

567 367

es posible para reducir espacios de color a sRGB, los cuales soportaran espacios sRGB y otros espacios RGB, y un espacio de color multicanal con un contexto de color. Esto deja un número razonable de clases de espacio de color básicas para soportar, incluyendo solamente espacios sRGB y multicanal.

El `ColorContext` esta asociado con una clase de color de gráficos de vector o un objeto `ImageData`. Otra alternativa es limitar `Visuals` a un `ColorContext` único. Esto ayudaría a optimizar la cantidad de conversiones de color y de validaciones de color en muchas circunstancias y puede ser más natural para los desarrolladores de aplicaciones, por ejemplo, un control individual no le gustaría usar colores de sistemas múltiples de color. Note que el perfil de color todavía se permite cambiar para aplicaciones avanzadas que explícitamente usan tipos múltiples de color por medio de un mecanismo que permite cambios dinámicos. Un `ColorContext` también permite la conversión de las gamas de color o intención para el mapeo entre dos gamas de dispositivos que son especificados y por lo tanto asociados con el objeto de color. Ya que el `ColorContext` solamente trata con un objeto de color único, la gama de designación puede ser un dispositivo virtual. Esto permite que el `ColorContext` contenga tanto la descripción objetiva del espacio de color lo mismo que la intención de conversión subjetiva para el color.

GSD

Los nombres de color son tablas simples de búsqueda enbedidas en el perfil que está asociado con el contexto de color que suministra un enlace entre un valor de color con base en el tipo de ColorContext y el nombre de color real. Esto permite que diferentes diccionarios de nombramiento de color para cada objeto de color. Por ejemplo, es posible asociar un tipo de sistema de nombramiento, tal como Trumatch con objetos de color de proceso y otro tipo de sistema de nombramiento, tal como Pantone con objetos de color de punto.

Los tipos de color de gráficos de vector públicos casan con el sistema de nivel inferior para optimizar el desempeño minimizando las conversiones cuando se transmiten datos a un sistema de más bajo nivel. Un conjunto "nativo" (es decir CMYK o similares) de valores de punto flotante son almacenado en el contexto de color y es sincronizado cuando ocurre cualquier cambio. El ColorContext color Value nativo es un tipo de valor (Estructura) basado en un arreglo de flotantes para poder soportar transparentemente escalas de grises, RGB y aún espacios de color CMYK. El arreglo ColorContext ColorValue nativo debería ser dinámico y no estar limitado a 1, 3, 4 o aún 8 canales de color. Esto permite que soluciones de proceso de color espectral o puenteado con las mismas arquitecturas. Note que aunque el costo de la asignación es relativamente medianamente significativo al costo del arreglo

de 5 elementos que solamente usado parcialmente, asegura una solución consistente, coherente, y flexible para el futuro, y no hay ningún costo nulo cuando los flojos de trabajo sRGB son usados. El valor alfa es separado del ColorContext ColorValue, debido a que es un concepto diferente y es tratado diferentemente en la mayoría de los usos.

NOMBRE	TIPO	DESCRIPCIÓN	OTRA INFORMACIÓN
InternalColor	Rojo flotante, verde flotante, azul flotante, alfa flotante	Estructura idéntica a la estructura interna de conversión de color para optimizar desempeño	Estructura interna, esta segunda estructura interna es usada para soportar un formado eficiente de los datos
Context	ColorContext	El contexto color que suministra métodos relacionados para usar información de contexto de color	Lleva valores de color nativos que son sincronizados a valores InternalColor

NOMBRE	ARGUMENTOS	DESCRIPCIÓN	OTRA INFORMACIÓN
FromProfile (...)	Frase	ICC u otro nombre de archivo de perfil con base en constructor	Estática pública
FromProfileAndRenderingIntent (...)	Frase, frase	ICC u otro nombre de archivo de perfil con base	Estática pública

		en constructor	
FromAValues (...)	Flotante, Flotante[], nombre de archivo	Constructor general con base en valor del canal alfa, arreglo de valores de punto flotante y un ICC u otro nombre de archivo de perfil	Estática pública
FromValues	Flotante[], nombre de archivo	Igual a FromAValues (...) pero el alfa se asume que es 1,0f	Estática pública
FromARGB (...)	Byte, byte, byte, byte	Constructor de herencia sRGB con base en valores alfa, rojo, verde, y azul sRGB	Estática pública, los valores sRGB son convertidos internamente a scRGB para el procesamiento
FromRGB (...)	Byte, byte, byte	El constructor de herencia sRGB con base en rojo, verde y azul valores sRGB (alfa es asumido como 1,0f)	Estática pública, los valores sRGB son convertidos internamente a scRGB para el procesamiento
FromscRGB (...)	Flotante, flotante, flotante, flotante	El constructor scRGB con base en valores alfa, rojo, verde y azul scRGB	Estática pública

Note que obtener el valor del color del sistema de objetos del color UI puede ser dependiente de otros contextos, ya que otros tipos mas avanzados de temática, y deben ser

recolectados con otros sistemas métricos con la aplicación modelo/temática API.

NOMBRE	TIPO DE RETORNO	DESCRIPCIÓN	OTRA INFORMACIÓN
R	Byte	El valor rojo sRGB del componente rojo scRGB del color actual	R/W
G	Byte	El valor verde sRGB del componente rojo scRGB del color actual	R/W
B	Byte	El valor azul sRGB del componente rojo scRGB del color actual	R/W
A	Byte	El valor alfa sRGB del componente rojo scRGB del color actual	R/W
ScR	Flotante	El valor rojo scRGB del componente rojo scRGB del color actual	R/W
ScG	Flotante	El valor verde scRGB del componente rojo scRGB del color actual	R/W
ScB	Flotante	El valor azul scRGB del componente rojo scRGB del color actual	R/W
ScA	Flotante	El valor alfa scRGB del componente rojo scRGB del color actual	R/W

Note que los valores scRGB pueden variar por debajo de 0,0 y por encima de 1,0 para soportar rangos y ramas dinámicas extendidas.

El operador pasa por encima de los objetos Color que están en contexto específico, por que el mezclar y agregar colores es dependiente del espacio de color. Por ejemplo, los espacios

572

de iluminancia RGB son aditivos y lineales, de modo que con las operaciones típicas matemáticas son medianamente intuitivas, pero los espacios iluminación RGB lo mismo que los espacios CMYK no son ni lineales ni aditivos, por lo tanto esta operación resulta en diferentes efectos visuales. Adicionalmente, la , mayoría de las operaciones color pueden resultar en valores más haya de la gamma deseada y por lo tanto requieren compensación de mapeo de gamma. Esto puede ser tan simple como sujeción de baja calidad o puede ser significativamente más sofisticado.

Algún operador de animación que pasa por encima puede ser suministrado si ellos están limitados específicamente al ColorContext scRGB ya que el scRGB está basado en luz física y se mezcla lineal y aditivamente.

El CMYK, scRGB y otros espacios de color son modelos de mezclado muy diferentes.

NOMBRE	TIPO DE RETORNO	ARGUMENTO	DESCRIPCIÓN	OTRA INFORMACIÓN
+	Color	Color, color	Contexto dependiente de la adición del color	El contexto RGB es lineal con respecto a la luminancia (como se mezclan fotones) lo que es correcto para los gráficos scRGB y

605

				gráficos de computador.
Add	Color	Color, color	Contexto dependiente de la adición del color	Público
-	Color	Color, color	Contexto dependiente de la resta de color	Público
Subtract	Color	Color, color	Contexto dependiente de la resta de color	Público
*	Color	Color, flotante	Contexto dependiente multiplicar el color por un valor de punto flotante	Público
Multiply	Color	Color, flotante	Contexto dependiente multiplicar el color por un valor de punto flotante	Público
Equals	Bool	Color, color	Retorno verdadero si dos valores de color son iguales.	Público
Equals	Bool	Objeto	Retorno verdadero si objeto de color es igual al color actual	Público
=	Bool	Color, color	Retorno verdadero si dos valores de color son iguales.	Público
IsEqual	Bool	Color, color	Retorno verdadero si dos valores de color son iguales.	Público
!=	Bool	Color, color	Retorno verdadero si dos valores de color no son iguales.	Público
IsNotEqu al	Bool	Color, color	Retorno verdadero si dos valores de	

			color no son iguales.	

Métodos similares como los usados en colores multicanal también pueden ser usados para soportar HSB, YCC y YUV y espacios de color similares que están relacionados cercanamente a sRGB o scRGB.

Nombre	Tipo de retorno	Argumento	descripción	Otra información
Clamp	Color	Vacío	Los valores fijación de color en el rango [0,0...1,0]	Publico
GetHashCode	Entero	Vacío	Retorno de código de fijado de color	Publico
Name	expresión	Retorno de nombre de color	Redundante con la llamada color context	Publico
ToRgbColor	Ninguno	Color	Retorno de color equivalente scRGB del color actual	Publico
ToString	Expresión	Vacío	Retorno valor de expresión formateada del color	Publico
AreClose	Bool	Color, color	Retorno verdadero si los valores de color son	Estático

915

			cercanos usando funciones FloatUtil	
IsClose	Bool	Color	Retornos verdaderos si el color es cercano al color actual usando la función FloatUtil	Publico
SetRenderingIntent	Bool	Expresión	Retornos verdaderos si la intención de conversión para color context es exitosa.	publico

Los valores flotantes en las formas rgb y argb con base sRGB son especificadas en una escala desde 0,0 a 1,0. por definición, estos valores nunca estarán por fuera de este rango y así deben ser fijados. De otro lado, los valores con base scRGB son válidos por debajo de 0,0 y por encima de 1,0. Estos valores deberían ser recortados solo si el dispositivo de destino no puede soportar la gamma de color extendida. Esto puede ser determinado preguntando por el perfil asociado con la gamma de destino. Idealmente, para pantallas, el hardware de gráficos puede hacerse cargo de este problema usando la funcionalidad de administración de gamma DX.

658

5/6

Si la expresión que está siendo analizada es inválida, el color será inicializado para ARGB = (0,0, 0,0, 0,0, 0,0).

Cuando se persiste, el valor será escrito como un nombre de color conocido si el color fue creado como un color conocido, sino, entonces la forma RGB (flotante, flotante, flotante,) será usada si alfa es 1,0. Si alfa no es 1,0 entonces la forma ARGB (flotante, flotante, flotante, flotante) será usada.

Los gráficos de cuadrícula o trama o los formatos de visualización o de píxeles son diferentes comparados con las soluciones de gráficos de vectores anteriores. Dicho simplemente, la entrada formación de imágenes puede ser cercanamente a cualquier cosa desde 1 bpp hasta 128 bpp con una variedad de espacios de colores soportados desde negro y blanco a sRGB a scRGB a CMYK. Por lo tanto, la solución de color para ImageData requiere un ColorContext para ImageData o formatos de píxeles. Esto puede ser generado a partir de perfiles en embebidos o gamma embebida e información de cromaticidad dentro los formatos de archivo estándar. Esto elimina la necesidad de suministrar gamma u otras propiedades redundantes o campos en la clase ImageData.

El código MILRender entiende especificaciones de color de 32-bpc (scRGB. La conversión de la entrada de color debería

5/6

57

suceder arriba del código de conversión (pero no necesariamente por fuera del código no manejado).

La animación de valores de color debería principalmente ocurrir en un espacio lineal. Podría ser un espacio HSB linearizado o un scRGB, pero debe ser lineal para "rebote" u otras animaciones por canal para tener sentido.

Tres modos de fidelidad de color son descritos, incluyendo:

32bpc completo por el sistema; memoria de respaldo de 128bpp/ memoria principal 10-bpc +; composición de 32bpc completa.

Hibrido- especificaciones de color de 32bpc/interpolación; colores de oscilación o fijación para un pre-compuesto de 32bpp.

Herencia -especificación de color de 32bpc-convertido inmediatamente a 32bpp; compuesto/ salida de 32bpp.

Estos modos serán soportados con dos formatos de memoria de respaldo, particularmente 1,0 gamma de 128bpp (scRGB) y 2,2, gamma de 32bpp (sRGB). El soporte también se puede suministrar para manejar baja profundidad bit (pantallas de 16 y 8bpp) en escenarios de memoria principal. La memoria de

60

respaldo es oscilada sobre Present para impedir perdida en la composición.

La clase de geometría de los objetos puede ser usada para recortar, ensayo de golpe, y conversión de datos con base en vectores 2D con el Pen y el Brush. Las clases Geometry derivadas suministran más semánticas de construcción y enumeración específicas. Un número de tipos de Geometry de forma específica son suministrados, lo mismo que una PathGeometry generalizada que permite la definición explícita de una Geometry de forma más compleja. Para aquellos familiares con GDI+ es más similar a GraphicsPath.

Libro de texto	GDI+	Presente Invención
Path	GraphicsPath	PathGeometry
SubPath	GraphicsPath	PathFigure

La geometría es una clase de base abstracta.

```
[TypeConverter(typeof(PathGeometryConverter))]  
public abstract class Geometry: Animatable, IDposable  
{  
    Internal Geometry ();
```

```

public new Geometry Copy();

// If animated, Bounds returns the "current" bounds
// This does not take into account any pen
public virtual Rect Bounds { get; }

// This takes the pen into account. When animations are present
// this takes the "current" values of both the geometry and pen.
public virtual Rect GetBounds(Pen pen);

// Returns identity if no transformation.
public Transform Transform { get; set; }

// Release resources cached by Geometry (tessellations, path data, etc)
public virtual void Dispose();
}

```

Para establecer una transformación que es aplicada a la geometría, establezca la propiedad Transform. Una GeometryCollection es una colección de objetos Geometry que han sido combinados usando operaciones booleanas particulares sobre sus áreas definidas. Este objeto permite unas combinaciones de construcción visual más fáciles de los objetos Geometry de lo que es posible usando estrictamente objetos PathFigure dentro de una PathGeometry.

Numeración del modo de combinación dirige la combinación del área de geometría en la colección. Las operaciones Booleanas Union, XOR, Intersect son conmutativas y por lo tanto aplican independientemente del orden a las geometrías. Complement y Exclude no son conmutativas y por lo tanto son definidas entre la primera geometría y la geometría restante individual. En otras palabras, una combinación de excluir de {g1, g2, g3} sería aplicada como ((g1 exclude g2) y (g1 exclude g3)). El Complement especifica que la región existente es reemplazada por el resultado de la región que existe que esta siendo removida de una nueva región. En otras palabras, la región existente es excluida de la nueva región. Exclude especifica que la región existente es reemplazada por el resultado de la nueva región que esta siendo removida de la región existente. En otras palabras, la nueva región es excluida de la región existente. Intersect se refiere a combinar regiones por medio de tomar sus intersecciones, Union se refiere a combinar regiones tomando la unión de ambos, y XOR se refiere a combinar regiones tomando solamente las áreas encerradas por una u otra región, pero no de ambas:

```
public enum System.Windows.CombineMode
```

```
{  
    Complement,  
    Exclude,  
    Intersect,  
    Union,  
    Xor  
}
```

```
public sealed class GeometryCollection : Geometry, IAddChild, IList
```

```
{  
    public GeometryCollection();  
    public GeometryCollection(int capacity);  
    public GeometryCollection(  
        CombineMode combineMode,  
        params Geometry[] collection);  
    public GeometryCollection(  
        CombineMode combineMode,  
        ICollection collection);  
    public GeometryCollection(  
        CombineMode combineMode,  
        ICollection collection,  
        Transform transform);  
    public new GeometryCollection Copy();  
  
    // Specify how to combine geometry  
    [DefaultValue(CombineMode.Union)]
```

```
public CombineMode CombineMode { get; set; }

// IEnumerable
public IEnumerator GetEnumerator();

// ICollection
public int Count { get; }
public bool IsSynchronized { get; }
public object SyncRoot();
public void CopyTo(Array array, int index);

// IList
public bool IsFixedSize { get; }
bool IList.IsReadOnly { get; }
object IList.this[int index] { get; set; }
int IList.Add(object value);
public void Clear();
bool IList.Contains(object value);
int IList.IndexOf(object item);
void IList.Insert(int index, object value);
void IList.Remove(object value);
public void RemoveAt(int index);

// Extra functions
public int Capacity { get; set; }
public Geometry this[int index] { get; set; }
public int Add(Geometry geometry);
public bool Contains(Geometry geometry);
public int IndexOf(Geometry geometry);
public int IndexOf(Geometry geometry, int startIndex);
public int IndexOf(Geometry geometry, int startIndex, int count);
public int LastIndexOf(Geometry geometry);
```

```

public int LastIndexOf(Geometry geometry, int startIndex);
public int LastIndexOf(Geometry geometry, int startIndex, int count);
public void Insert(int index, Geometry geometry);
public void Remove(Geometry geometry);
public void AddRange(GeometryCollection geometryCollection);
public void SetRange(int index, GeometryCollection geometryCollection);
public void InsertRange(int index, GeometryCollection geometryCollection);
public void RemoveRange(int index, int count);
public GeometryCollection GetRange(int index, int count);

public Geometry GetOptimizedGeometry();
}

```

La `GetOptimizedGeometry ()` colapsa la colección de geometrías en donde sea posible, el resultado no necesariamente es una `GeometryCollection`. Esto puede influir optimizaciones tales como combinar la geometría de rectángulo adyacente en una geometría de rectángulo única, llevando a cabo una operación booleana entre la geometría de trayecto adyacente para crear una nueva geometría de trayecto, o aplanar una `GeometryCollection` con el mismo modo de combinación. En situaciones en donde la geometría es usada en muchos diferentes contextos esto suministra una mejora de desempeño en el procesamiento y almacenamiento.

El siguiente ejemplo demuestra la marca que usa una `GeometryCollection`:

```

<Path Fill="Red">
  <Path.Data>
    <GeometryCollection Transform="translate(425 25)" CombineMode="Xor">
      <RectangleGeometry Rect="0 0 100 100" />
      <RectangleGeometry Rect="50 50 100 100" />
    </GeometryCollection>
  </Path.Data>
</Path>

```

La PathGeometry es una colección de objetos PathFigure. Cada uno de los objetos PathFigure está compuesto de uno o más objetos PathSegment los cuales definen realmente su forma. El área de llenado del PathGeometry está definida por medio de tomar los PathFigures contenidos que tienen su propiedad Filled establecida en verdadero y aplicar la FillRule para determinar el área encerrada. La figura 13 representa las relaciones de objetos PathGeometry.

La enumeración FillRule especifica como las áreas de intersección de los objetos Figure contenidos en una Geometry son combinados para formar el área resultante de la Geometry:

585

```
public enum System.Windows.Media.FillRule
{
    EvenOdd,
    NonZero
}

{
    public PathGeometry();
    public PathGeometry(params PathFigure[] figures);
    public PathGeometry(ICollection figureCollection);
    public PathGeometry(
        ICollection figureCollection,
        FillRule FillRule,
        Transform transform);
    public new PathGeometry Copy();

    [DefaultValue(FillRule.EvenOdd)]
    public FillRule FillRule { get; set; }

    // This method will add the Figures from the specific
    // Geometry to this PathGeometry, but will use the
    // current FillRule instead of the geometry's FillRule.
    public void AddGeometry(Geometry geometry);
    public void AddFigure(PathFigure figure);
    public void AddPointAndTypes(
```

666

```

    Point[] points,
    byte[] types);

    public PathFigureCollection Figures { get; set; }

    public PathGeometry GetWidenedPathGeometry(Pen pen);
    public PathGeometry GetOutlinedPathGeometry();
}

```

La regla EvenOdd determina "interiorización" de un punto en el cuadro por medio de dibujar un rayo desde ese punto hacia infinito en cualquier dirección y entonces examinar los lugares en donde el segmento de la forma cruza el rayo. Iniciando con una cuenta de cero, agregar uno cada vez a Segment cruza el rayo desde izquierda a derecha y restar uno cada vez que un segmento de trayecto cruza el rayo desde derecha a izquierda. Después de contar los cruces, si el resultado es cero entonces el punto esta por fuera del trayecto. De otra manera, está adentro.

La regla NonZero determina la "interioridad" de un punto en el cuadro por medio de dibujar un rayo desde ese punto hasta infinito en cualquier dirección y contar el numero de Segments de trayecto desde la forma dada que cruce el rayo. Si este número es impar, el punto esta adentro; si es par, el punto esta afuera.

587

Para convertir otros tipos de geometría en una geometría de trayecto por la inclusión con otras figuras, el método `AddGeometry` es usado. Este agrega una figura que caza geométricamente con la geometría de entrada. Para una geometría no animada el cace es exacto, mientras que la geometría animada puede ser disipativa en la conversión. La razón para esta conversión disipativa es que los parámetros animados de la geometría de entrada no cazan con la forma que se ajusta a un segmento.

Geometría	Sintética/Disipativo	Figuras
<code>LineGeometry</code>	Disipativo con animación	<code>StartSegment</code> and <code>LineSegment</code>
<code>RectangleGeometry</code>	Disipativo con animación	<code>PolyLineSegment</code>
<code>EllipseGeometry</code>	Disipativo con animación	<code>ArcSegment</code> and <code>ArcSegment</code>
<code>GeometryCollection</code>	Disipativo con animación	Muchos clases de segmentos
<code>PathGeometry</code>	Disipativo con animación	Todos excepto los segmentos <code>Arc</code> y <code>Quadratic</code>

200

58

La enumeración y estructura del PathGeometry resultante no está garantizado que cace con la geometría de entrada exactamente.

Una colección de figuras, es una colección de objetos PathFigure y los contenidos principales que definen una PathGeometry.

```
public sealed class PathFigureCollection : Animatable, IAddChild, IList
{
    public PathFigureCollection();
    public PathFigureCollection(int capacity);
    public PathFigureCollection(params PathFigure[] figures);
    public PathFigureCollection(ICollection figureCollection);
    public new PathFigureCollection Copy();

    // IEnumerable
    public IEnumerator GetEnumerator();

    // ICollection
    public int Count { get; }
    public bool IsSynchronized { get; }
    public object SyncRoot;
    public void CopyTo(Array array, int index);

    // IList
    public bool IsFixedSize { get; }
    bool IList.IsReadOnly { get; }
    object IList.this[int index] { get; set; }
    int IList.Add(object value);
    public void Clear();
    bool IList.Contains(object value);
    int IList.IndexOf(object item);
    void IList.Insert(int index, object value);
}
```

gel

```

void IList.Remove(object value);
public void RemoveAt(int index);

// Extra functions
public int Capacity { get; set; }
public PathFigure this[int index] { get; set; }
public int Add(PathFigure figure);
public bool Contains(PathFigure figure);
public int IndexOf(PathFigure figure);
public int IndexOf(PathFigure figure, int startIndex);
public int IndexOf(PathFigure figure, int startIndex, int count);
public int LastIndexOf(PathFigure figure);
public int LastIndexOf(PathFigure figure, int startIndex);
public int LastIndexOf(PathFigure figure, int startIndex, int count);
public void Insert(int index, PathFigure figure);
public void Remove(PathFigure figure);
public void AddRange(PathFigureCollection figureCollection);
public void SetRange(int index, PathFigureCollection figureCollection);
public void InsertRange(int index, PathFigureCollection figureCollection);
public void RemoveRange(int index, int count);
public PathFigureCollection GetRange(int index, int count);
}

```

Un PathFigure es una subsección de una Geometry que define una colección de segmentos. Esta colección de segmentos es una serie conectada única de objetos bidimensionales PathSegment. La PathFigure puede ser una forma cerrada con un área definida, o unas series conectadas de Segments que definen una curva, pero no un área encerrada. La clase PathFigure incluye un número de funciones de conveniencia

para permitir una construcción más simple de un PathFigure a partir de el método ArcTo/Linito/ (y otros) explícitos sin requerir un objeto PathSegment la llamada AddSegment explícita puede ser usada para agregar un Segment compuesto.

```

public sealed class PathFigure : Animatable
{
    public PathFigure();
    public PathFigure(params PathSegment[] segments);
    public PathFigure(ICollection segmentCollection);
    public PathFigure(ICollection segmentCollection, bool isFilled);
    public new PathFigure Copy();

    [DefaultValue(true)]
    public bool IsFilled { get; set; }

    // Start the figure at an initial point.
    public void StartAt(Point pt);
    public void StartAt(Point pt, PointAnimationCollection ptAnimations);

    // Adds a line segment from the current point to the specified
    // new point. The current point is updated to the new location.
    public void LineTo(Point pt);
    public void LineTo(Point pt, PointAnimationCollection ptAnimations);

    // Adds a polyline starting at the current point, then to
    // each point in the points array. The current point is
    // updated to the last point of the array.
    public void PolyLineTo(Point[] points);

    public void ArcTo(
        Point pt,
        Size sz,
        double xRotation,
        bool largeArcFlag,
        bool sweepFlag);

```

```

public void ArcTo(
    Point pt,
    PointAnimationCollection ptAnimations,
    Size sz,
    SizeAnimationCollection szAnimations,
    double xRotation,
    DoubleAnimationCollection xRotationAnimations,
    bool largeArcFlag,
    bool sweepFlag);

// Adds a cubic bezier segment using the current point as the
// first control point. The current point is updated to the
// the destination point.

```

```

public void BezierTo(
    Point pt1,
    Point pt2,
    Point ptDest);

```

```

public void BezierTo(
    Point pt1,
    PointAnimationCollection pt1Animations,
    Point pt2,
    PointAnimationCollection pt2Animations,
    Point ptDest,
    PointAnimationCollection ptDestAnimations);

```

```

// Adds cubic Bezier sections using the current point as the
// first control point, then iterating through 3 points at a
// time. The current point is updated to the last point of
// the array. When the number of points passed in is not a
// multiple of 3, we throw an invalid arg exception.

```

```

public void PolyBezierTo(Point[] points);

```

```

// Adds a quadratic Bezier section using the current point
// as the first control point. The current point is updated to
// destination point.
public void QuadraticBezierTo(Point pt1, Point ptDest);

public void QuadraticBezierTo(
    Point pt1,
    PointAnimationCollection pt1Animations,
    Point ptDest,
    PointAnimationCollection ptDestAnimations);

// Adds quadratic Bezier sections using the current point
// as the first control point, then iterating through 2 points
// at a time. The current point is updated to the last
// point of the array. When the number of points passed in is
// not a multiple of 2, we throw an invalid arg exception.
public void PolyQuadraticBezierTo(Point[] points);

// Close the figure. No new segments can be added.
public void CloseFigure();

// Stroke the segments of new figure. Default is true.
public void StrokeNewSegments(bool strokeNewSegments);

// Add a new Segment to the PathFigure
public void AddSegment(PathSegment segment);
public PathSegmentCollection Segments { get; set; }

public PathFigure GetFlattenedPathFigure(float flatness);
}

```

Una figura requiere un punto de inicio, ya que cada segmento mantiene la continuidad relativa al último punto agregado. Para especificar un punto de inicio se llama `StartSegment (pt)`. O `Add (nuevo StartSegment (pt))`. Después de agregar los segmentos, para agregar un segmento cerrado apropiadamente que una el último punto de nuevo con el punto de inicio, se usa `CloseFigure()` o `Add (nuevo CloseSegment())`. Los segmentos de inicio y cierre aparecen en la colección de segmentos.

Una excepción es lanzada si un `PathFigure` es construido y el `StartSegment` no es el primer segmento en la colección o el `CloseSegment`, si esta presente, no es el último segmento en la colección. El `StartSegment` y el `CloseSegment` no son válidos en cualquiera otras posiciones en la figura, una excepción es una colección de segmentos vacíos completamente.

La propiedad `PathFigure.IsFilled` explícitamente controla si o no el área contenida de una figura cerrada a es o va a ser usada para un tallo de golpes, conversión, y recorte. Si esta propiedad se establece en falso, entonces solamente el esquema del `PathFigure` será usado, y su área contenida no contribuirá al área general del `PathGeometry`. El valor por omisión para esta propiedad es verdadero.

544

Para enumerar los contenidos de un PathFigure como puntos, una manera directa es aplanar la figura y examinar el PathSegmentCollection resultante. El proceso de aplanamiento es disipativo con respecto a los parámetros de segmento de animación y curva, sin embargo el dato refinado esta expuesto por PolyLineSegment para un procesamiento de punto adicional.

Un PathSegmentCollection es una colección de objetos PathSegment y en los contenidos principales definen un Pathfigure:

```
public sealed class PathSegmentCollection : Animatable, IAddChild, IList
{
    public PathSegmentCollection();
    public PathSegmentCollection(int capacity);
    public PathSegmentCollection(params PathSegment[] segments);
    public PathSegmentCollection(ICollection segments);
    public new PathSegmentCollection Copy();
}
```

702

```
// IEnumerable
public IEnumerator GetEnumerator();

// ICollection
public int Count { get; }
public bool IsSynchronized { get; }
public object SyncRoot();
public void CopyTo(Array array, int index);

// IList
public bool IsFixedSize { get; }
bool IList.IsReadOnly { get; }
object IList.this[int index] { get; set; }
int IList.Add(object value);
public void Clear();
bool IList.Contains(object value);
int IList.IndexOf(object item);
void IList.Insert(int index, object value);
void IList.Remove(object value);
public void RemoveAt(int index);

// Extra functions
public int Capacity { get; set; }
public PathSegment this[int index] { get; set; }
public int Add(PathSegment segment);
public bool Contains(PathSegment segment);
public int IndexOf(PathSegment segment);
public int IndexOf(PathSegment segment, int startIndex);
public int IndexOf(PathSegment segment, int startIndex, int count);
public int LastIndexOf(PathSegment segment);
public int LastIndexOf(PathSegment segment, int startIndex);
public int LastIndexOf(PathSegment segment, int startIndex, int count);
```



```
public void Insert(int index, PathSegment segment);  
public void Remove(PathSegment segment);  
public void AddRange(PathSegmentCollection segmentCollection);  
public void SetRange(int index, PathSegmentCollection segmentCollection);  
public void InsertRange(int index, PathSegmentCollection segmentCollection);  
public void RemoveRange(int index, int count);  
public PathSegmentCollection GetRange(int index, int count);
```

Un `PathSegment` representa una sección del esquema de un `PathFigure`. Los segmentos de líneas rectas simples, segmentos de arcos elípticos, segmentos cúbicos de bezier, y los segmentos cuadráticos de bezier pueden ser combinados juntos para formar un `PathFigure`.

547

```
public abstract class PathSegment : Animatable
{
    internal PathSegment();
    public new PathSegment Copy();

    [DefaultValue(true)]
    public bool IsStroked { get; set; }
}

public sealed class StartSegment : PathSegment
{
    public StartSegment();
    public StartSegment(Point point);
    public StartSegment(Point point, PointAnimationCollection pointAnimations);
    public new StartSegment Copy();

    [Animations("PointAnimations")]
    public Point Point { get; set; }
    public PointAnimationCollection PointAnimations { get; set; }
}
```

2/2

asegura que el ViewPort completo es llenado por el ViewBox (potencialmente causando porciones del ViewBox que queden por fuera del ViewPort). Si el área ViewBox esta vacía, entonces ningún Stretch será aplicado. El Alignment todavía ocurrirá, y este posicionará el "punto" del ViewBox.

Una vez que el ViewPort esta determinado (con base en el ViewPortUnits) y el tamaño de destino del ViewBox esta determinado (con base en Stretch), el ViewBox requiere estar posicionado dentro del ViewPort. Si el ViewBox es el del mismo tamaño que el ViewPort (si Stretch es Fill, o si solamente sucede que ocurre con uno de los otros tres valores Stretch), entonces el ViewBox es posicionado en el Origin de modo que es idéntico al ViewPort. Si no, entonces se consideran los HorizontalAlignment y VerticalAlignment. Con base en estas propiedades, el ViewBox es alineado tanto en las dimensiones X y Y. Si el HorizontalAlignment is Left, entonces el borde izquierdo del ViewBox será posicionado en el borde Left del ViewPort. Si este es center, entonces el centro del ViewBox será posicionado en el centro del ViewPort, y si es Right, entonces en los bordes derechos casaran. El proceso es repetido para la dimensión Y.

Si el ViewBox es Empty entonces se considera no establecido. Si está como no establecido, entonces el ContentUnits son considerados. Si el ContentUnits es Absolute, no hay puesta a

escala o desplazamiento que ocurra, y los contenidos son dibujados dentro del ViewPort sin ninguna transformación. Si los ContenUnits son RelativeToBoundingBox, entonces el contenido origen es alineado con el ViewPortOrigin, y los contenidos son puestos a escala por el ancho y el alto de la caja de límites del objeto.

Cuando se llena un espacio con un TileBrush, los contenidos son mapiados dentro del ViewPort como se mencionó anteriormente, y recortados al ViewPort. Esto forma la baldosa base para el llenado, y el resto del espacio es llenado con base en el TileMode del Brush's. Si esta establecido, la transformación del Brush es aplicada, lo que ocurre después de otro mapeo, puesta en escala, y desplazamiento, Etcétera.

Un VisualBrush es un TileBrush cuyos contenidos están especificados por un Visual. Este Brush puede ser usado para crear patrones complejos, o puede ser usado para dibujar copias adicionales de los contenidos de otras partes de la escena.

```

public sealed class System.Windows.Media.VisualBrush : TileBrush
{
    public VisualBrush(); // initializes to transparent
    public VisualBrush(Visual v);

    public new VisualBrush Copy(); // hides TileBrush.Copy()

    // Visual - Default is null (transparent Brush)
    public Visual Visual { get; set; }
}

```

Como se mencionó, un VisualBrush no tiene sintaxis simple para la marca, aunque puede ser descrito por medio de sintaxis compleja.

Un DrawingBrush es un TileBrush cuyos contenidos están especificados por un Drawing. Este Brush puede ser usado para crear patrones complejos que han sido creados por medio de un DrawingContext.

```
public sealed class System.Windows.Media.DrawingBrush : TileBrush
{
```

```
    public DrawingBrush(); // initializes to transparent
    public DrawingBrush(Drawing drawing);

    public new DrawingBrush Copy(); // hides TileBrush.Copy()

    // Drawing - Default is null (transparent Brush)
    public Drawing Drawing { get; set; }
}
```

Como se mencionó, un DrawingBrush no tiene sintaxis simple para la marca, aunque puede ser descrito por medio de una sintaxis compleja.

ImageBrush es un TileBrush que tiene contenidos especificados por un ImageSource. Este Brush puede ser usado para llenar un espacio con una imagen.

```

public sealed class System.Windows.Media.ImageBrush : TileBrush
{
    public ImageBrush(); // initializes to transparent contents

    // Sets the image, sets ViewBox to (0,0,Width,Height)
    // and Stretch to Fill
    public ImageBrush(ImageData image);

    public new ImageBrush Copy(); // hides TileBrush.Copy()

    // Default is null
    public ImageSource ImageSource { get; set; }

    // Default is true
    // If this is true, the ViewBox property will be overridden
    // and effectively set to the native size of the Image
    public bool SizeViewBoxToContent { get; set; }
}

```

La sintaxis simple para la marca del ImageBrush:

Image-brush:

"image" image-uri

El VideoBrush es un Tilebrush que tiene contenidos especificados por un VideoData. Este Brush puede ser usado para llenar un espacio con un video.

```

public sealed class System.Windows.Media.VideoBrush : TileBrush
{
    public VideoBrush(); // Initializes to transparent contents

    // Sets the image, sets ViewBox to (0,0,Width,Height) and the
    // Stretch to Fill
    public VideoBrush(VideoData video);

    public new VideoBrush Copy(); // hides TileBrush.Copy()

    // Default is null
    public VideoData VideoData { get; set; }

    // Default is true
    // If this is true, the ViewBox property will be overridden
    // and effectively set to the native size of the Video
    public bool SizeViewBoxToContent { get; set; }
}

```

La sintaxis simple para la marca VideoBrush:

Video-brush:

"Video" video-uri

NineGridBrush es un Brush que siempre llena la caja de límites del objeto con su imagen de contenido, y el estiramiento de la imagen no se lleva a cabo puramente por medio de una escala visual. La fuente de la imagen esta

dividida en 9 rectángulos por 4 bordes (por lo que se llama NineGrid). Los contenidos de la imagen en cada una de aquellas nueve regiones esta puesta a escala en 0,1 0 2 dimensiones hasta que se llena la caja del límite del objeto. Las dimensiones en las cuales cada sección esta puesta en escala pueden verse en este diagrama: la figura 28 representa el concepto de un NineGrid, que muestra las nueve rejillas que están definidas por los bordes Top, Left, Bottom y Right. Las flechas en cada rejilla cuadrada muestran las dimensiones en las cuales aquellos contenidos serán estirados para cumplir con el tamaño del ViewPort.

En adición a las nueve regiones de rejillas ilustradas anteriormente, hay una "décima" rejilla opcional. Esta toma la forma de una imagen adicional que es centrada en el ViewPort y que no esta puesta a escala. Esta puede ser usada para colocar una forma en el centro de un botón, etcétera. Esta "décima rejilla" es llamada un grifo, que esta expuesta por la propiedad GlyphImageSource:

```
public sealed class System.Windows.Media.NineGridBrush : Brush
{
    public NineGridBrush(ImageSource imageSource,
        int LeftBorder,
        int RightBorder,
        int TopBorder,
        int BottomBorder);

    public NineGridBrush(ImageSource imageSource,
        int LeftBorder,
        int RightBorder,
        int TopBorder,
        int BottomBorder,
        ImageSource glyphImage);

    public new NineGridBrush Copy(); // hides Brush.Copy()

    // Default is null
    public ImageSource ImageSource { get; set; }

    // Default is 0
```

606

```

public int LeftBorder { get; set; }

// Default is 0
public int RightBorder { get; set; }

// Default is 0
public int TopBorder { get; set; }

// Default is 0
public int BottomBorder { get; set; }

// Default is null
public ImageSource GlyphImageSource{ get; set; }
}

```

Note que los miembros de borde se cuentan desde el borde de la imagen en píxeles de imagen.

La sintaxis simple para la marca NineGridBrush:

Nine-grid-brush:

"NineGrid" image-uri int int int int. [^glyph-image-uri]

Los cuatro enteros son los valores para LeftBorder, TopBorder y BottomBorder respectivamente. El URI final para el Tenth-grid, o glifo, es opcional.

El Pen es un objeto que toma un Brush y otros parámetros que describen como dibujar un espacio/geometría. Conceptualmente,

734-

607

un Pen describe como crear un área de golpe para una Geometry. Una nueva región es creada la cual esta basada en los bordes de la Geometry, el espesor del Pen's, el PenLineJoin, PenLineCap, y etcétera. Una vez esta región es creada, es llenada con el Brush.

```
public sealed class System.Windows.Media.Pen : Changeable
{
    // Constructors
    Public Pen();
    public Pen(Brush brush, double thickness);
}
```

795

608

```

public new Pen Copy(); // hides Changeable.Copy()

// Properties
// Default is DashArrays.Solid (no dashes)
public DoubleCollection DashArray { get; set;}

// Default is 0
[Animations(DashOffsetAnimations)]
public double DashOffset { get; set;}
public DoubleAnimationCollection DashOffsetAnimations { get; set;}

// Default is Flat
public PenLineCap StartLineCap { get; set;}

// Default is Flat
public PenLineCap EndLineCap { get; set;}

// Default is Flat
public PenDashCap DashCap { get; set;}

// Default is Miter
public PenLineJoin LineJoin { get; set;}

// Default is 10
public double MiterLimit { get; set;}

// Default is null
public Brush Brush { get; set;}

// Default is 1.0
[Animations(ThicknessAnimations)]

```

686

```

public double Thickness { get; set;}
public DoubleAnimationCollection ThicknessAnimations { get; set;}
}

```

El PenLineCap determina como se dibujan los extremos de una línea frotada:

```

public enum System.Windows.Media.PenLineCap
{
    // This is effectively no line cap – the line is squared off
    // at the last point in the line
    Flat,
    // The line is capped by a hemi-circle of diameter equal to
    // the line width
    Round,
    // The dash is capped by a triangle
    Triangle,
    // The line is capped with a square of side with equal to the
    // line width, centered on the end point
    Square
}

```

El PenDashCap determina como los extremos de una línea punteada frotada son dibujados:

```

public enum System.Windows.Media.PenDashCap
{
    // This is effectively no dash cap – the line is squared off
    // at the last point in the line
    Flat,
    // The dash is capped by a hemi-circle of diameter equal to
    // the line width
    Round,
    // The dash is capped by a triangle

```

```

    Triangle
}

```

El PenLineJoin determina como las uniones son dibujadas cuando se frota una línea:

```

public enum System.Windows.Media.PenLineJoin
{
    // A sharp corner is created at the intersection of the outer
    // edges of the intersecting line segments
    Miter,
    // Similar to Miter, but the corner is rounded
    Round,
    // A beveled join, this produces a diagonal corner
    Bevel
}

```

La clase DashArrays comprende propiedades estáticas que suministran acceso a los estilos de línea punteada comunes, bien conocidos:

```
public sealed System.Windows.Media.DashArrays
{
    // A solid Dash array (no dashes)
    public static DoubleCollection Solid { get; }
    // Dash - 3 on, 1 off
    public static DoubleCollection Dash { get; }
    // Dot - 1 on, 1 off
    public static DoubleCollection Dot { get; }
    // DashDot - 3 on, 1 off, 1 on, 1 off
    public static DoubleCollection DashDot { get; }
    // DashDotDot - 3 on, 1 off, 1 on, 1 off, 1 on, 1 off
    public static DoubleCollection DashDotDot { get; }
}
```

COLOR

La arquitectura del color es construida sobre algunos principios generales, incluyendo ese color que requiere un contexto; por lo tanto los valores de color tendrá un contexto explícitamente asignado o un contexto por defecto implícitamente asumido para minimizar las fallas de color en los flujos de trabajo. Adicionalmente, el diseño de la plataforma de núcleo requieren trayectos de código mínimo y API con espacios de vida larga para seguridad, confiabilidad,

mantenimiento, expansión futura y desempeño; por lo tanto convertir el núcleo puede estar limitado principalmente un trayecto de código sRGB al cual las corrientes de contenido que ingresan serán convertidas a y las corrientes que salen serán convertidas de (una herencia sRGB adicional que tendrá una calidad inferior también será permitido). Note que "sRGB" se refiere a representaciones por defecto gráficas de vector interno con base en la norma internacional IEC 61966-2-2; (aunque no hay una definición oficial suministrada para que significa "sc", "composición estándar) será usado aquí para ayudar a clarificar que esto es el espacio optimo para el procesamiento de la composición).

El desempeño requiere que el procesamiento complejo sea llevado a cabo tan cerca de la definición de objeto color/etapa de especificaciones como sea posible, envés de la etapa de conversión en tiempo real; esto requiere que los parámetros de color a API sean convertidos a sRGB (esencialmente de manera inmediata) luego de la especificación y que los valores de color sRGB se mantengan y se sincronicen para objetos definidos non-sRGB. La facilidad del uso requiere un API en capas en el cual en la mayoría de los casos de desarrolladores comunes se exponen primero, mientras que el caso más avanzado tiene los API limpios pero mínimos; por lo tanto los API sRGB son suministrados (pero inmediatamente convertidos a sRGB

6/2

internamente), los API sRGB son suministrados, y un contexto mínimo asociado al API es suministrado para soportar CMYK avanzado (Cyan-Magenta-Yellow-Black), soluciones de monocromo, sistemas visuales humanos y soluciones de multicanal. Ya que el sRGB es esencialmente una gama de color "infinito" caracterización de dispositivo adicional y soluciones de mapeo de gama son requeridos para "enganchar" flujos de trabajo sRGB con dispositivos del mundo real.

El color es una percepción psicológica usada muy frecuentemente de manera típica por una sensación física externa. Esto significa que el color con base en una computadora requiere un contexto físico para poder comunicar efectivamente los colores percibidos a través de dispositivos y entre usuarios. Históricamente diversas tecnologías han sido inconsistentes al suministrar un significado contextual razonable para las implementaciones arquitecturales de color, por ejemplo, esto resultó en "rojo" significando "naranja" para un dispositivo o usuario y "rosado" para otro, con métodos poco viables para resolver el error. La arquitectura presente suministra una combinación de contextos implícitos (usando valores por defecto) y explícitos para cualquier objeto de color. Esto significa que no habrá objetos de color sin significado contextual. Esto es una tecnología imperfecta, y por lo tanto un aspecto de la arquitectura es suministrar contextos de color consistentes en una manera que

puedan ser resueltos progresos de tecnología. Note que la mayoría de usuarios de computadora (y la mayoría de desarrolladores) no quieren tratar con el manejo del color y prefieren que el color se trabaje simplemente de manera correcta.

En general, los intentos de arquitectura para minimizar trayectos de código interno, que es llevado a cabo hasta un grado permitiendo dos trayectos de códigos básicos internamente, uno para la calidad y hardware del futuro, y el otro para la herencia y restricciones de memoria/desempeño. El núcleo MIL interno convierte y el motor de composición soporta 32bpp sRGB y 128bpp scRGB de punto flotante (aunque el 64bpp scRGB también está bajo consideración y, en algunos casos de soportar 64bpp esta mejor implementado con el punto fijo, algunos con flotante y algunos con enteros).

La arquitectura suministra trayectos scRGB de 128bpp a partir de capturar para mostrar la edición para almacenar para imprimir (la pantalla será una memoria de respaldo de 128bpp y 10bpc o mas de memoria frontal), y permite herencia de trayectos sRGB de 32bpp que sacrifica la calidad para el desempeño, la memoria y/o el ancho de banda.

El manejo de color de la presente invención mejora sobre los inconvenientes del estado de la técnica por medio de

suministrar dispositivos y aplicaciones con flexibilidad adicional y suministra una solución de manejo de color con base en perfil. Los escenarios más comunes están basados en objetos de color sRGB y sRGB para soportar obteniendo y estableciendo los valores de color para elemento UI comunes, y soporta la mayoría de creación de contenido para la red, multimedia y gráficos de computador. Los escenarios menos comunes incluyen contextos de color RGB con perfiles de espacio de trabajo específicos para flujos de trabajo de fotografía profesional, usando valores de color CMYK para editar objetos de color para preimpresión y trabajo de diseños gráficos, y trabajos de flujo de color monocromo y multicanal, que soportan escenarios de nichos de impresión y prensado lo mismo que suministran la flexibilidad para soportar flujos de trabajo futuros que no han sido definidos. Los flujos de trabajo HVS (espacios con bases en sistema visual humano) soportan algunos escenarios de edición de nichos de fotografía profesional.

Para poder casar la tecnología de sensor de captura que continua progresando en calidad y profundidad de bit, la imagen soportará por lo menos formatos de 64bpp para cada rasgo/API poder soportar iniciativas de negativo digital contemporáneo. A medida que la presente invención implementa una nueva arquitectura para gráficos de vector, los gráficos de vector serán implementados en punto flotante de 32 bit por

presición de bit de canal. Esta implementación es actualmente "escondida" para poder suministrar acceso al color de 8bpc tradicional lo mismo que a interfaces de escala de grises HSV.

Otro tipo de dato de color se llama dato de color, tal como "CornflowerBlue" o "Pantone". Por medio de suministrar un contexto de color que esta basado en la extensión de perfiles de administración de color tradicional, un color muy genérico y poderoso particularmente es suministrado. Para mantener alguna compatibilidad de herencia con los API anteriores y la práctica común, los constructores por defecto serán obligados hacia entradas sRGB.

El soporte de espacio del color para gráficos de vector está dividido en soporte scRGB nativo, soporte de propiedad para sRGB y espacios similares que no requieren un contexto de color explícito, soporte de método para espacios de color relacionados cercanamente a sRGB o scRGB tales como HSV (matiz, saturación y valor) que también no requiere un contexto de color explícitamente asociado, particularmente el color y espacios de color relacionados tal como paletas y espacios de color indexados que están basados en el contexto de color asociado implícitamente o el explícitamente asociado, y espacios de color que requieren canales de color adicionales lo mismo que contextos de color explícitamente

cur

asociados, tal como CMYK, color de alta fidelidad (CMYK mas naranja y verde), espacios de color de chorro de tinta CcMmYK y soporte de color potencialmente espectral en el futuro.

Mientras estos espacios de color son convertidos a sRGB o sRGB para volverlos en el núcleo MIL o motor de composición, ellos pueden ser almacenados o enviados a impresoras (tales como CMYK) usando el lenguaje de marca gráfico de vector como un lenguaje de diseño de programa. La sintaxis de marca de color influye 4 mecanismos de especificación básicas: hexád decimal, colores conocidos, propiedades sRGB y contexto de color avanzado. Los primeros tres asumen un contexto de color de espacio de color sRGB.

El ejemplo que sigue crea un gradiente usando estos 4 mecanismos:

Example:

<Canvas Width="100px" Height="100px"

Background="VerticalGradient #cc7711 CornFlowerBlue"

617

```

xmlns="using: System.Windows; System.Windows.Controls; System.Windows.Docum
ents; System.Windows.Shapes; System.Windows.Media; System.Windows.Presenters"
>

<Rectangle Top="12.5%" Left="0" Width="100%" Height="12.5%">
  <Rectangle.Fill>
    <RadialGradient>
      <GradientStop Offset="0" Color="sGray.icc 0.5"/>
      <GradientStop Offset="0.5">
        <GradientStop.Color>
          <Color A="0.8" R="0.2" G="1" B="0.2"/>
        </GradientStop.Color>
      </GradientStop>
      <GradientStop Offset="1" Color="mswopintent8.icm 0.9 0.2 0.1 0.3"/>
    </RadialGradient>
  </Rectangle.Fill>
</Rectangle>
</Canvas>

```

El primer color de fondo es especificado como hexád decimal (#ee7711). Esta representación hexád decimal es idéntica a como .NET encuadra V1 y los colores especificados WinForms. Esto es flexible para permitir 4 variaciones diferentes; #RGB, #ARGB, #RRG BBB, o #AARRG GBB.

El segundo color de fondo es especificado como un color conocido (CornFlowerBlue). Esta representación es idéntica a como .NET encuadra V1 y los colores especificados WinForms.

```

// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.Opacity = 0.5;
myButton.Background = myBrush;

// Causes an exception to be thrown.
((SolidColorBrush)myButton.Background).Color = Colors.Blue;

' VB .NET
Dim myBrush As new _
    System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Causes an exception to be thrown.
CType(myButton.Background, System.Windows.Media.SolidColorBrush).Color = _
    System.Windows.Media.Colors.Blue

```

Modificando el objeto Changeable original no se actualizan las copias:

```

// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.Opacity = 0.5;
myButton.Background = myBrush;

```



```

// Does not change the background of the button.
myBrush.Color = Colors.Blue;

' VB .NET
Dim myBrush As new _
    System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Does not change the background of the button.
    myBrush.Color = System.Windows.Media.Colors.Blue

```

Para cambiar el trasfondo del botón en este ejemplo, el programador reasigna la brocha modificada a la propiedad de trasfondo del botón:

```

// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.Opacity = 0.5;
myButton.Background = myBrush;

// Does not change the background of the button.
myBrush.Color = Colors.Blue;

// Updates the background of the button.
myButton.Background = myBrush;

' VB .NET
Dim myBrush As new _
    System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Does not change the background of the button.

```

```
myBrush.Color = System.Windows.Media.Colors.Blue
```

```
* Updates the background of the button.
```

```
myButton.Background = myBrush
```

El programador también puede utilizar el método Copy para recuperar una copia modificable de un objeto Changeable utilizado. La copia recuperada es aún reasignada de nuevo a la propiedad para tener un efecto:

```
// C#
```

```
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
```

```
myBrush.Opacity = 0.5;
```

```
myButton.Background = myBrush;
```

```
SolidColorBrush anotherBrush = (SolidColorBrush)myButton.Background.Copy();
```

```
anotherBrush.Color = Colors.Purple;
```

```
// Updates the background of the button.
```

```
myButton.Background = anotherBrush;
```

```
* VB .NET
```

```
Dim myBrush As new _
```

```
System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
```

```
myBrush.Opacity = 0.5
```

```
myButton.Background = myBrush
```

```
Dim anotherBrush As new System.Windows.Media.SolidColorBrush
```

```
anotherBrush = _
```

```
CType(myButton.Background.Copy, System.Windows.Media.SolidColorBrush)
```

```
anotherBrush.Color = System.Windows.Media.Colors.Purple
```

```
* Updates the background of the button.
```

```
myButton.Background = anotherBrush
```

2

Ya que este no es el comportamiento ideal de un Changeable en todas las situaciones, por ejemplo, considere el programador que quiere modificar la versión utilizada (la copia de trabajo) del Changeable, la clase Changeable le posibilita al programador especificar como se comporta esta cuando se utiliza al suministrar la propiedad StatusOfNextUse.

El StatusOfNextUse suministra tres opciones de cómo se comporta el Changeable cuando se utiliza:

Unchangeable: El comportamiento por defecto, mostrado en los ejemplos de la sección previa. Cuando el objeto Changeable se utiliza, este crea una copia inmutable de sí mismo la cual se utiliza en lugar del objeto original. El programador puede continuar para modificar el objeto original; la versión utilizada (la copia que fue hecha) no se afecta por las modificaciones del objeto original y no se pueden modificar. Para modificar la versión utilizada, el método Copy se utiliza para obtener una versión modificable, esta versión es actualizada, y la nueva versión reemplaza la versión utilizada.

ChangeableCopy: Cuando el objeto Changeable se utiliza, este crea una copia modificable de sí misma que es utilizada en lugar del objeto original. El programador

593

puede continuar para modificar el objeto original; la versión utilizada (la copia que fue hecha) no se afecta por las modificaciones al objeto original, pero es también modificable. La versión utilizada tiene un `StatusOfNextUse` de `Unchangeable`.

ChangeableReference: Cuando el objeto `Changeable` se utiliza, este suministra una referencia de sí mismo. El programador puede continuar modificando el objeto original; los cambios al objeto original afectan la versión utilizada - ellos son el mismo objeto.

El `ChangeableCopy` altera el comportamiento de un `Changeable` de tal forma que cuando se utiliza esta crea una copia modificable de sí mismo, no una copia no modificable (como es el caso con la configuración por defecto de `Unchangeable`). El siguiente código (mostrado previamente) arroja una excepción porque la propiedad `StatusOfNextUse` de `myBrush` tiene una configuración por defecto de `Unchangeable`:

```
// C#  
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
```

```

myBrush.Opacity = 0.5;
myButton.Background = myBrush;

// Causes an exception to be thrown.
((SolidColorBrush)myButton.Background).Color = Colors.Blue;

' VB .NET
Dim myBrush As new _

System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Causes an exception to be thrown.
CType(myButton.Background, System.Windows.Media.SolidColorBrush).Color =
-
System.Windows.Media.Colors.Blue

```

Sin embargo, si la propiedad de `StatusOfNextUse` de la brocha se ajusta al `ChangeableCopy`, el código trabaja como se pretende:

```

// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.StatusOfNextUse = UseStatus.ChangeableCopy;
myBrush.Opacity = 0.5;
myButton.Background = myBrush;

// Works because the brush has a UseStatus of ChangeableCopy.
((SolidColorBrush)myButton.Background).Color = Colors.Blue;

' VB .NET
Dim myBrush As new _

```

```

System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.StatusOfNextUse = MS Avalon.Windows.UseStatus.ChangeableCopy
myBrush.Opacity = 0.5
myButton.Background = myBrush

' Works because the brush has a UseStatus of ChangeableCopy.
CType(myButton.Background, System.Windows.Media.SolidColorBrush).Color = _
    System.Windows.Media.Colors.Blue

```

La configuración de ChangeableCopy también mantiene cualquier sub-objeto del objeto principal modificable. En el siguiente ejemplo, al LinearGradientBrush le es dado un StatusOfNextUse de ChangeableCopy. Como resultado, el LinearGradientBrush y sus sub-objetos permanecen modificables después de que ellos han sido utilizados; el programador no tiene que ajustar la propiedad de StatusOfNextUse de cualquier objeto Changeable contenido por el objeto, tal como el GradientStop en este ejemplo:

546

```

// C#
LinearGradientBrush myBrush = new LinearGradientBrush();
myBrush.StatusOfNextUse = UseStatus.ChangeableCopy;

myBrush.GradientStops.Add(new GradientStop(Colors.Blue, 0));
myBrush.GradientStops.Add(new GradientStop(Colors.Green, 1));

myButton.Background = myBrush;

// Works because the brush has a UseStatus of ChangeableCopy.
((LinearGradientBrush)myButton.Background).GradientStops[0].Color =
Colors.LightBlue;

' VB .NET
Dim myBrush As new System.Windows.Media.LinearGradientBrush
myBrush.StatusOfNextUse = MS Avalon.Windows.UseStatus.ChangeableCopy

```

```

myBrush.GradientStops.Add( _
    new System.Windows.Media.GradientStop(System.Windows.Media.Colors.Blue,0))
myBrush.GradientStops.Add(new _
    System.Windows.Media.GradientStop(System.Windows.Media.Colors.Green, 1))

myButton.Background = myBrush

' Works because the brush has a UseStatus of ChangeableCopy.
CType(myButton.Background, _
    System.Windows.Media.LinearGradientBrush).GradientStops(0).Color = _
    System.Windows.Media.Colors.LightBlue

```

Cuando se utilizan objetos Changeable con un StatusOfNextUse de ChangeableCopy, el programador también puede retener un manejo a la versión utilizada del Changeable y utilizar esa referencia para modificar el objeto. En el siguiente ejemplo, una referencia al LinearGradientBrush utilizado se recupera y se utiliza para modificar el trasfondo de un botón:

```
// C#  
LinearGradientBrush myBrush = new LinearGradientBrush();  
myBrush.StatusOfNextUse = UseStatus.ChangeableCopy;  
  
myBrush.GradientStops.Add(new GradientStop(Colors.Blue, 0));  
myBrush.GradientStops.Add(new GradientStop(Colors.Green, 1));  
  
myButton.Background = myBrush;  
  
LinearGradientBrush usedBrush = (LinearGradientBrush)myButton.Background;  
  
// Works because the brush has a UseStatus of ChangeableCopy.  
usedBrush.GradientStops[0].Color = Colors.LightBlue;  
  
' VB .NET  
Dim myBrush As new System.Windows.Media.LinearGradientBrush
```



```

myBrush.StatusOfNextUse = MS Avalon.Windows.UseStatus.ChangeableCopy
myBrush.GradientStops.Add( _
    new
    System.Windows.Media.GradientStop(System.Windows.Media.Colors.Blue,0))
myBrush.GradientStops.Add( _
    new
    System.Windows.Media.GradientStop(System.Windows.Media.Colors.Green,1))

myButton.Background = myBrush

Dim usedBrush As new System.Windows.Media.LinearGradientBrush
usedBrush = CType(myButton.Background, LinearGradientBrush)

' Works because the brush has a UseStatus of ChangeableCopy.
usedBrush.GradientStops(0).Color = System.Windows.Media.Colors.LightBlue

```

La configuración del `ChangeableReference` altera el comportamiento de un `Changeable` de tal forma que este suministra una referencia de sí mismo cuando se utiliza. El programador puede continuar modificando el objeto original; los cambios al objeto original afectan la versión utilizada porque ellos son el mismo objeto. El siguiente es un ejemplo:

549

```
// C#
SolidColorBrush changeableReferenceBrush = new SolidColorBrush();
changeableReferenceBrush.Color = Colors.LimeGreen;

button1.Background = changeableReferenceBrush;
button2.Background = changeableReferenceBrush;
button3.Background = changeableReferenceBrush;

// Changes the color of the three buttons.
changeableReferenceBrush.Color = Colors.Purple;
```

```
// Also changes the color of all three buttons.
((SolidColorBrush)button1.Background).Color = Colors.Blue;

' VB .NET
Dim changeableReferenceBrush As new System.Windows.Media.SolidColorBrush
changeableReferenceBrush.Color = System.Windows.Media.Colors.LimeGreen

button1.Background = changeableReferenceBrush
button2.Background = changeableReferenceBrush
button3.Background = changeableReferenceBrush

' Changes the color of all three buttons.
changeableReferenceBrush.Color = System.Windows.Media.Colors.Purple

' Also changes the color of all three buttons.
CType(button1.Background, System.Windows.Media.SolidColorBrush).Color = _
    System.Windows.Media.Colors.Blue
```

BRUSH Y PEN

La brocha es un objeto que representa un método para llenar un plano. Además de ser capaz de llenar un plano de manera absoluta, las brochas de la capa de integración de medios también son capaces de adaptar como ellas llenan el plano con relación al tamaño del objeto que ellas están llenando. Los ejemplos de tipos de brochas incluyen SolidColorBrush, VisualBrush (que puede referenciar un Visual), DrawingBrush (que puede referenciar un recurso de gráficos de vector), LinearGradient, RadialGradient, ImageBrush y NineGridBrush. Los valores por defecto para las propiedades de brocha son especificados adelante, y son generalmente valores que no resultan en acción. Esto es, los colores son por defecto transparentes y así sucesivamente. También, las colecciones de animación por defecto son nulas.

Como se mencionó anteriormente, ciertos objetos brocha tendrán una idea de cómo ellos se relacionan con el sistema de coordenadas cuando ellos son utilizados, y una idea de cómo ellos se relacionan con la caja de unión de la geometría con la cual ellos son utilizados. Este tamaño está basado desde el objeto que la brocha está llenando.

La clase de base Brush tiene un Transform, una opacidad general, y un modo de mezcla:

```

public abstract class System.Windows.Media.Brush : Changeable
{
    internal Brush();
    public new Brush Copy(); // hides Changeable.Copy()

    // Default is Transform.Identity
    public Transform Transform { get; set; }

    [Animation("OpacityAnimations")]
    public double Opacity { get; set; } // Default is 1.0
    public DoubleAnimationCollection OpacityAnimations { get;set; }

    /// The BlendMode to apply to this Brush and its destination when drawn.
    /// Default is BlendModes.Normal
    public BlendMode BlendMode {get; set; }
}

```

Los objetos Brush (y otros recursos de objeto en el VectorGraphics y el MIL API) son Changeables y son escribibles después de que ellos han sido creados, y siguen el patrón Changeable general por como ellos se comportan después de que ellos son utilizados en uso calificado.

Las brochas (excepto para el VisualBrush y el DrawingBrush) tienen una sintaxis simple para uso en marcas, sin embargo, esta sintaxis simple no permite acceso a todas las propiedades y comportamientos. Si un programador necesita más acceso que la sintaxis simple suministra, el programador

necesitará utilizar sintaxis compleja. Note que solo la sintaxis compleja es documentada aquí para evitar redundancia, en razón a que la sintaxis compleja sigue el mismo patrón que otras clases CLR.

Los siguientes son tipos de brocha que son con sintaxis simple para marcas en una implementación habitual:

```
brush:  
  solid-color-brush |  
  linear-gradient-brush |  
  radial-gradient-brush |  
  image-brush |  
  video-brush |  
  nine-grid-brush
```

Muchos de los tipos de brocha utilizan un sistema de coordenadas para especificar algunos de sus parámetros. Este sistema de coordenadas puede ser definido como relación a la caja de unión simple de la geometría con la cual la brocha se utiliza, o puede ser absoluta, e interpretada en el espacio de coordenada que es activa en el momento en que la brocha es utilizada. Estos son conocidos, respectivamente, como el modo `RelativeToBoundingBox` y el modo `Absolute`.

```
public enum System.Windows.Media.BrushMappingMode
{
    Absolute,
    RelativeToBoundingBox,
}
```

El SolidColorBrush llena el plano con un color sólido. Si existe un componente alfa del color, este se combina de una manera multiplicativa con el atributo de opacidad correspondiente en el Brush.

```
public sealed class System.Windows.Media.SolidColorBrush : Brush
{
    // Constructors
    public SolidColorBrush(); // initialize to transparent
    public SolidColorBrush(Color color);
    public new SolidColorBrush Copy(); // hides Changeable.Copy()

    // Default is transparent
    [Animation("ColorAnimations")]
    public Color Color { get; set; }
    public ColorAnimationCollection ColorAnimations { get; set; }
}
```

En razón a que este es un tipo simple, (que es, ninguna de sus propiedades son Changeables), el único método protegido que necesita ser implementado es CloneCore(). También, en razón a que existe una combinación de valores que hagan este

test

objeto inválido, no existe la necesidad de suministrar un método `ValidateObjectState()`. Estos métodos y otros métodos relacionados son descritos en el Apéndice anexo.

La sintaxis simple para las marcas para `SolidColorBrush`:

```
solid-color-paint:  
color
```

La clase `Brushes` contiene propiedades estáticas para las instancias `SolidColorBrush` que son expuestas. Cada una es un conjunto de valor de color del mismo nombre. Note que en razón a que estas son brochas estandarizadas, ellas tienen el `IsChangeable` ajustado a falso, (por ejemplo, la implementación llama el `MakeUnchangeable()` luego de construcción).

Lo siguiente establece algunos colores estándar:

```
public sealed class Brushes  
{  
    public static SolidColorBrush AliceBlue { get; }  
    public static SolidColorBrush AntiqueWhite { get; }
```

```
public static SolidColorBrush Aqua { get; }
public static SolidColorBrush Aquamarine { get; }
public static SolidColorBrush Azure { get; }
public static SolidColorBrush Beige { get; }
public static SolidColorBrush Bisque { get; }
public static SolidColorBrush Black { get; }
public static SolidColorBrush BlanchedAlmond { get; }
public static SolidColorBrush Blue { get; }
public static SolidColorBrush BlueViolet { get; }
public static SolidColorBrush Brown { get; }
public static SolidColorBrush BurlyWood { get; }
public static SolidColorBrush CadetBlue { get; }
public static SolidColorBrush Chartreuse { get; }
public static SolidColorBrush Chocolate { get; }
public static SolidColorBrush Coral { get; }
public static SolidColorBrush CornflowerBlue { get; }
public static SolidColorBrush Cornsilk { get; }
public static SolidColorBrush Crimson { get; }
public static SolidColorBrush Cyan { get; }
public static SolidColorBrush DarkBlue { get; }
public static SolidColorBrush DarkCyan { get; }
public static SolidColorBrush DarkGoldenrod { get; }
public static SolidColorBrush DarkGray { get; }
public static SolidColorBrush DarkGreen { get; }
public static SolidColorBrush DarkKhaki { get; }
public static SolidColorBrush DarkMagenta { get; }
public static SolidColorBrush DarkOliveGreen { get; }
public static SolidColorBrush DarkOrange { get; }
public static SolidColorBrush DarkOrchid { get; }
public static SolidColorBrush DarkRed { get; }
public static SolidColorBrush DarkSalmon { get; }
public static SolidColorBrush DarkSeaGreen { get; }
```



```
public static SolidColorBrush DarkSlateBlue { get; }  
public static SolidColorBrush DarkSlateGray { get; }  
public static SolidColorBrush DarkTurquoise { get; }  
public static SolidColorBrush DarkViolet { get; }  
public static SolidColorBrush DeepPink { get; }  
public static SolidColorBrush DeepSkyBlue { get; }  
public static SolidColorBrush DimGray { get; }  
public static SolidColorBrush DodgerBlue { get; }  
public static SolidColorBrush Firebrick { get; }  
public static SolidColorBrush FloralWhite { get; }  
public static SolidColorBrush ForestGreen { get; }  
public static SolidColorBrush Fuchsia { get; }  
public static SolidColorBrush Gainsboro { get; }  
public static SolidColorBrush GhostWhite { get; }  
public static SolidColorBrush Gold { get; }  
public static SolidColorBrush Goldenrod { get; }  
public static SolidColorBrush Gray { get; }  
public static SolidColorBrush Green { get; }  
public static SolidColorBrush GreenYellow { get; }  
public static SolidColorBrush Honeydew { get; }  
public static SolidColorBrush HotPink { get; }  
public static SolidColorBrush IndianRed { get; }  
public static SolidColorBrush Indigo { get; }  
public static SolidColorBrush Ivory { get; }  
public static SolidColorBrush Khaki { get; }  
public static SolidColorBrush Lavender { get; }  
public static SolidColorBrush LavenderBlush { get; }  
public static SolidColorBrush LawnGreen { get; }  
public static SolidColorBrush LemonChiffon { get; }  
public static SolidColorBrush LightBlue { get; }  
public static SolidColorBrush LightCoral { get; }  
public static SolidColorBrush LightCyan { get; }
```

```
public static SolidColorBrush LightGoldenrodYellow { get; }
public static SolidColorBrush LightGray { get; }
public static SolidColorBrush LightGreen { get; }
public static SolidColorBrush LightPink { get; }
public static SolidColorBrush LightSalmon { get; }
public static SolidColorBrush LightSeaGreen { get; }
public static SolidColorBrush LightSkyBlue { get; }
public static SolidColorBrush LightSlateGray { get; }
public static SolidColorBrush LightSteelBlue { get; }
public static SolidColorBrush LightYellow { get; }
public static SolidColorBrush Lime { get; }
public static SolidColorBrush LimeGreen { get; }
public static SolidColorBrush Linen { get; }
public static SolidColorBrush Magenta { get; }
public static SolidColorBrush Maroon { get; }
public static SolidColorBrush MediumAquaMarine { get; }
public static SolidColorBrush MediumBlue { get; }
public static SolidColorBrush MediumOrchid { get; }
public static SolidColorBrush MediumPurple { get; }
public static SolidColorBrush MediumSeaGreen { get; }
public static SolidColorBrush MediumSlateBlue { get; }
public static SolidColorBrush MediumSpringGreen { get; }
public static SolidColorBrush MediumTurquoise { get; }
public static SolidColorBrush MediumVioletRed { get; }
public static SolidColorBrush MidnightBlue { get; }
public static SolidColorBrush MintCream { get; }
public static SolidColorBrush MistyRose { get; }
public static SolidColorBrush Moccasin { get; }
public static SolidColorBrush NavajoWhite { get; }
public static SolidColorBrush Navy { get; }
public static SolidColorBrush OldLace { get; }
public static SolidColorBrush Olive { get; }
```

632

```
public static SolidColorBrush OliveDrab { get; }
public static SolidColorBrush Orange { get; }
public static SolidColorBrush OrangeRed { get; }
public static SolidColorBrush Orchid { get; }
public static SolidColorBrush PaleGoldenrod { get; }
public static SolidColorBrush PaleGreen { get; }
public static SolidColorBrush PaleTurquoise { get; }
public static SolidColorBrush PaleVioletRed { get; }
public static SolidColorBrush PapayaWhip { get; }
public static SolidColorBrush PeachPuff { get; }
public static SolidColorBrush Peru { get; }
public static SolidColorBrush Pink { get; }
public static SolidColorBrush Plum { get; }
public static SolidColorBrush PowderBlue { get; }
public static SolidColorBrush Purple { get; }
public static SolidColorBrush Red { get; }
public static SolidColorBrush RosyBrown { get; }
public static SolidColorBrush RoyalBlue { get; }
public static SolidColorBrush SaddleBrown { get; }
public static SolidColorBrush Salmon { get; }
public static SolidColorBrush SandyBrown { get; }
public static SolidColorBrush SeaGreen { get; }
public static SolidColorBrush SeaShell { get; }
public static SolidColorBrush Sienna { get; }
public static SolidColorBrush Silver { get; }
public static SolidColorBrush SkyBlue { get; }
public static SolidColorBrush SlateBlue { get; }
public static SolidColorBrush SlateGray { get; }
public static SolidColorBrush Snow { get; }
public static SolidColorBrush SpringGreen { get; }
public static SolidColorBrush SteelBlue { get; }
public static SolidColorBrush Tan { get; }
```

632

684

```

public static SolidColorBrush Teal { get; }
public static SolidColorBrush Thistle { get; }
public static SolidColorBrush Tomato { get; }
public static SolidColorBrush Transparent { get; }
public static SolidColorBrush Turquoise { get; }
public static SolidColorBrush Violet { get; }
public static SolidColorBrush Wheat { get; }
public static SolidColorBrush White { get; }
public static SolidColorBrush WhiteSmoke { get; }
public static SolidColorBrush Yellow { get; }
public static SolidColorBrush YellowGreen { get; }
}

```

Los gradientes son dibujados especificando un conjunto de paradas de gradiente. Estas paradas de gradientes especifican los colores a lo largo por medio de alguna clase de progresión. Hay dos tipos de gradientes soportados actualmente, particularmente gradientes lineales y radiales. El gradiente es dibujado por medio de hacer interpolaciones entre las paradas de gradientes en el espacio de color especificado. Los gradientes están compuestos de una lista de paradas de gradientes. Cada una de estas paradas de gradientes contiene un color (con el valor alfa incluido) y un desplazamiento. Sino hay paradas de gradientes especificados, la brocha es dibujada como transparente (como si no hubiera una brocha especificada). Si hay solamente una

1270

OK

parada de gradiente especificada, la brocha es dibujada en un color sólido con el color especificado. Cualesquier paradas de gradientes con desplazamientos en el rango de 0 a 1 ($0,0\dots1,0$) son considerados, junto con la parada mas grande en el rango $(-\alpha\dots0,0]$ y el mas pequeña en el rango $(1,0\dots+\alpha)$. Si el conjunto de paradas que se esta considerando incluye una parada que esta por fuera del rango cero a uno, una parada implícita es derivada en cero (y/o uno) lo que representa el color interpolado que ocurriría en esta parada. También, si dos o mas paradas se establecen para en el mismo desplazamiento, una transición dura (envés de interpoladas) ocurre en ese desplazamiento. El orden en el cual las paradas son agregadas determina el comportamiento de este desplazamiento; la primer parada para ser agregada es el color efectivo antes de ese desplazamiento, la ultima parada que es establecida es el color efectivo después de esta parada y cualesquier paradas adicionales en este desplazamiento son ignoradas.

Esta clase es modificable como cualquier otra clase de recursos:

EAT

811

```

public sealed class System.Windows.Media.GradientStop : Changeable
{
    public GradientStop();    public GradientStop(Color color, double offset);
    public GradientStop(Color color, ColorAnimationCollection colorAnimations,
        double offset, DoubleAnimationCollection offsetAnimations);
    public new GradientStop Copy(); // hides Changeable.Copy()

    // Default is transparent
    [Animation("ColorAnimations")]
    public Color Color { get; set; }
    public ColorAnimationCollection ColorAnimations { get; set; }

    // Default is 0
    [Animation("OffsetAnimations")]
    public double Offset { get; set; }
    public DoubleAnimationCollection OffsetAnimations { get; set; }
}

```

Tal como la SolidColorBrush, esta tiene modificables anidados en las colecciones de animación.

El GradientSpreadMethod enum especifica como el gradiente debería ser dibujado por fuera del rector o espacio especificado. Hay tres valores posibles, incluyendo Pad, en el cual los colores finales (primero y ultimo) son usados para llenar el espacio restante, Reflect, en el cual las paradas son vueltas a ocurrir en orden inverso repetidamente para llenar el espacio, y Repeat, en la cual las paradas son

1672

642

repetidas en orden hasta que el espacio es llenado. El valor por omisión para las propiedades de este tipo es Pad:

```
public enum System.Windows.Media.GradientSpreadMethod
{
    Pad,
    Reflect,
    Repeat
}
}
```

La figura 24 suministra algunos ejemplos de GradientSpreadMethod, (aunque en escala de grises envés de en color). Cada forma tiene un gradiente lineal que va desde el blanco al gris. La línea sólida representa el vector de gradiente.

El ColorInterpolationMode enum define el modo de interpolación para colores dentro de un gradiente. Las dos opciones son PhysicallyLinearGamma y PerceptuallyLinearGamma.

613

```
public enum ColorInterpolationMode
{
    // Colors are interpolated in Gamma 1.0 space
    PhysicallyLinearGamma,
    // Colors are interpolated in Gamma 2.2 space
    PerceptuallyLinearGamma
}
```

Esta es una clase base abstracta.

```
public abstract class System.Windows.Media.GradientBrush : Brush
{
    internal GradientBrush();
    public new GradientBrush Copy(); // hides Changeable.Copy()

    // Default is "PerceptuallyLinearGamma"
    public ColorInterpolationMode ColorInterpolationMode { get; set; }

    // Default is RelativeToBoundingBox
    public BrushMappingMode MappingMode { get; set; }

    // Default is Pad
}
```



```

public GradientSpreadMethod SpreadMethod { get; set; }

// Gradient Stops
public void AddStop(Color color, double offset);
public GradientStopCollection GradientStops { get; set; }

// ColorInterpolationMode
public ColorInterpolationMode ColorInterpolationMode { get; set; }
}

```

El LinearGradient especifica una brocha de gradiente lineal junto a un vector. La parada individual especifica las paradas de colores junto con ese vector.

```

public sealed class System.Windows.Media.LinearGradient : GradientBrush
{
    public LinearGradient(); // initializes to transparent

    // Sets up a gradient with two colors and a gradient vector
    // specified to fill the object the gradient is applied to.
    // This implies RelativeToBoundingBox for the GradientUnits
    // property
    public LinearGradient(Color color1, Color color2, double angle);
    public LinearGradient(Color color1, Color color2,
                          Point vectorStart, Point vectorEnd);
    public new LinearGradient Copy(); // hides Changeable.Copy()

    // Gradient Vector Start Point
    // Default is 0,0
    [Animation("StartPointAnimations")]
    public Point StartPoint { get; set; }
    public PointAnimationCollection StartPointAnimations { get; set; }
}

```

```

// Default is 1,1
[Animation("EndPointAnimations")]
public Point EndPoint { get; set; }
public PointAnimationCollection EndPointAnimations { get; set; }
}

```

La sintaxis simple para la marcación para la LinearGradientBrush:

Linear-gradient-brush:

"HorizontalGradient" comma-wsp color comma-wsp color

"VerticalGradient" comma-wsp color comma-wsp color

"LinearGradient" comma-wsp coordinate-pair comma-wsp color
comma-wsp color

La marcación para el LinearGradient permite la especificación de un LinearGradient con dos paradas de color, en los desplazamientos 0 y 1. Si la versión "LinearGradient", es usada, el punto de inicio y el punto final son especificados, respectivamente. Si se usa "HorizontalGradient" el punto de inicio es establecido en 0,0 y el punto final es establecido en 1,0. Si se usa "VerticalGradient", el punto de inicio es establecido en 0,0 y el punto final es establecido en 0,1. En estos casos, el Mappingmode por omisión es usado, el cual es RelativeToBoundingBox.

El RadialGradient es similar en el modelo de programación al gradiente lineal. Sin embargo, aunque el gradiente lineal tiene un punto de partida y uno de final para definir el vector del gradiente, el gradiente radial tiene un círculo junto con un punto focal para definir el comportamiento del gradiente. El círculo define el punto final del gradiente, en otras palabras, una parada de gradiente en 1,0 define el color en la circunferencia del círculo. El punto focal define el centro del gradiente. Una parada de gradiente en 0,0 define el color en el punto focal. La figura 25 representa un RadialGradient que (en escalas de grises) va del blanco al gris. El círculo exterior representa el círculo de gradiente mientras que el punto sólido denota el punto focal. Este gradiente tiene SpreadMethod establecido para el Pad.

```
public sealed class System.Windows.Media.RadialGradient : GradientBrush
{
    public RadialGradient(); // initialize to transparent

    // Sets up a gradient with two colors.
```

047

```

// This implies RelativeToBoundingBox for the GradientUnits
// property along with a center at (0.5,0.5)
// a radius of 0.5 and a focal point at (0.5,0.5)
public RadialGradient(Color color1, Color color2);

public new RadialGradient Copy(); // hides Changeable.Copy()

// Default is 0.5,0.5
[Animation("CenterAnimations")]
public Point Center { get; set; }
public PointAnimationCollection CenterAnimations { get; set; }

// Default is 0.5
[Animation("RadiusXAnimations")]
public double RadiusX { get; set; }
public DoubleAnimationCollection RadiusXAnimations { get; set; }

// Default is 0.5
[Animation("RadiusYAnimations")]
public double RadiusY { get; set; }
public DoubleAnimationCollection RadiusYAnimations { get; set; }

// Default is 0.5,0.5
[Animation("FocusAnimations")]
public Point Focus { get; set; }
public PointAnimationCollection FocusAnimations { get; set; }
}

```

La marca para el RadialGradient permite la especificación de un RadialGradient con dos paradas de color en los desplazamientos 0 y 1 respectivamente. El MappingMode por

618

omisión es usado, lo cual es `RelativeToBoundingBox`, ya que son radios por omisión, 0,5:

radial-gradient-brush:

"RadialGradient" comma-wsp color comma-wsp color

El `TileBrush` es una clase de base abstracta que contiene la lógica para describir una baldosa y significa por medio de que esa baldosa debe llenar un área. Las subclases de `TileBrush` contienen contenido, y lógicamente definen una manera de llenar un plano infinito.

El `Stretch` enum es usado para describir como un `ViewBox` (espacio de fuente coordinado) es mapeado a un `ViewPort` (espacio de destino coordinado). Esto es usado en `TileBrush`:

```

public enum System.Windows.Stretch
{
    // Preserve original size
    None,
    // Aspect ratio is not preserved, ViewBox fills ViewPort
    Fill,
    // Aspect ratio is preserved, ViewBox is uniformly scaled as large as
    // possible such that both width and height fit within ViewPort
    Uniform,
    // Aspect ratio is preserved, ViewBox is uniformly scaled as small as
    // possible such that the entire ViewPort is filled by the ViewBox
    UniformToFill
}

```

La figura 26 suministra ejemplos de estiramientos. En estos ejemplos los contenidos son alineados desde la parte superior /a la parte izquierda.

El TileMode enum es usado para describir si y como un espacio es llenado por Tiles. Un TileBrush define en donde la base Tile esta (especificado por el ViewPort). El resto del espacio es llenado y con base en el valor TileMode.

```

public enum System.Windows.Media.TileMode
{
    // Do not tile - only the base tile is drawn, the remaining area is

```

```

// left as transparent
None,
// The basic tile mode – the base tile is drawn and the remaining area
// is filled by repeating the base tile such that the right edge of one
// tile butts the left edge of the next, and similarly for bottom and top
Tile,
// The same as tile, but alternate columns of tiles are flipped
// horizontally. The base tile is drawn untransformed.
FlipX,
// The same as tile, but alternate rows of tiles are flipped vertically
// The base tile is drawn untransformed.
FlipY,
// The combination of FlipX and FlipY. The base tile is drawn
// untransformed
FlipXY
}

```

La figura 27 suministra ejemplos de TileMode. La baldosa mas superior izquierda en cada ejemplo es la baldosa base. Estos ejemplos representan None, Tile, FlipX, FlipY, y FlipXY.

El VerticalAlignment enum es usado para describir como el contenido esta posicionado dentro de un contenedor verticalmente:

```

public enum System.Windows.VerticalAlignment
{
    // Align contents towards the top of a space
    Top,
    // Center contents vertically
    Center,
    // Align contents towards the bottom of a space
    Bottom,
}

```

El HorizontalAlignment enum es usado para describir como el contenido esta posicionado dentro de un contenedor horizontalmente.

```

public enum System.Windows.HorizontalAlignment
{
    // Align contents towards the left of a space
    Left,
    // Center contents horizontally
    Center,
    // Align contents towards the right of a space
    Right,
}

```

Las propiedades de TileBrush seleccionan una porción rectangular de un plano infinito que para que sea una baldosa (el ViewBox) y describe un rectángulo de destino (ViewPort) el cual será el Tile base en el área que va a ser llenada. El

área de destino restante será llenado con base en la propiedad de `TileMode`, que controla si y como la baldosa original es replicada para llenar el espacio restante:

```
public abstract class System.Windows.Media.TileBrush : Brush
{
    public new TileBrush Copy(); // hides Brush.Copy()

    // Default is RelativeToBoundingBox
    public BrushMappingMode ViewPortUnits { get; set; }

    // Default is RelativeToBoundingBox
    public BrushMappingMode ContentUnits { get; set; }

    // Default is Rect.Empty
    [Animation("ViewBoxAnimations")]
    public Rect ViewBox { get; set; }
    public RectAnimationCollection ViewBoxAnimations { get; set; }
}
```

```
// Default is Fill
public Stretch Stretch { get; set; }

// Default is None
public TileMode TileMode { get; set; }

// Default is Center
public HorizontalAlignment HorizontalAlignment { get; set; }

// Default is Center
public VerticalAlignment VerticalAlignment { get; set; }

// Default is 0,0,1,1
[Animation("ViewPortAnimations")]
public Rect ViewPort { get; set; }
public RectAnimationCollection ViewPortAnimations { get; set; }
}
```

63

Los contenidos de un `TileBrush`'s no tienen límites intrínsecos, y describen de manera efectiva un plano infinito. Estos contenidos existen en su propio espacio coordinado, y el espacio que esta siendo llenado por el `TileBrush` es el espacio local coordinado en el momento de la aplicación. El espacio de contenido es mapeado en el espacio local con base en el `ViewBox`, `Viewport`, `Alignments` y las propiedades `Stretch`. El `ViewBox` es especificado en el espacio de contenido y este rectángulo es mapeado dentro del rectángulo `Viewport`.

El `Viewport` define la localización en donde los contenidos serán eventualmente dibujados, creando la baldosa de base para este `Brush`. El valor del `ViewportUnits` es absoluto, el valor del `Viewport` es considerado que esta en el espacio local en el momento de la aplicación. Si, en su lugar, el valor de los `ViewportUnits` es `RelativeToBoundingBox`, entonces el valor del `Viewport` es considerado como en el que esta en el espacio coordinado en donde 0,0 es la esquina superior/izquierda de la caja de límites del objeto que esta siendo pintado y 1,1 es la esquina inferior/derecha de la misma caja. Por ejemplo, considere un `RectangleGeometry` que esta siendo llenado el cual es dibujado desde 100,100 a 200,200. Entonces, si el `ViewportUnits` es absoluto, un `Viewport` de (100,100,100,100) describiría el área de

624

654

contenido completa. Si el ViewPortUnits es RelativeToBoundingBox, un ViewPort de (0,0,1,1) describiría el área de contenido completa. Si el tamaño del ViewPort's esta vacío y el Stretch no es None, este Brush se convierte en nada.

El ViewBox esta especificado en el espacio de contenido. Este rectángulo es transformado para ajustarse dentro del ViewPort según se determina por las propiedades Alignment y la propiedad Stretch. Si el Stretch es None, entonces no habrá una puesta en escala aplicada a estos contenidos. Si el Stretch es Fill, entonces el ViewBox es puesto en escala independientemente tanto en X como Y para que sea del mismo tamaño que el ViewPort. Si el Stretch es Uniform o UniformToFill, la lógica es similar pero las dimensiones X y Y son puestas a escala uniformemente, preservando la proporción de aspecto de los contenidos. Si el Stretch es Uniform, el ViewBox es puesto en escala para tener la dimensión más constreñida igual al tamaño del ViewPort. Si el Stretch es UniformToFill, el ViewBox es puesto en escala para tener la dimensión menos constreñida igual a la dimensión del ViewPort. Otra manera de pensar de esto es que tanto el Uniform como el UniformToFill preservan la proporción de aspecto, pero el Uniform asegura que el ViewBox completo esta dentro del ViewPort (potencialmente dejando porciones del ViewPort sin cubrir por el ViewBox), y el UniformToFill

625

655

el objeto lo mismo pero crea un nuevo objeto en el estado objetivo. En este diagrama de estado, existen dos estados, y las transiciones ocurren cuando cualquier `Copy()` o `MakeUnchangeable()` son llamados, y cuando un objeto se utiliza de una manera que califica como un uso, como se describió anteriormente. Note que llamar el método `Copy()` de cualquier estado da como resultado un nuevo valor con su propiedad `IsChangeable` establecida como cierta, mientras que llamar el `MakeUnchangeable()` resulta en el valor objetivo siendo `IsChangeable` ajustado como falso.

La descripción anterior presenta un modelo directo, auto-consistente, que describe el uso básico con solo dos estados, los métodos `Copy()` y `MakeUnchangeable()`, y la noción de "utilizar" un valor `Changeable`. Sin embargo, con respecto a los cambios, los ejemplos de modificación anteriores están basados en el concepto de reemplazo, esto es, copiar un ítem existente, cambiarlo en lugar, y copiarlo de nuevo. Esto implica ubicar montón (que puede ser potencialmente significativo, dependiendo de que tan profundo vaya a ser el cambio, y que tan amplio es el objeto mismo para los clones superficiales), así como también una carga adicional sobre el programador para mantener algún mecanismo para encontrar la senda al atributo para modificar.

7566

66

De acuerdo con un aspecto de la presente invención, agregar soporte para el concepto de mutabilidad cierta de valores, otra propiedad se agrega al modelo, denominado `StatusOfNextUse` del tipo `UseStatus`. Note que el tema fundamental que previene la mutabilidad en el modelo de propiedad simple es que un uso calificado de un valor da como resultado incondicionalmente en el valor resultante con la propiedad `IsChangeable` igual a falso. La propiedad `StatusOfNextUse` resuelve este tema.

```
public enum System.Windows.UseStatus
{
    Unchangeable,
    ChangeableCopy
}
```

Por defecto, la propiedad `StatusOfNextUse` es `UseStatus.Unchangeable`, pero puede ser ajustada a `UseStatus.ChangeableCopy`, por medio de la cual un uso del valor sobre el cual este se ajustará como resultado un objeto clon siendo hecho de tal forma que tiene la propiedad `IsChangeable` ajustada a cierto. Como resultado, el valor del objeto se puede modificar en lugar sin el sitio de apilado adicional.

509

Adicionalmente, en razón a que los valores en uso pueden ser mutados en este método, se suministra una notificación cuando tales cambios ocurren, por vía de un evento simple cambiado. Adicionalmente, en razón a que el objeto no es ya inmutable, el contexto-afinidad es suministrado a través del miembro `UIContext`. Note que cuando el objeto es mutable, este tiene un valor nulo. De otra manera este pertenece al `UIContext` en el cual fue creado. La definición de la clase `Changeable` resultante se vuelve:

```
public class System.Windows.Changeable
{
    // Carryover from "basic usage" above
    public bool IsChangeable { get; } // defaults to true
    public Changeable Copy();
    public void MakeUnchangeable();

    // New members
    public bool CanMakeUnchangeable { get; } // will
        // MakeUnchangeable succeed?
    public UseStatus StatusOfNextUse { get; set; } // defaults
        // to Unchangeable
    public event EventHandler Changed { add; remove; }
    public UIContext UIContext { get; } // null when immutable
}
```

El ejemplo anterior es simple, la mutabilidad superficial describe los requisitos para cambiar la opacidad de un `Brush`,

650

con el código necesitando correr cada tiempo que la opacidad tenía que ser cambiada. Como contraste, con el mecanismo de mutabilidad basado en la propiedad `StatusOfNextUse`, primero el `Btn.Background` mismo tiene un valor de propiedad `IsChangeable` de cierto:

```
Brush b = Btn.Background.Copy(); // get IsChangeable copy
b.StatusOfNextUse = UseStatus.ChangeableCopy;
Btn.Background = b;
```

Lo anterior ha utilizado (en un uso calificado) un valor con `StatusOfNextUse` de `UseStatus.ChangeableCopy`, de tal forma que el resultado mismo es cambiable. Una vez configurado, el programador puede hacer modificaciones como se desee, como en el siguiente ejemplo:

```
Btn.Background.Opacity = 0.3;
```

El programador puede hacer tales modificaciones tan a menudo como desee, y las modificaciones ocurrirán directamente, sin ninguna localización de objetos sobre las configuraciones subsecuentes.

651

659

Note que el ejemplo anterior no describe como el `Btn.Background` viene a estar en el primer lugar, y así una copia de este necesita ser hecha por vía del método `Copy()`. En una situación de mutabilidad deliberada donde el programador quiere crear un trasfondo a ser modificado, una mejor manera de hacer esto puede ser directamente, como en el siguiente ejemplo:

```
SolidColorBrush b = new SolidColorBrush();  
b.Color = Colors.Red;  
b.StatusOfNextUse = UseStatus.ChangeableCopy; // set this up  
// directly  
Btn.Background = b;
```

En este punto, el programador puede especificar la opacidad (`Btn.Background.Opacity=...`) siempre y cuando lo desee, en la medida en que la brocha era creada inicialmente como `StatusOfNextUse` igual a `UseStatus.ChangeableCopy`.

Se debe notar que utilizar el modelo basado en reemplazo en lugar del modelo basado en mutante no es particularmente difícil dados los ejemplos anteriores. Esto es en razón a que los cambios son hechos en el primer nivel, y pueden no aparecer prohibitivamente costosos para reemplazarlos siempre en lugar de mutarlos. De hecho, esta es la técnica válida cuando solamente se desea mutabilidad limitada. Sin embargo,

552

cuando los cambios son hechos a valores más profundos un objeto, el modelo mutante es claramente superior.

Como un ejemplo de tal mutabilidad más profunda, considere el `LinearGradientBrush` (`lgb`), en donde el programador quiere repetidamente cambiar el color de la séptima parada (`lgb.Stops[6]`). El programador puede utilizar las mismas instrucciones que anteriormente para instalar una versión mutable en `Btn.Background`:

```
Brush b = Btn.Background.Copy(); // get IsChangeable copy
b.StatusOfNextUse = UseStatus.ChangeableCopy;
Btn.Background = b;
```

Posteriormente, el programador puede elaborar repetidamente los cambios deseados:

```
LinearGradientBrush lgb = ((LinearGradientBrush)Btn.Background);
lgb.Stops[6].Color = ... some new color ...;
```

El programador también puede acceder a la variable "`lgb`" una vez, almacenarla, y luego repetidamente ajustarla a esta, lo cual es muy eficiente.

La FIG. 16 es una extensión del diagrama de estado de la FIG. 15 con el estado adicional representado por una propiedad `StatusOfNextUse` agregada. Note que el modelo es solamente ligeramente más complicado, en razón a que el diagrama de estado base de la FIG. 16 tiene dos estados y siete transiciones, mientras que el diagrama de estado de mutabilidad tiene tres estados y once transiciones. Como se puede ver de la FIG. 15, una adición significativa es el estado `(StatusOfNextUse==ChangeableCopy)`, y la transición "Use" de ese estado, que da como resultado una nueva copia con el valor de propiedad `IsChangeable` se ajusta como cierto.

Como en la FIG. 15, llamar el método `Copy()` resulta en un nuevo valor con una propiedad `IsChangeable` igual a cierto, con la propiedad `StatusOfNextUse` ajustada a `Unchangeable`. Similarmente, el método `MakeUnchangeable()` llama los resultados en el valor objetivo para la propiedad `IsChangeable` ajustada a falso. Note que aunque la flexibilidad ha sido agregada por vía de la mutabilidad, estas constantes no han cambiado.

Existen algunas situaciones donde el uso del `StatusOfNextUse` igual a `ChangeableCopy` no será permitido, porque las modificaciones subsecuentes son ciertamente no definidas, o son explícitamente no permitidas. Ejemplos de esto incluyen intentar modificar valores en una hoja de estilo por defecto

554

662

compartida, o ajustar propiedades de motor de propiedad no locales. En tales situaciones, los subsistemas que no permiten tal uso escogerán levantar una excepción, o hacer el valor no cambiante ellos mismos. Se recomienda que una excepción se eleve como una indicación más clara al programador de lo que ha ocurrido y de esta manera evitar confusiones posteriores.

Además, existen situaciones en las cuales un objeto cambiante no puede ser hecho no cambiante. Ejemplos incluyen VisualBrush, (como se describe en la Solicitud de Patente Estadounidense Serie No. 10/402,268), en el cual un visual subyacente puede no ser restringido de cambio, y por lo tanto no tendría sentido establecer que el VisualBrush sea "no cambiante". Las animaciones y el VideoData (en razón a que hay tiempo variante) también son complejas. Los intentos para llamar el MakeUnchangeable() sobre tales objetos elevarán excepciones, o aún peor, pueden dejar el objeto en un estado malo, en la medida en que las porciones de ésta puedan haber sido hechas no cambiantes aunque otras no lo sean. Estos problemas se pueden evitar por vía de otra propiedad, CanMakeUnchangeable. Si esta propiedad regresa a un valor "cierto", entonces MakeUnchangeable() le garantiza el éxito, no suministrando cambios al objeto que ocurra entre estas llamadas.

663



Existe un conflicto en semánticas que ocasionalmente ocurre entre `StatusOfNextUse` y `CanMakeUnchangeable`. Si `CanMakeUnchangeable` es igual a falso, entonces un valor de `UseStatus.Unchangeable` para `StatusOfNextUse` realmente no tiene sentido, en la medida en que el siguiente uso no puede ser no cambiabile. Por lo tanto, cuando `StatusOfNextUse` es preguntado cuando `CanMakeUnchangeable` es igual a falso, este nunca regresa a `UseStatus.Unchangeable`. En su lugar, cuando de otra manera se regresaría a `UseStatus.Unchangeable`, este regresa a `Usetatus.ChangeableCopy`.

Lo anterior representa un modelo donde cada uso (calificado) de un objeto cambiabile (`IsChangeable` igual a cierto) resulta en una copia de un objeto que está siendo hecho, y, dependiendo del valor de `StatusOfNextUse`, ese "uso" puede o no puede en sí mismo ser cambiabile. Lo que no suministra el modelo anterior es el uso de un valor en lugares múltiples, y manteniendo las referencias compartidas para ese valor. Por ejemplo, en el modelo anterior un programador no puede crear un `LinearGradientBrush`, usarlo en dos controles `Button`, y luego cambiar el `LinearGradientBrush` una vez para afectar ambos controles. En su lugar el programador necesitaría utilizarlo dos veces, conseguir que la brocha regrese atrás de los controles, y luego ajustar cada una independientemente. En general, este modelo consigue ser el más esperado y/o menos sorprendente para los programadores,

61

pero existen escenarios donde es deseable la funcionalidad adicional.

Uno de tales escenarios es en Animation, donde, si un programador quiere crear una escena con n elementos que responda cada uno a la misma línea de tiempo, esa línea de tiempo necesita ser clonada n veces, y preguntada a BeginIn() n veces. Una aproximación mucho mejor, de tanto un punto de vista de desempeño como la eficiencia, así como también para programar la conveniencia del modelo, es compartir una referencia a una línea de tiempo simple, invocar BeginIn() en este, y propagarlo según sea apropiado.

Para posibilitar este escenario, un tercer valor, ChangeableReference, se suministra con la enumeración de UseStatus. UseStatus ahora aparece:

```
public enum System.Windows.UseStatus
{
    Unchangeable,
    ChangeableCopy,
    ChangeableReference
}
```

Cuando un objeto cambiabile que tiene StatusOfNextUse se ajusta a UseStatus.ChangeableReference se utiliza (de una

552

manera calificada), ese valor ya no es copiado. En su lugar, una referencia al valor existente es manejada, y las modificaciones subsecuentes a ese valor (o cualquier referencia previa o subsecuentemente dadas) afectarán el resultado del uso. En otras palabras, el valor cambiante es ahora compartido con potencialmente cualquier número de usos.

El siguiente es un ejemplo del nivel de elemento de uso:

```
Brush b = new SolidColorBrush(...);  
b.Color = Colors.Yellow;  
b.StatusOfNextUse = UseStatus.ChangeableReference;  
Btn1.Background = b;  
Btn2.Background = Btn1.Background;  
Btn3.Background = Btn2.Background;  
Btn4.Background = b;  
// at this point, all four buttons are yellow  
  
((SolidColorBrush)Btn3.Background).Color = Colors.Purple;  
// They change to purple, and they are notified.
```

En un ejemplo anterior, una simple operación de dibujo se describió la cual generó dos rectángulos, uno rojo, y uno azul:

```
SolidColorBrush scb = new SolidColorBrush();  
scb.Color = Colors.Red;  
ctx.DrawRectangle(scb, ...); // this is a "use"  
scb.Color = Colors.Green;  
ctx.DrawRectangle(scb, ...); // this is a "use"
```

Este es el comportamiento deseado. Sin embargo, si un programador en su lugar quiere que la brocha sea compartida, y aún es cambiable, se podrían usar las siguientes instrucciones:

```
SolidColorBrush scb = new SolidColorBrush();  
scb.UseResult = UseResult.ChangeableReference;  
scb.Color = Colors.Red;  
ctx.DrawRectangle(scb, ...); // this is a "use"  
scb.Color = Colors.Green; // initial rectangle becomes green  
ctx.DrawRectangle(scb, ...); // this is a "use"
```

Aquí, ambos rectángulos son verdes. Si posteriormente el color es cambiado, por ejemplo, `scb.Color=Colors.Yellow`, ambos rectángulos se volverán amarillos. Note que `ctx.DrawRectangle(...)` parece ser un comando de dibujo de modo inmediato, sin embargo este está de hecho construyendo una lista de despliegue/meta archivo para ser retenida y subsecuentemente desplegada.

67

Desde el punto de vista del modelo de usuario, el modo `ChangeableReference` garantiza que las partes que están utilizando el objeto cambiante serán notificadas de cualquier cambio a ese valor. Esto será hecho a través del evento "Changed" el cual, como otros eventos, es un delegado multimolde. Para implementación, el sistema necesita asegurarse de que múltiples usos con un sumidero de notificación simple no notifica ese sumidero de cada uso. Además, los mecanismos de limpieza tienen requisitos cuando remueven ítems, con el fin de solo remover un sumidero cuando los usos conectados a ese sumidero se han ido. Una aproximación para hacer esto es referenciar los delegados de conteo. La implementación habitual puede lograr estos requisitos por vía de una estructura de datos privada, por ejemplo, `RefCountedMulticastEventHandler`.

La FIG. 17 es un diagrama de estado basado en la FIG. 15 y 16 pero que tiene un estado `CHangeableReference` (por vía de otra configuración en la propiedad de `StatusOfNextUse`) agregada. Note que al mantener un aspecto de la presente invención el estado `ChangeableReference` y la transición `Use` separada de ese nodo no hace una copia. En su lugar, un uso calificado da como resultado el estatus de la siguiente propiedad de uso que permanece en el estado de referencia `Changeable`, suministrando de esta manera una mutabilidad cierta. Además, note que aunque la FIG. 17 es más compleja que aquellas de

560

b6d

las FIGS. 15 y 16, el comportamiento de los métodos Copy() y el MakeUnchangeable() permanecen constantes; el método Copy() aún resulta en un nuevo objeto de valor que tiene la propiedad IsChangeable de cierta y la propiedad StatusOfNextUse de Unchangeable, y el método MakeUnchangeable() dará como resultado aún en el objeto de valor objetivo tener una propiedad IsChangeable de falsa.

Se debe notar que junto con las ventajas de un conjunto simple de tipos, la presente invención suministra flexibilidad significativa a los programadores. Por ejemplo, la mayoría de los valores contruidos mediante una aplicación ordinariamente no serán mutables, en razón a que los valores inmutables son consumidores de menos recursos. Sin embargo, como se describió anteriormente, la mutabilidad está disponible, dándole a los programadores una forma potente e intuitiva para cambiar valores, particularmente valores profundos, con alto desempeño. Note también que aunque no está representada la FIG. 17, el estado en el cual se crea el nuevo tipo (la propiedad IsChangeable como cierta, la propiedad StatusOfNextUse de Unchangeable) es solamente un estado por defecto posible. Así, en implementaciones alternativas, el tipo puede estar en otro estado luego de la creación. Por ejemplo, la propiedad IsChangeable de cierta, la propiedad StatusOfNextUse de ChangeableReference) tal como por defecto a valores mutables.

561

Volviendo a una explicación de la operación de la presente invención, la presente invención suministra ventajas significativas cuando trata con propiedades profundas de un objeto, denominadas como "dotting-down". Por ejemplo, considere lo siguiente:

```
GeometryCollection g;  
g.Geometries[12].Figures[2].Segments[0].Points[17] =  
  new Point(0.2, 0.3);
```

El acceso profundo en el objeto de geometría 'g' es un ejemplo de lo que se denomina como "dotting-down". Note que el acceso a las propiedades (Geometries, [12], Figures, [2], Segments, [0], y Points) son denominadas propiedades de obtención, no de asignación; [17] es la única propiedad que accede a esos resultados en una asignación que está siendo llamada. Los lenguajes de programación generalmente no pueden distinguir entre acceso a una propiedad por el hecho de asignar un valor de propiedad más profundo contra acceso para leer un valor más profundo.

Cuando el "dotting-down" arranca de un objeto no cambiante, el número local variable se accede. Un ejemplo incluye

acceder un elemento que no se ha hecho cambiable a través del uso de la propiedad "ChangeableValue".

Cuando una propiedad pasa de un objeto cambiable, el valor resultante es cambiable también, de tal forma que este puede ser mutado. Para este fin, la propiedad de consecución sobre el padre regresa al sub-objeto directamente si ya ha cambiado, o hace un clon superficial del sub-objeto, que se ajuste en el miembro local, y regrese ese clon. Note que los atributos hacen que el código anterior, después de regresar la primera vez y localizar los clones superficiales asignantes, libre en términos de no requerir localización de amontonamiento.

Un aspecto de la presente invención es que la clonación superficial, por demanda es desarrollada solamente cuando es necesario. Esto maximiza compartir, minimiza la localización de amontonamiento, permite las modificaciones sin localización de amontonamiento, y no fuerza el concepto de clonar sobre modelo de usuario. Esto se hace más importante con árboles más profundos, y cuando se trabaja en tres dimensiones. Para este fin el método Copy() suministra la ilusión de una copia profunda, pero realmente primero solamente hace una copia superficial, y luego perezosamente hace copias más profundas según sea necesario. Tal "dotting-down" suministra mejoras de desempeño significativas.

De acuerdo con un aspecto de la presente invención, otra propiedad del cambiabile (generalmente invisible a las aplicaciones) es que un cambiabile tiene un evento cambiado (del tipo EventHandler). Cuando una propiedad de un Changeable cambia, un Changad delegado sobre el cambiabile consigue ser invocado, con el cambio cambiabile como un enviador. El acto de hacer un clon superficial a través de "dotting-down" empuja los manejadores de evento cambiado hacia abajo en el clon superficial. Esto permite que ocurran cambios subsecuentes sobre elementos más profundos y tienen los manejadores de evento propio configurado. Note que el evento Changad también existe de manera que los clientes diferentes al sistema de propiedad pueda utilizar este sistema y registrar las notificaciones.

Las modificaciones a un manejador de evento cambiado se propagan hacia abajo a los sub-objetos. Además, las modificaciones a un cambiabile en sí mismo involucra otros cambiabiles (por ejemplo, agregar un sub-objeto cambiabile a un objeto cambiabile, removerlo, y así sucesivamente) da como resultado contener manejadores de evento cambiabiles que son removidos de los viejos y empujados a los nuevos, recursivamente.

Las FIGS. 18-23 demuestran como clonar superficialmente y "dotting-down" funciona, con base en el siguiente código para este ejemplo:

```
LinearGradientBrush lgb = new LinearGradientBrush();
lgb.Stops.Add(new GradientStop(Colors.Red,0.3));
lgb.Stops.Add (new GradientStop(Colors.Blue,0.2));
lgb.Stops.Add (new GradientStop(Colors.Green,0.1));

Btn1.Background = lgb;
Btn2.Background = lgb;
```

Como se muestra en la FIG. 18, tanto el punto Button1 y Button2 de la misma brocha de gradiente lineal 1802, que tiene un nodo de paradas 1804 y las propiedades de color y posición para las paradas especificadas se disponen jerárquicamente adelante. Considere el código:

```
Btn1.Background = Btn1.Background.Copy();
```

Ejecutando este código se da como resultado una copia 1902 de una brocha de gradiente lineal 1802 que está siendo hecha y puntualizada a un Button1, como se muestra en la FIG. 19.

Ejecute el código:

63

```
LinearGradientBrush lgb=((LinearGradientBrush)
Btn1.Background);
Lgb.Stops[1].Color=Colors.Orange;
```

Suministra acceso a una propiedad valorada `Changeable` de un objeto `Changeable` que tiene `IsChangeable == cierto`, significando que lo que se recupera es garantizado para ser escribible. Como generalmente se representan en las FIGS. 20-22, ejecutar este código da como resultado (1) otro nodo de paradas 2004 insertado en la jerarquía que puntualiza a cada una de las paradas individuales (FIG. 20); (2) una copia 2110 (FIG. 21) del segundo nodo de parada (nodo de parada [1], etiquetado 2010 en las FIGS. 20 y 21, que tiene una propiedad "azul" adelante) para ser hecha, de tal forma que el padre de esta copia, el nodo de paradas previamente copiado 2004, tiene como su hijo esta copia 2110 (en lugar del nodo de parada original 2010 para la propiedad azul); y (3) cambia la propiedad azul de este nodo 2110 a naranja, como se representa en la FIG. 20. Note que Orange es un tipo de valor denotado por las formas diamante en las figuras, y los cambios subsecuentes no dan como resultado localización, tal como el cambio al color Red en la FIG. 23.

Cuando se está en un estado no cambiabile, un objeto cambiabile puede ser leído de un escrito o cualquier contexto. Si en un estado cambiabile, entonces un `UiContext`, determinado en el

366

674

tiempo de construcción, se puede utilizar para asociarse con el objeto cambiabile con el fin de solo permitir acceso desde el contexto. Si `MakeUnchangeable` es posteriormente llamado, el contexto va a nulo. También, en cualquier momento un `Copy()` de un cambiabile se hace, la nueva copia consigue el `UIContext` del llamador, no del contexto de la fuente cambiabile. El API suministra una propiedad de consecución de solamente `UIContext` en el cambiabile el cual es nulo cuando es inmutable. Esta propiedad es pública de tal forma que las aplicaciones pueden decir si un objeto dado puede ser accedido.

Los objetos cambiabiles contruidos con nulidad pasan el constructor que será definido con un `UIContext` nulo. Si un objeto cambiabile tiene un `UIContext` nulo y la propiedad `IsChangeable` se ajusta como cierta, entonces la aplicación necesita manejar cualquier tema de contención de hilo posible que pueda surgir. En este caso, el sistema no evita el acceso concurrente desde contextos múltiples.

Una situación puede surgir por medio de la cual un objeto cambiabile se intenta encajar en otro objeto cambiabile sobre este (por ejemplo, ajustando una parada de gradiente en una brocha de gradiente lineal), y el `UIContext` no casa. Por ejemplo, considere un `LinearGradientBrush` `lgb` que tiene un `UIContext` de `A`, aunque el `GradientStop` `gs` tiene tanto

5. 2

UIContext de B, como un StatusOfNextUse igual a ChangeableReference. Intentar ajustar gs en lgb dará como resultado que surja una excepción, en razón a que este es un intento para mezclar UIContext, lo cual no está permitido.

Cuando se hace un cambio a un objeto cambiabile, el evento Changed surge, y ese cambiabile es suministrado como el objeto enviador para el manejador de evento. Sin embargo, existen situaciones cuando enviar el objeto que de hecho cambiado no es deseable, y en donde tener un objeto diferente como el enviador sería de más ayuda. Un ejemplo de esto es con objetos animados, donde una animación (en sí misma un cambiabile) se mantiene sobre una línea de tiempo (algunas veces denominada como reloj) que describe su comportamiento de animación. Eventos tales como Pause() ocurren en la línea de tiempo, y no la animación, pero generalmente las aplicaciones quieren saber que la animación ha pausado.

Varias soluciones son posibles, tales como disparar eventos cambiados arriba de la cadena de cambiabiles. Esto presenta un número de problemas, incluyendo decidir donde parar, originando tormentas de notificación con muchos más eventos disparados aún cuando nada está recibiendo y/o utilizando los eventos, y que los cambiabiles, por diseño, no conocen sus padres, pero en su lugar generalmente solo conocen qué notificar en el evento de un cambio. Tener un esquema en el

cual los cambiables siguen la pista de sus padres requeriría almacenamiento adicional y "bookkeeping". A pesar de esto, tal solución se puede implementar.

Otra solución es implementar un cambiable de tal forma que el enviador es el objeto cambiable, y no el cambiable interior que de hecho ha cambiado. El `PropagateEventHandler` se implementa para no empujar hacia abajo el manejador que este recibe, sino en su lugar almacenar el manejador desprendido, crear un nuevo manejador local que, cuando se invoca, llama el manejador almacenado, pero con el cambiable como el argumento de enviador. El nuevo manejador local es empujado hacia abajo al `PropagateEventHandler` sobre los hijos cambiables. Note que esta técnica intercepta cada manejador de evento, que requiere que el `PropagateEventHandler` sea correctamente manejado cuando se llama con "falso" (cuando los manejadores tienen que ser removidos), requiriendo así que el "bookkeeping" sea desarrollado.

Note que esta solución no tiene un mecanismo de alcance `BeginChange/EndChange` explícito, haciéndose más directo y robusto en la cara de las excepciones (en razón a que no hay una modalidad involucrada, y ningún `EndChange()` que pueda conseguir omitir ser pasado por una excepción inesperada). Sin embargo, `Begin/EndChange` existió para permitir que los "obtentores" de objetos no tengan que hacer clones

63

superficiales de los valores que ellos están consiguiendo, cuando aquellos clones fueron congelados y el sistema no estaba en modo escrito. De otra manera, aquellos valores congelados, en modo escrito, consiguen que sea hecho un clon superficial de ellos. Como resultado, el árbol es explotado más frecuentemente que con Begin/EndChange, y puede hacerlo así cuando no existe absolutamente ningún ajuste involucrado, solo una consecución. A pesar de esto, si está siendo invocada una consecución sobre un valor no cambiabile para iniciar, no se hará un clon superficial (note que esto es distinto de una consecución que está siendo invocada sobre un valor cambiabile, y el valor que es obtenido por vía de "conseguir" es no cambiabile, lo cual es donde la operación de clonación ocurre).

Por vía de ejemplo, si acceder al Btn.Background.Opacity, y al Btn.Background no es cambiabile (por ejemplo, como por defecto), entonces copiar no se hace. En su lugar, la ocurrirá cuando "Btn.Background = Btn.Background.ChangeableValue" o algo similar tiene lugar, significando que el gasto de copiar solo ocurre si se utiliza. En otras palabras, en un intento para cambiar el valor no se expresa, entonces la "consecución" arbitraria no incurre en el costo de la copia. Note que si los valores mantienen una noción de su "último clon creado" entonces en clon puede ser manejado sobre el uso del objeto, en tanto que

594

677

el objeto no ha sido cambiado en razón a que el clon fue hecho (tales cambios solo dan como resultado permitir irse de ese clon de caché). Esto permite más compartir. Además, note que los implementadores de control no están indebidamente cargados al participar en este patrón como lo está el patrón a ser útil a los usuarios. Similarmente, si el tipo de extensibilidad se suministra, los MediaTypes escritos no deben ser exageradamente complejos.

El implementador de control es presentado con el mismo modelo para tratar con un Changeable como con cualquier otro valor. Por ejemplo, el siguiente código suministra un control Grid con una propiedad AlternateBrush de tipo Brush:

```
public static readonly DependencyProperty AlternateBrushID = ...;
private Brush _alternateBrush;
public Brush AlternateBrush
```

```
{
    get
    {
        return ComputeValue(AlternateBrushID,
                            ref _alternateBrush);
    }

    set
    {
        WriteLocal(AlternateBrushID, value);
    }
}
```

577

67A

Note que esto es idéntico a una propiedad general que participa en el sistema de propiedad. Eso es porque WriteLocal hará manejo especial para propiedades profundas que derivan de la clase Changeable.

El implementador del tipo cambiable necesita un preámbulo de una línea y un postscript de una línea sobre cualquier cosa que modifique el cambiable (por ejemplo, propiedades). También, un preámbulo de una línea simple se necesita sobre objetos que accedan al estado de un cambiable (por ejemplo, consecución de propiedad). Son necesarias las implementaciones de CloneCore(), MakeUnchangeableCore(), PropagateEventHandlerCore(), PropagateEventHandlers() (note que las últimas tres solo son necesarias para tipos que tienen otras Changeables como propiedades), y también se necesita un typesafe-wrapper para Copy().

Los siguientes ejemplos vienen del prototipo referenciado, incluyendo un ejemplo (artificial) de un GradientStop el cual es un tipo cambiable simple (que está simplificado porque ninguno de sus sub-tipos son cambiables). Note que en la práctica, muy pocos cambiables será este simplificado, porque nada que contenga una colección de animación (en sí misma un cambiable) será más complejo:

572

680

```
public class GradientStop : Changeable
{
    public GradientStop()
```

```
    {
    }

    public GradientStop(Color color, float location)
    {
        _color = color;
        _location = location;
    }

    public Color Color
    {
        get
        {
            ReadPreamble();
            return _color;
        }
        set
        {
            // This is the pattern for setting simple value types.
            WritePreamble();
            if (_color != value)
            {
                _color = value;
                WritePostscript();
            }
        }
    }

    public float Location
    {
        get
        {
```

543

```

    ReadPreamble();
    return _location;
}

set
{
    // This is the pattern for setting simple value types.
    WritePreamble();
    if (_location != value)
    {
        _location = value;
        WritePostscript();
    }
}

// Create a new, typesafe version of Copy().
public new GradientStop Copy()
{
    return (GradientStop)base.Copy();
}

protected override Changeable CloneCore(bool shallowClone)
{
    // Shallow and deep are treated the same here when
    // all members are value types
    return new GradientStop(_color, _location);
}

private Color _color;
private float _location;
}

```

282

El siguiente es un ejemplo de un LinearGradientBrush que es un tipo cambiabile complejo (porque algunos de sus sub-tipos, a saber GradientStops, son en sí mismos cambiabiles):

```
public class LinearGradientBrush : Brush
{
    public LinearGradientBrush()
    {
    }

    public LinearGradientBrush(GradientStop stop1, GradientStop stop2,
                                double opacity)
        : base(opacity)
    {
        // Assigning through the properties results in "using" the
        // parameters.
        Stop1 = stop1;
        Stop2 = stop2;
    }

    public GradientStop Stop1
    {
        get
        {
            _stop1 = (GradientStop)EmbeddedChangeableGetter(_stop1);
            return _stop1;
        }
        set
        {
            if (_stop1 != value)
            {
                _stop1 = (GradientStop)EmbeddedChangeableSetter(_stop1, value);
            }
        }
    }
}
```

575

63:

```

        WritePostscript();
    }
}

public GradientStop Stop2
{
    get
    {
        _stop2 = (GradientStop)EmbeddedChangeableReader(_stop2);
        return _stop2;
    }
    set
    {
        if (_stop2 != value)
        {
            _stop2 = (GradientStop)EmbeddedChangeableWriter(_stop2, value);
            WritePostscript();
        }
    }
}

// In place modification to r/o (or check whether it can be done).
// Helpers check if arguments are null.
protected override bool MakeUnchangeableCore(bool checking)
{
    return Changeable.MakeUnchangeable(_stop1, checking);
    && Changeable.MakeUnchangeable(_stop2, checking);
}

// Propagation of event changes
protected override void PropagateEventHandlerCore(EventHandler handler,

```

576

681

```

        bool adding)
    {
        Changeable.ModifyHandlerIfChangeable(_stop1, handler, adding);
        Changeable.ModifyHandlerIfChangeable(_stop2, handler, adding);
    }

    protected override Changeable CloneCore(bool shallowClone)
    {
        // Always use member fields, not properties, for cloning.
        // Otherwise you inadvertently cause extra shallow copying.
        if (shallowClone)
        {
            return new LinearGradientBrush(_stop1, _stop2, _opacity);
        }
        else
        {
            return new LinearGradientBrush(
                (GradientStop)Changeable.CloneDownToUnchangeable(_stop1),
                (GradientStop)Changeable.CloneDownToUnchangeable(_stop2),
                _opacity);
        }
    }

    private GradientStop _stop1;
    private GradientStop _stop2;
}

```

El modelo de objeto cambiable está dividido en una parte pública, y en una parte que es la que los entendedores y los albergadores ven. Note de nuevo que, sin embargo, que las

572

cosas son directas para el escritor componente que usa estos tipos.

```
// PUBLIC API Everyone expected to use this portion of the API.
// Anything outside of this section will be hidden
```

```
// from the editors.
public abstractclass System.Windows.Changeable
{
    public bool IsChangeable { get; } // defaults to true
    public Changeable Copy();
    public void MakeUnchangeable();
    public bool CanMakeUnchangeable { get; }
    public UseStatus StatusOfNextUse { get; set; } // defaults to false
    public event EventHandler Changed { add; remove; }
    public UIContext UIContext { get; } // null if IsChangeable==false

    public bool AllowChangeableReferenceOverride { get; set; }
        // defaults to false

    // protected stuff shown below.
}

Public enum System.Windows.UseStatus
{
    Unchangeable,
    ChangeableCopy,

    [EditorState.Advanced]
    ChangeableReference
}
```

```
public class System.Windows.Changeable
{
    // COMPONENT WRITER API. This is for use by Component Writers
```

```

// (Note: there is nothing that component writers need to know about)

// EXTENDER API. This is for use by those who write new Changeables

protected Changtable(); // use UIContext.CurrentContext
protected Changeable(UIContext context);

// Subtypes implement this to provide cloning as described above.

protected abstract Changeable CloneCore(bool shallowClone);

// Subclasses implement to freeze their complex subtypes (when
// 'checking' is false, or to check whether they can (when
// 'checking' is true. Note that when checking is false, the return
// boolean doesn't really make sense, but implementers are to return
// 'true' in this case, so that calling code (like the above example)
// will work correctly. If a call is made with 'true' that cannot
// actually make the value unchangeable, then the implementation should
// raise an exception.
// The default implementation returns true in these cases.
protected virtual bool MakeUnchangeableCore(bool checking);

// Subclasses implement to push down changes to event handlers
// that happen up above (default impl is empty). If adding is true
// then the handler should be added to sub-changeables, else it
// should be removed from them as described above
protected virtual void PropagateEventHandler(EventHandler handler,
                                              bool adding);

// Subclasses implement to validate that their in a good state. They

```

687

```

// throw a meaningful exception if not. This will be called on
// WritePostscript(), EmbeddedChangeableWriter().
// Default impl is empty.
protected virtual void ValidateObjectState();

// Called when there's a change to the value. Default implementation
// invokes registered Changed handlers.
protected virtual void OnChanged();

// Call from anything that reads an embedded changeable out of
// a changeable (Reader) or writes an embedded changeable into
// a changeable (Writer). The primary uses of these are for property
// sets and gets. See examples above.
protected Changeable EmbeddedChangeableReader(Changeable property);
protected Changeable EmbeddedChangeableWriter(Changeable originalChangeable,
                                                Changeable newChangeable);

// Call from simple property setters and getters, or anything that
// access non-Changeable state on a Changeable. See examples above.
protected void ReadPreamble();
protected void WritePreamble();

// Call from both simple and complex property setters and anything else
// that sets state on a Changeable. See examples above.
protected void WritePostscript();

//// Static helper methods

// Create a deep, unchangeable clone of the provided Changeable, but stop at nodes
// that are already unchangeable (since those can just be "cloned" by handing
// out references to them). If null is passed in, null is returned
static protected Changeable CloneDownToUnchangeable(Changeable changeable);

```

582

```

// Make the specified Changeable be unmodifiable (IsChangeable == false) and
// returns true when the 'checking' parameter is 'false'. Otherwise, it checks
// to see if the changeable can be made unchangeable, returning true or false
// as appropriate. If null is passed in as the changeable, this operation
// doesn't do anything and returns 'true'.
static protected bool MakeUnchangeable(Changeable changeable, bool checking);
}

// UsageOverride to be used in ChangeableHelper.UseChangeable
public enum ChangeableUsageOverride
{
    NoOverride, // default, use as prescribed
    ForceUnchangeable, // force the "use" to be unchangeable
    PreferChangeableReference // as described above.
}

// This helper class exists to reduce OM clutter on
// the core Changeable type.
public class System.Windows.ChangeableHelper
{
    // HOSTING API. This is for use by systems other than the Property System
    // that are going to host these values. The Property System will use
    // these entry points for their own hosting in WriteLocal

    // remove handler from getting notified of changes to oldChangeable
    static public void UnsubscribeChangeable(Changeable changeable,
                                              EventHandler handlerToRemove);

    // return a Changeable, adding handler to getting notified of changes to it,
    // if any. Note that this is a "use" of the Changeable, and the returned value
    // will follow the "use" semantics. If "usageOverride" is NoOverride, all

```

689

```

// proceeds as expected, if it's "ForceUnchangeable", then
// always treat the Changeable as StatusOfNextUse UseStatus.Unchangeable.
// See above for a description
static public Changeable UseChangeable(Changeable changeable,
                                     EventHandler handlerToAssociate,
                                     ChangeableUsageOverride usageOverride);
}

```

Note que el acto de un uso calificado de un Changeable, que confía solamente en su StatusOfNextUse, no trabaja exactamente en cada situación. En general, el problema es que cuando un objeto cambiable (tal como Brush, VideoData, y así sucesivamente) consiguen ser asignados en una propiedad Element (como VideoSource), ese cambiable es "utilizado" en un uso calificado. En el caso de Changeables animados (tales como VideoData, pero también cualquier animación), el acto de "utilizar" crea un clon, el cual es el comportamiento correcto y esperado. Entonces, cuando el método OnRender() de elementos es invocado, la implementación OnRender() típicamente empuja el valor hacia un DrawingContext, por ejemplo, a través del DrawingContext.DrawVideo(videoData,...). Esto llama a que dentro del DrawingContext también "se utilice" el Changeable (videoDat en este caso), dando como resultado que se haga otro clon.

Jgt

Ambos comportamientos de cuando "se utilizan" los cambiables de esta manera son correctos y tienen sentido, cuando se consideran aisladamente. Sin embargo, el problema ocurre cuando ellos combinan, en que el implementador de control no espere un uso calificado cada vez que el `OnRender()` es invocado, y no hay beneficio de hacerlo así, en razón a que este uso no es expuesto a la aplicación, y de hecho, es pura sobrecarga que debe ser eliminada. Más aún, cuando las animaciones dependientes y las animaciones independientes se combinan, `OnRender()` será frecuentemente llamado, y las animaciones serán repetidamente copiadas, lo cual no es un comportamiento correcto. Un mecanismo denominado `ChangeableReference` le permite a un "uso" que de hecho ~~no~~ se copie, pero en su lugar solo obtener una referencia al valor que estaba siendo utilizado.

Una solución es tener cooperación entre una entidad como `DrawingContext` y `DependencyProperties` sobre un `DependencyObject`. En particular, el `DependencyProperty` de un control, cuando tiene un valor ajustado a este, necesita indicar que este "permitirá" que el `Changeable` sea tratado como un `ChangeableReference` si el contexto particular en el cual este es subsecuentemente utilizado quiere ser como tal. Entonces, las operaciones `DrawingContext` dicen indicar que ellas "preferirían" que el `Changeable` sea tratado como un

ChangeableReference, suministrado el Changeable en sí mismo lo permitirá.

Para hacer esto, se suministra una propiedad Boolean denominada Changeable.AllowChangeableReferenceOverride y la enumeración ChangeableUsageOverride, utilizada en ChangeableHelper.UseChangeable. En esta implementación, UseChangeable trabaja como antes con el mapeo cierto/falso al ForceUnchangeable/NoOverride. Si UseChangeable es llamado con PreferChangeableReference y el cambiable tiene IsChangeable==cierto, y el cambiable tiene allowChangeableReferenceOverride==cierto, entonces el uso del Changeable será como un ChangeableReference.

Este se utiliza al tener DependencyObject.SetValue() ajustado al Changeable que está retenido (cuando este es modificable) a AllowChangeableReferenceOverride, y tener métodos DrawingContext que invocan UseChangeable con UsageOverridePreferChangeableReference.

Note que cuando ambas condiciones no son ciertas, el comportamiento correcto también ocurre, porque Elt2.Prop = Elt1.Prop utilizará la propiedad como se espera en uso calificado, copiándola si esta es modificable, a menos de que esta ha sido explícitamente ajustada a ChangeableReference, en razón a que UseChangeable no será invocado con el

PreferChangeableReference. El uso directo del DrawingContext también funcionará apropiadamente, en razón a que los cambiables que están siendo enviados hacia abajo a este no tendrán AllowChangeableReferenceOverride. .

Note que cuando existe un cambiable donde el sub-objeto de este es ChangeableReference, un clon superficial y un clon profundo se pueden hacer. Un Shallow Clone debe trabajar, en la medida que el método CloneCore creará una nueva "cubierta" superficial, y asignará sobre los hijos, que no irán más profundo dentro de ellos. Con los clones profundos, el proceso es directo en el caso de un árbol de ChangeableCopys y Unchangeables , al clonar los Unchangeables, haciendo cada clon de la misma manera que este sea Unchangeable (asumiendo que CanMakeUnchangeable es cierto). Esto da como resultado un clon profundo donde el nivel superior es Changeable, y todo lo de adelante es Unchangeable. Note que "dotting-down" hará los sub-elementos modificables de nuevo.

Sin embargo, cuando existe un ChangeableReference, la operación del clon necesita ser efectivamente desarrollada, sin embargo se mantiene una referencia para una senda "Changeable" abajo del ChangeableReference. Esto se necesita de tal forma que cuando existe una notificación de un ChangeableReference, los manejadores correctos conseguirán invocar todo lo que esté albergado.

Considere el siguiente ejemplo:

```
Brush b = new LinearGradientBrush();
b.Stops = new GradientStopCollection();
GradientStop gs = new GradientStop();
gs.StatusOfNextUse = UseStatus.ChangeableReference;
b.Stops.Add(gs);

Button button1, button2;

button1.Background = b;
button2.Background = b; (or button2.Background = button1.Background)

gs.Color = Colors.Purple;
```

Aquí, se crea un LinearGradientBrush, como es su colección Stops y un Stop simple, y se hace un ChangeableReference. La brocha puede ser utilizada en múltiples lugares, y la modificación al ChangeableReferenceGradientStop necesita afectar ambas brochas.

EJEMPLOS (VÁLIDO E INVÁLIDO) DE UTILIZAR OBJETOS CHANGEABLE

La siguiente sección suministra un resumen de utilizar y manipular objetos que derivan de la clase Changeable, por cuyos objetos, tales como brochas, lápices, y animaciones,

694

son mutables controlados por un programador. Las clases que derivando `Changeable` simulan mutabilidad al construir automáticamente una versión inmutable de ellos mismos cuando se utilizan en un uso calificado. Como se describió anteriormente, un `Changeable` es considerado utilizado en un uso calificado cuando el objeto se ajusta en una propiedad `Property System`, utilizada como un sub-objeto en un objeto `Changeable` complejo, o utilizado en un comando `DrawingContext`.

Cuando se desarrolla la aplicación con tales objetos, los objetos del sistema de gráficos y medios son generalmente creados, ajustados, utilizados, y entonces nunca modificados. Por ejemplo, para ajustar el trasfondo de un `Button` un programador podría utilizar un `SolidColorBrush`, que deriva de un `Changeable`; pero el programador nunca podría modificar el trasfondo del botón de nuevo durante el transcurso de la aplicación. El siguiente es un ejemplo:

567

695

```

// C#
SolidColorBrush myBrush = new SolidColorBrush(Colors.Yellow);
myBrush.Opacity = 0.5;

// A "use" of myBrush.
myButton.Background = myBrush;

' VB .NET
Dim myBrush As new _
    System.Windows.Media.SolidColorBrush(System.Windows.Media.Colors.Yellow)
myBrush.Opacity = 0.5

' A "use" of myBrush.
myButton.Background = myBrush

```

Cuando se utiliza de esta manera, un `Changeable` se comporta como un tipo de valor, tal como un `Rect` o `Color`. El `Changeable` es copiado sobre su destino y el cambio al original no afecta los cambios al valor utilizado. Sin embargo, existen situaciones donde el programador podría necesitar modificar tal objeto después de que ha sido utilizado. Por ejemplo, supóngase que el programador quiere cambiar el trasfondo del botón en el código previo después de que el usuario lo ha obturado.

El patrón `Changeable` existe para cumplir las necesidades de una variedad de situaciones tales como una anterior. En

586

696

general, un Changeable es un valor que o no puede ser modificable, denotado por el valor de la propiedad IsChangeable. Intentando modificar el valor cuando IsChangeable es falso resulta en una excepción. Adicionalmente, los objetos IsChangeable que pueden ser modificados sacan sus eventos Changed cuando ellos cambian o cuando cualquiera de sus miembros es cambiado. Cuando trabajan con Changeables, es así importante entender cuando Changeable es "utilizado" en un uso calificado.

Por defecto, cuando un objeto Changeable es utilizado en un uso calificado, una copia inmutable es creada y esa copia es de hecho utilizada. La copia tiene un valor IsChangeable de falso. El siguiente código origina que una excepción sea lanzada en razón a que el código intenta modificar la copia no modificable de myBrush que fue utilizada para ajustar el trasfondo del botón:

567

697

La clase no puede ser públicamente iniciada porque no existe un medio "por defecto". El VisualRederer también es un objeto ContextAffinity.

Una propiedad BackgroundColor se suministra:

```
public Color BackgroundColor { get; set; }
```

Este es un color de transfono por defecto del manejador visual, que puede por defecto hacer esto transparente para los VisualManagers. Sin embargo, algunos medios, tales como representar a legacy HWnds) no pueden soportar por transferencia de píxel, y así cada VisualManager puede definir su propio por defecto para está propiedad. La mayoría de las aplicaciones ignorarían está propiedad, y por ejemplo podrían ajustar el color de transfono de la ventana del sistema o hacerla transparente.

La propiedad RootVisual identifica el visual de raíz para representar:

```
public Visual RootVisual { get; set; }
```

527

698

Este defecto anula. Cuando la propiedad RootVisual es nula el VisualManager dibuja el color de transfondo sobre 1 medio.

La propiedad RootSize regresa, en unidades virtuales, al tamaño del objetivo representador. Por ejemplo, para un VisualManager puesto atrás por una ventana, este será el tamaño del cliente de la ventana:

```
public abstract Size RootSize { get; }
```

La Resolution Information también se suministra:

```
public ResolutionInformation ResolutionInformation { get; set; }
```

En cada medio se requiere para tener una resolución de dispositivo aun si esta no es puesta atrás por píxeles. Por ejemplo, cuando se imprime, aun si se captura un meta archivo, se necesita que la información de resolución se haga disponible a través del VisualManager de tal forma que el contenido se pueda utilizar para esa resolución. Nótese que en el caso de una captura de meta archivo, en una resolución por defecto relativamente alta se puede utilizar, aunque permitiéndole al usuario configurar la resolución directamente.

J25

699

La configuración de transformación inicial "World to device" para el RootVisual la hace de tal forma que una unidad en ese visual es igual a $1/96^\circ$ de cada pulgada en el dispositivo. Por ejemplo, si existe un ScreenVisualManager que está expuesto atrás por un dispositivo que es de 192 dpi, entonces la transformación inicial debe ser configurado de tal forma que la unidad en la del cuadro de coordenada para el RootVisual deba ser igual a dos unidades en el dispositivo. En este caso, el ResolutionInformation.PixelSize regresará (0,5, 0,5) para significar es $1/48^\circ$ de una pulgada en un lado.

Un VisualManager establece una conexión de largo plazo con el visual raíz y el objetivo representador y rastrea diferencias:

```
public System.Windows.Media.Visuals.VisualManager :
    System.Windows.Media.Visuals.VisualRenderer
{
    // Constructors
    internal VisualManager();
    internal VisualManager(System.Threading.UiContext context);
```

320

700

```

// Visual tree dirty
public bool IsVisualTreeDirty { get; }
public event EventHandler VisualTreeDirtyChanged { add; remove; }
}

```

La WindowVisualManager es una forma primaria de dibujar en la pantalla. Está maneja representando un VisualTree o un WindowContext:

```

public sealed class System.Windows.Media.Visuals.ScreenVisualManager :
    VisualManager
{
    // Constructors
    public ScreenVisualManager(WindowContext windowContext);
    public ScreenVisualManager(WindowContext windowContext,
        System.Threading.UiContext context);

    // WindowContext
    public WindowContext WindowContext { get; set; }
}

```

CONTEXTO DE DIBUJO

Los DrawingContext API representan un modelo de programación "basado en contexto" familiar para aquellos expertos en la técnica, para como un contenido visual de construcción que puebla un Visual o son representados en un ImageData. Esta

528

701

sección describe las clases `DrawingContext`, así como también las clases y puntos de entrada necesarios para adquirir el `DrawingContext` y enumerar el contenido visual en un `RetainedVisual/DrawingVisual`.

Las aplicaciones no construyen directamente un `DrawingContext`, y las versiones expuestas del `DrawingContext` son clases abstractas. Existen un número de maneras de adquirir un `DrawingContext` para poner contenido visual en estas. Estas incluyen `RetainedVisual.RenderOpen()` o `RetainedVisual.RenderAppend()`, la cual regresa cada una un `DrawingContext` unas instrucciones emitidas en está. Otras maneras incluyen `IRetainedRender.Render()`, `DrawingVisual.RenderOpen()`, `DrawingVisual.RenderAppend()`, y `PaintVisual.PaintingOpen()` o `PaintVisual.PaintingAppend()` (aunque `PaintingVisuals` no posee animaciones). El `ImageData` (o una subclase del `ImageData`) tiene un mecanismo para regresar un `DrawingContext` para representarlo sobre una superficie de mapa de bits de resolución fija. El `ImageData` tampoco procesa animaciones.

Lo siguiente establece el `DrawingContext` API:

528

702

```

public abstract class System.Windows.Media.StaticDrawingContext :
    ContextAffinityObject, IDisposable
{
    // No public constructors

    // Vector graphics drawing methods
    public abstract void DrawLine(Pen pen, Point point0, Point point1);
    public abstract void DrawLine(Pen pen,
        Point point0, PointAnimationCollection point0Animations,
        Point point1, PointAnimationCollection point1Animations);

    public abstract void DrawRectangle(Brush brush, Pen pen, Rect rectangle);
    public abstract void DrawRectangle(Brush brush, Pen pen,
        Rect rectangle, RectAnimationCollection rectangleAnimations);

    public abstract void DrawRoundedRectangle(Brush brush, Pen pen, Rect rectangle,
        double radiusX, double radiusY);
    public abstract void DrawRoundedRectangle(Brush brush, Pen pen,
        Rect rectangle, RectAnimationCollection rectangleAnimations,
        double radiusX, double radiusY);

    public abstract void DrawEllipse(Brush brush, Pen pen,
        Point center, double radiusX, double radiusY);
    public abstract void DrawEllipse(Brush brush, Pen pen,
        Point center, PointAnimationCollection centerAnimations,
        double radiusX, DoubleAnimationCollection radiusXAnimations,

```

527

705

```

    double radiusY, DoubleAnimationCollection radiusYAnimations);

public abstract void DrawGeometry(Brush brush, Pen pen, Geometry geometry);

public abstract void DrawDrawing(Drawing drawing, Point location);

public abstract void DrawDrawing(Drawing drawing,
    Point location,
    PointAnimationCollection locationAnimation);

// Image and Video Drawing
public abstract void DrawImage(ImageData image, Rect rectangle);
public abstract void DrawImage(ImageData image,
    Rect rectangle, RectAnimationCollection rectangleAnimations);

public abstract void DrawVideo(MediaData video, Rect rectangle);
public abstract void DrawVideo(MediaData video,
    Rect rectangle, RectAnimationCollection rectangleAnimations);

// Text Drawing
public abstract void DrawText(FormattedText text, Point origin);
public abstract void DrawText(FormattedText text,
    Point origin, PointAnimationCollection originAnimations);

public abstract void DrawGlyphs(GlyphRun glyphRun);

// Context state methods
public abstract void PushClip(Geometry clipGeometry);
public abstract void PushOpacity(double opacity);
public abstract void PushOpacity(double opacity,

```

530

704

```

        DoubleAnimationCollection doubleAnimations);
    public abstract void PushTransform(Transform transform);
    public abstract void PushBlendMode(BlendMode blendMode);
    public abstract void Pop(); // applies to last Pushed state.

    // Quality metrics

    // Enumeration hints
    public abstract bool PreserveReadbackOrder { get; set; } // default = false

    // Close() can be called without all Push's having been Pop'd.
    public abstract void Close(); // same as IDisposable.Dispose();
}

```

La mayoría de los métodos del objeto del `DrawingContext` son autos explicativos para aquellos expertos en la técnica, sin embargo se debe notar que el `DrawingContext` es un `ContextAffinityObject` y se debe utilizar solamente de un `UIContext simple`.

Los objetos del `DrawingContext` son también `IDisposable`s, y el patrón recomendado, en C#, es utilizarlo en una cláusula de "uso" si se recibe de un `RenderOpen/Append`, por ejemplo. Además, nótese que los métodos tales como el `DrawArc`, `DrawPie`, `DrawBezier` y `DrawPolyline` no están aquí. Estos requieren construcciones de `Geometry` relevante y uso de `DrawGeometry` (como se describe adelante).

531

Adicionalmente, existen múltiples métodos Push*, solo existe un método Pop simple. Esto implica que no pueden atributos traslapados. Los atributos establecidos por el Push*() se componen apropiadamente. Por ejemplo, el Clip compone por vía del operador Intersection, el Opacity por vía de la operación multiplication, y el Transform por vía de la operación ComposeTransform.

En razón a que un usuario puede llamar, por ejemplo, DrawGeometry, con un Brush nulo, o con un Pen nulo, es válido llamarlo con nulos para ambos Brush y Pen, sin embargo no habrá nada presentado o ensayos de golpe de nuevo.

Cualesquiera propiedades animadas suministradas por el contexto de dibujo cuando se utiliza en un objetivo no animado (por ejemplo, cuando se representa directamente un raster) ajustará a un código de tiempo 0 que fue escrito para DrawingContext dinámico a una transición más fácil sobre la utilización de un DrawingContext que no procesa animación.

La enumeración del contenido interior de un Visual "retenido" está cercanamente relacionado con el DrawingContext por lo cual ese contenido fue insertado. La enumeración de los contenidos de un Visual y/o modificación de la corriente de instrucción puede ser desarrollada por vía de un mecanismo

70

Changeable, si se desea, como se describe adelante. La idea general es suministrar un mecanismo de enumeración "modelo de empuje".

La clase `DrawingContext` misma suministra una interfase para registrar el caminado del contenido de modo de empuje, una subclase del `DrawingContext` llamada `DrawingContextWalker`, (con la mayoría de los métodos permaneciendo abstractos). La subclase de usuarios subclasifican `DrawingContextWalker`, y luego pasan una instancia en el `Visual` para iniciar la enumeración. El `Visual` llama de regreso el método `DrawingContext` apropiado para comunicar el contenido que este tiene. El `DrawingContextWalker` también suministrará una pieza de estado que se puede manipular sobre esta para controlar como progresa la enumeración (es decir si se debe detener inmediatamente, por ejemplo).

El siguiente es el código de ejemplo de un objeto `DrawingContextWalker`:

533

207

```

public class MyContextWalker : DrawingContextWalker
{
    public void DrawLine(Pen pen, Point point0, Point point1)
    {
        Console.WriteLine("Line from {0} to {1} with Pen {2}", point0, point1, pen);
    }
    public void DrawRectangle(Brush brush, Pen pen,
        Rect rectangle, RectAnimationCollection rectangleAnimations)
    {
        Console.WriteLine("AnimateRectangle with ...", ...);
    }
    // etc, etc...
}

// Get the contents of any RetainedVisual (including Controls, DrawingVisuals, etc)
MyContextWalker ctxWalker = new MyContextWalker ();
ctxWalker.WalkContent(myFancyControl, 0); // results in invocations on ctxWalker

```

El método WalkContent se suministra en la clase base DrawingContextWalker:

```

public abstract class DrawingContextWalker : DrawingContext
{
    protected DrawingContextWalker(); // this lets public subclasses start happening

    public void WalkContent (RetainedVisual visual,
        DrawingContextWalkOptions options);
    protected void StopWalking();

    // implementation of abstract methods from DrawingContext

    // This doesn't make sense on the context walker, and will throw an exception
    // if accessed.

```

534

208

```

public bool PreserveReadbackOrder { get; set; }

// DrawText() expands into GlyphRun's, so it will never be called back,
// therefore we don't want the user to have to implement it.
public void DrawText(... regular ...); // should assert if ever called
public void DrawText(... animate ...); // should assert if ever called

// other methods remain abstract, and the subclasser implement.
}

```

Los usuarios subclasifican esto e implementan todos los métodos abstractos según se desee. Los usuarios entonces crean una instancia del objeto y llaman el WalkContent sobre este. El WalkContent llama entonces de regreso los métodos apropiados en la medida en que estos caminan el Visual. La implementación de cualquiera de los métodos puede tener el caminar, si se desea, al llamar el método protegido StopWalking (). Nótese que es ilegal iniciar una caminata cuando el DrawingContext sobre el Visual está abierto para representarlo dentro.

Las opciones determinan como procede el caminar:

535

29

```
[Flags]
public enum DrawingContextWalkOptions
{
    IncludeAnimations = 0x1
}
```

Si `includeAnimations` es ajustado, el caminador invoca los métodos apropiados con el contenido animado. De otra forma, el valor instantáneo del contenido se suministra a los métodos `DrawingContextWalker`.

El `DrawingContext` tiene un `PreserveReadbackOrder` `Boolean` que puede suministrar el orden o estructura del contenido al Visual con el fin en el orden y estructura regresado por vía de la clase `DrawingContext`. Este hace por defecto a falso, pero se puede ajustar a cierto antes de que se inserte el contenido cuando es importante preservar el orden. Por ejemplo, como se describió anteriormente `DrawGeometry` puede ser suministrado con un `Brush` nulo o `Pen` nulo. Si el `PreserveReadbackOrder` es cierto, esta instrucción necesita ser mantenida en el estado de 1 Visual. Si el `PreserveReadbackOrder` es falso, la implementación es libre de descartar esa instrucción.

Nótese que existen numerosas ventajas de esta aproximación de empuje sobre otras aproximaciones de tipo seguro, que

536

740

incluyen que no existe necesidad de conjunto paralelo de tipos reflejar la salida de la enumeración, y que no es necesariamente un requisito para sitio de acumulo de memoria en la interfase de llamada de regreso. Además, los métodos pueden ser llamados directamente sin crear objetos que pasen de regreso, la interfase `DrawingContext` ya esta presente, y no es una API adicional necesaria en el Visual mismo para permitir caminar.

Modificar el contenido de visual es otra consideración. Una forma de expresar modificaciones al `VisualContent` es utilizar recursos (`Pens`, `Brushes`, etc.), los cuales, como se describieron aquí son subclases de `Changeable`, con `StatusOfNextUse=UseStatus.ChageableReference`. este permitirá que las referencias sean mantenidas por medio de la aplicación a los datos que están en estructuras manejadas enviadas en el `DrawingContext`. Este representa una forma uniforme de permitir que los cambios sean hechos, y,, porque en razón a que estos objetos están en un estado conocido ellos se han ajustado explícitamente, en la implementación conocida que cuyos objetos van a ser probablemente modificados. Nótese que este no permite, por ejemplo, cambios en el orden de instrucciones, o adición o supresiones de instrucciones (aunque `RenderAppend()` esta allí para adiciones).

53.8

711

DIBUJO

La clase Drawing contiene una colección de comandos Drawing. Este es exactamente equivalente al contenido almacenado en el DrawingVisual, y es construido por medio de un DrawingContext. El Drawing no tiene afinidad de contexto cuando es inmutable, y así este y la clase relacionada DrawingBrush puede ser utilizada a través de los contextos y en láminas de propiedad por defecto cuando este mismo es inmutable.

Este no soporta directamente la jerarquía en la que este no suministra medios de iterar hijos o encontrar un padre, pero, por vía del DrawingContext, un Drawing puede ser dibujado en otro Drawing. Un Drawing puede ser dibujado en un DrawingContext por vía del DrawDrawing(Drawing), y puede ser utilizado como descripción del contenido para un DrawingBrush.

El Drawing es completamente animable, y soporta retrolectura/iteración de la misma manera que el RetainedVisual.

5 38

712

```

public class System.Windows.Media.DrawingCollection: Changeable
{
    // Constructors
    public DrawingCollection();

    public new DrawingCollection Copy();

    // Properties
    public Rect Bounds { get; }

    // Open/Close operations
    public DrawingContext Open();
    public DrawingContext Append();
}

```

EL PATRÓN CAMBIABLE

Para propósitos de explicación, la presente invención primariamente se va descrita en el contexto de ambiente de programación en el cual los objetos de ejemplo en la escena de gráficos son contruidos, utilizados y modificados. Sin embargo, como se entenderá, aunque la presente invención suministra beneficios significativos en los ambientes de programación relacionados con gráficos, la presente invención está limitada a los ambientes de programación relacionados con gráficos, pero más generalmente aplica a muchos otros tipos de ambiente de programación.

537-

713

En una implementación, la presente invención suministra un conjunto simple de tipos que derivan de una clase base común, por ejemplo, System.Windows.Changeable. Cualquier clase puede ser mutable, al derivar de la clase Changeable y obtener de esta manera las semánticas de tipo valor que el Changeable ofrece. Por ejemplo, una programación de gráficos, el modelo objeto incluye Brushes, Pens, Geometries, FloatAnimations, GradientStops, Segments, y así sucesivamente, como se escribe de manera general en la solicitud de patente estadounidense anteriormente mencionada serie No. 10/402,268. Por ejemplo, la jerarquía para una brocha de dibujo puede ser algo como:

Object: Changeable: Animatable: Brush: TileBrush:
DrawingBrush.

Para propósitos de uso básico, un objeto cambiabile incluye la siguiente propiedad y métodos:

```
public class System.Windows.Changeable
{
    public bool IsChangeable { get; } // defaults to true
    public Changeable Copy();
    public void MakeUnchangeable();
}
```

8 uio

714

La propiedad IsChangeable especifica si el objeto cambiabile puede ser modificado o no, dependiendo de su valor normal. Por ejemplo, un intento para ajusta la propiedad de opacidad de una brocha solamente sucederá si ese objeto brocha tiene una propiedad IsChangeable igual a cierto. De otra manera, se generará una excepción. Cuando se construyen, los objetos cambiables tienen la propiedad IsChangeables igual a cierta por defecto, y son así inmediatamente modificables.

Como se representó en la figura 14, las solicitudes 1402 son recibidas, tal como por vía de las llamadas de función originadas de un programa de aplicación, que están dirigidas a clases cambiables 1404. En general, un manejador de solicitud 1406, que incluye una máquina de estado 1408, procesa las solicitudes y mantiene el estado y los datos del objeto por vía de una estructura de datos de soporte 1410, clonando la estructura de datos en una copia clonada 1412 con un estado de propietario apropiado, basado en un estado de propietario habitual, como se describe adelante. Las excepciones 1414 pueden ser lanzadas, tales como cuando la solicitud busca una transición que no es permitida desde dentro de un estado de propiedad normal. Los estados de propiedad son descritos adelante con referencia a las figuras 15-17.

8-4

Se debe notar que las llamadas de función dirigidas a las clases cambiables pueden ser manejadas directa o indirectamente. Por ejemplo, el manejador de solicitud 1406 en la figura 14 puede influir un conjunto API que suministra las interfases a la máquina de estado. Alternativamente, el manejador de solicitud 1406 puede comprender código medio que convierte las solicitudes recibidas en un sistema operativo llamadas API manejadas por otro sistema operativo. Así, como se utilizan aquí, las solicitudes, por vía del manejador de solicitud, "origina", que ocurra el comportamiento solicitado, sin importar de donde ocurre el procesamiento de la máquina de estado del momento o donde las estructuras de datos y clases son suministradas.

De esta manera, (entre otros mecanismos), las aplicaciones pueden construir objetos cambiables por vía de una solicitud "nueva", ajustar valores en ellos, utilizarlos, continuar ajustando valores en ellos, y continuar utilizándolos.

El siguiente es un ejemplo de cómo una aplicación crea una brocha de un color sólido (scb), modifica la brocha para tener un color sólido (rojo) y utiliza la brocha para colorear un rojo de respaldo de botón:

7/6

```
SolidColorBrush scb = new SolidColorBrush();  
scb.Color = Colors.Red;  
Button1.Background = scb; // this qualifies as a "use" (as described below)
```

La noción de "utilizar" un valor tiene un significado específico, esto es, los valores no son solamente considerados para ser utilizados bajo ciertas condiciones. Estas condiciones incluyen cuando un valor es ajustado en una propiedad PropertySystem, cuando un valor es utilizado en un sub objeto en un cambiabile más complejo, y cuando un valor es utilizado en un comando DrawingContext o similar. Nótese que los entendedores del sistema pueden fácilmente definir otras instancias de "utilización" de un objeto cambiabile que califica como un "uso" y modifica el estado cambiabile del objeto.

Cuando un valor es utilizado en una de estas clases calificadas de usos, desde la perspectiva del modelo de usuario, se hace un clon, y ese clon tiene una propiedad IsChangeable ajustada a falso. Nótese que en la actualidad, un clon no es necesariamente creado, y cuando se crea uno, no es necesariamente profundo (dentro de la jerarquía de objeto, como se escribe adelante).

A pesar de esto, desde la perspectiva de un modelo, es apropiado considerar que un clon es hecho, y así como se utiliza aquí, el concepto de un "clon" cubre un clon que es

5 a 3

717

de hecho creado, un clon que es creado en parte, y/o un clon que es lógicamente creado desde la perspectiva de modelo aunque no necesariamente creado. El clon es el que de hecho consigue ser utilizado, y, por defecto, el clon no puede ser modificado.

Como se mostró anteriormente, el objeto cambiabile también contiene métodos que incluyen `Copy()` y `MakeUnchangeable()`. Una llamada explícita al método `Copy()` crea una copia del cambiabile con la propiedad `IsChangeable` de la copia ajustada a cierto. Este llamado no altera el objeto sobre el cual es llamado el método. El método `MakeUnchangeable()` puede ser llamado sobre cualquier cambiabile, y modifica la propiedad `IsChangeable` para hacerla falsa, (a menos que ya sea falsa, en cuyo caso la llamada no tiene efecto).

Los mecanismos anteriores facilitan el patrón para reemplazar una propiedad. Para cambiar un objeto cambiabile, el valor de la propiedad `IsChangeable` necesita ser cierto. En razón a que un uso calificado de un objeto crea un clon que no es cambiabile, el objeto necesita ser copiado por vía del método `Copy()`, cambiado, y utilizado de nuevo. Este es efectivamente reemplazar el objeto inicial que estaba presente con el nuevo objeto que es una copia modificada del original. Los ejemplos de este se establecen adelante. Note que un cambiabile en uso es aquel que ha sido utilizado, y así mediante la definición

5-5-11

anterior, no es cambiable porque la propiedad `IsChangeable` se establece como falso luego de uso. Así, el cambiable no es modificado, pero en su lugar un cambiable es copiado y reemplazado. Note que con los cambiables existe solamente un conjunto simple de tipos, que está generalmente lejos, más deseable de una perspectiva de programación. Adicionalmente, la mutabilidad cierta se suministra mediante propiedades adicionales, como se describe adelante.

Como se describió anteriormente, es directo crear una brocha, modificarla y utilizarla. Un ejemplo de un uso simple en una operación de dibujo se establece adelante:

```
SolidColorBrush scb = new SolidColorBrush();  
scb.Color = Colors.Red;  
ctx.DrawRectangle(scb, ...); // this is a "use"  
scb.Color = Colors.Green;  
ctx.DrawRectangle(scb, ...); // this is a "use"
```

Ejecutar las instrucciones anteriores dibuja un rectángulo rojo, y un rectángulo verde. Note que `'scb'` es, en efecto, clonado en cada uso de éste.

Una construcción más compleja que usa una brocha de gradiente lineal (`lgb`), en la cual los colores varían (por ejemplo, linealmente) de una parada a otra, se establece adelante:

719

```
LinearGradientBrush lgb = new LinearGradientBrush();  
GradientStop gs1 = new GradientStop(Colors.Red, 0.3);  
GradientStop gs2 = new GradientStop(Colors.Blue, 0.2);  
lgb.Stops.Add(gs1); // this is a "use" of gs1  
lgb.Stops.Add(gs2); // this is a "use" of gs2  
Button2.Background = lgb; // this is a "use" of lgb
```

Aquí, el proceso es construir valores (GradientStops) y utilizarlos en definiciones de valores más complejos.

Considerar otro ejemplo dirigido a cambiar la opacidad (que puede variar de cero a uno) de un trasfondo de botón (Btn) a 0.4. En este particular uso, el antecedente es copiado en un objeto con un propiedad IsChangeable establecida a un valor cierto, el trasfondo es modificado y establecido de regreso.

```
Brush b = Btn.Background.Copy(); // get IsChangeable copy  
b.Opacity = 0.4;  
Btn.Background = b; // "use" modified value
```

Note también que las asignaciones a "Btn.Background" en la última línea aparta cualquier heredado o valor de lámina de propiedad que podría haber entrado.

846

700

Las modificaciones que son más profundas dentro de la jerarquía de objeto parecen no diferentes al usuario que las modificaciones que son superficiales:

```
Brush b = Btn2.Background.Copy();  
  
LinearGradientBrush lgb = (LinearGradientBrush)b;  
lgb.Stops[1].Color = Colors.Green;  
lgb.Stops[0].Location = 0.2;  
  
Btn2.Background = b;
```

Note que Copy() solo necesita ser llamado sobre el objeto de nivel superior y no sobre el GradientStops individual. Esto es porque el sistema tiene cuidado de asegurar que los sub-objetos de un objeto con una propiedad IsChangeable cierta son ellos mismos ajustados con IsChangeable igual a cierto cuando ellos son accesados.

La FIG. 15 muestra un diagrama de estado que representa los estados de un objeto cambiabile en uso básico, iniciando con la propiedad IsChangeable igual a cierto cuando está recientemente creada. En general, las flechas sólidas muestran los estados de los objetos que hacen transición desde un estado habitual a un estado objetivo, aunque cualquier flecha punteada representa una operación que deja

500

721

transformación de escala se utiliza está afectará todo lo que es dibujado, aun si se especifica en temas físicos. El valor de $1/96^{\circ}$ de una pulgada fue escogido de tal forma que una unidad de usuario por defecto es equivalente a un píxel sobre los despliegues contemporáneos con configuraciones por defecto. Nótese que se puede suministrar un mecanismo para suministrar indicaciones al sistema de diseño y otro código de usuario de tal forma que ellos puedan ser optimizados para la resolución de salida del dispositivo a los cuales ellos están representando.

La composición de alto nivel y el motor de animación 214 en conjunto con el código de programa 202 construyen un gráfico de escena para representar una escena de gráficos suministrada por el código de programa 202. Por ejemplo, cada ítem a ser dibujado puede ser cargado con instrucciones de dibujo, el cual el sistema puede cachear en estructuras de datos de gráfico de 100 a 216. Como se describirá adelante, existe un número de formas varias para especificar esta estructura de datos 216, y que es dibujada. Adicionalmente, la composición de alto nivel y el motor de animación 214 se integra con los sistemas de temporización de animación 220 para proveer control de animación declarativo (u otro) (por ejemplo intervalos de animación) y control de tiempo. Nótese que el sistema de animación le permite animar valores para ser pasados esencialmente a donde sea en el sistema,

480

722

incluyendo, por ejemplo, en el nivel de propiedad del elemento 208, dentro de la capa API visual 212, y en cualquiera de los otros recursos. El sistema de tiempo es expuesto a los niveles de elemento y visual.

El motor de composición de animación de bajo nivel 218 maneja la composición, animación y representación de la escena, la cual es entonces suministrada al subsistema de gráficos 222. El motor de bajo nivel 218 compone las representaciones para las escenas de múltiples hebras (por ejemplo de una o más aplicaciones), y con componentes de representación, implementa la representación actual de gráficos a la pantalla. Nótese, sin embargo, que en momentos puede ser necesario y/o ventajoso para alguna de las representaciones ocurrir a mayores niveles. Por ejemplo, aunque el servicio de capas inferior solicita de múltiples hebras, las capas mayores son instanciadas sobre una base por hebra por medio de la cual es posible por vía de mecanismos de imagen 204 desarrollar consumo de tiempo o específico de hebra representando a mayores niveles, y pasar referencias a un mapa de bits a capas inferiores.

El MIL 200 suministra un número de conceptos que son integrados para suministrar mejoras en gráficos y programación audiovisual, tal como un conjunto de recursos y clases que son compartidas a través de esta pila de capas

481

723

completa, incluyendo Pens, Brushes, Geometry, Transforms y Effects. También, son suministrados tipos de primitivos simples, incluyendo Points, Rectangles, y así sucesivamente. El Pen y el Brush son tipos complejos que describen como afecta la representación en estos varios niveles. También se suministra un tipo especial de Brush denominado como VisualBrush, que les permite a los programadores utilizar un "meta archivo" de gráficos arbitrario para llenar un área (por vía de un VisualBrush explícitamente o un Pen que referencia un VisualBrush). En razón a que esta es una forma comprimida para almacenar y utilizar tráficos arbitrarios, este juega un papel en un recurso de gráficos. Existe un perfil específico de la sintaxis de marcadores de gráficos de vector utilizada para crear estos objetos directamente. Aunque generalmente similar al VisualBrush, el DrawingBrush es más comprimido y digerido, y es esencialmente un meta archivo mientras el VisualBrush es una brocha de gráfico de escena.

Otras clases incluyen Geometry, que es un tipo complejo que es utilizado para definir el área para llenar, delinear o cortar. El Transform es otra jerarquía de tipo complejo para definir como transformar espacios ordenados. Los efectos describen un sistema para poner efectos de filtro arbitrarios sobre una sección de contenido, por ejemplo, un achurado. Nótese que este también incluye un modelo de extensibilidad

482.

724

de programa complementario. Un Visual API es suministrado el cual utiliza los tipos, anteriores para conseguir bits a la pantalla u otros objetivos. Este incluye un montaje de nivel base para el resto del sistema (por vía de un hWnd u otro mecanismo) junto con estructura de datos de posicionamiento de pantalla inducida anteriormente. Imaging le permite al programador conseguir imágenes en y fuera del sistema basado en MIL. Los Media suministra la capacidad para utilizar otras formas de medios, incluyendo audio y video. En general, el Visual API se refiere al conjunto de API que operan por debajo del sistema de elemento y el sistema de diseño, y se refiere a programar con y a Visuals directamente, en lugar de a un nivel mayor. Nótese que un Visual es un objeto básico en la estructura de datos de cache 216, la cual comprende una estructura de datos retenida para cosas visuales sobre la pantalla, y también cachear la Instruction List y los recursos específicos del dispositivo por razones de desempeño.

Las figuras 3 y 4 muestran gráficos de escena de ejemplo 300 y 400, respectivamente, incluyendo un objeto base denominado como un visual. En general, un Visual comprende un objeto que representa una superficie virtual al usuario y tiene una representación Visual sobre la pantalla. Como se representa en la figura 5, la clase base Visual suministra una funcionalidad base para otros tipos de Visual, esto es, la

483

725

clase Visual es una clase base abstracta de la cual se derivan los tipos Visual.

Como se representa en la figura 3, un Visual de alto nivel (o raíz) 302 está conectado a un objeto manejador Visual 304, que también tiene una relación (por ejemplo, por vía de un manejo) con una ventana (HWnd) 306 o una unidad similar en la cual los datos de gráfico son sacados por el código de programa. El VisualManager 304 maneja el dibujo del Visual de alto nivel (en cualquier hijo de ese Visual) a esa ventana 306. La figura 6 muestra el VisualManager como uno de un conjunto de otros objetos 620 en el modelo objeto del sistema de gráficos descritos aquí.

Para dibujar, el VisualManager 304 procesa (por ejemplo atraviesa o transmite) la gráfica de escena como se programo por un despachador 308, y suministra instrucciones de gráficos y otros datos al componente de bajo nivel 218 (figura 2) para su ventana correspondiente 306, tal como se describe de manera general en las solicitudes de patente U.S. anteriormente mencionadas. El procesamiento de gráfico de escena será ordinariamente programado por el despachador 308 con un índice que es relativamente más lento que el índice de refresco del componente de más bajo nivel 218 y/o el subsistema de gráfico 222. La figura 3 muestra un número de Visuals hijos 310-314 dispuestos gráficamente por debajo del

784

76

Visual de alto nivel (raíz) 302, algunos de los cuales son representados como habiendo sido poblados por vías de contextos de dibujo 316, 317 (mostrados como cajas punteadas para representar su naturaleza temporal) con listas de instrucción asociadas 318 y 319, respectivamente; por ejemplo, conteniendo Instruction List y otros Visuals. Los Visuals pueden también contener otra información de propietario. En general, la mayoría de acceso sobre la clase visual básica viene por vía de una interfase IVisual, y el Visual se deriva del DependencyObject, como se representa en la figura 5. Los Visuals (adicionalmente descritos en el Appendix) también pueden contener otra información de propietario como se muestra en el siguiente ejemplo de clase Visual:

465

727

```

public class Visual : IVisual, DependencyObject
{
    protected Visual();
    protected Visual(UIContext context);

    bool IVisual.HasChildren { get; }
    VisualCollection IVisual.Children { get; }

    Visual IVisual.Parent { get; }

    Geometry IVisual.Clip { get; set; }
    Bool IVisual.Show { get; set; }
    double IVisual.Opacity { get; set; }
    BlendMode IVisual.BlendMode {get; set;}

    Matrix IVisual.TransformToAncestor(Visual ancestor);
    Matrix IVisual.TransformToDescendant(Visual descendant);
    Matrix IVisual.TransformFromAncestor(Visual ancestor);
    Matrix IVisual.TransformFromDescendant(Visual descendant);
    Matrix IVisual.TransformToVisual(Visual visual);
    Matrix IVisual.TransformFromVisual(Visual visual);

    bool IVisual.IsAncestorOf(Visual descendant);
    bool IVisual.IsDescendantOf(Visual ancestor);

    Visual IVisual.FindCommonVisualAncestor(Visual otherVisual);

    PointHitTestResult IVisual.HitTest(Point point);
    void IVisual.HitTest(
        HitTestFilterDelegate filterHitDelegate,
        HitTestResultDelegate resultHitDelegate,
        HitTestParameters hitTestParams);
}

```

406

70A

```

Rect IVisual.VisualContentBounds { get; }
Rect IVisual.VisualDescendantBounds { get; }

protected virtual PointHitTestResult HitTestCore(
    PointHitTestParameters point);
protected virtual GeometryHitTestResult HitTestCore(
    GeometryHitTestParameters geometry);

protected virtual Rect HitTestBounds { get; }
}

```

Un Visual es un contenedor para contenido gráfico y un conjunto de hijos. Varias propiedades sobre un visual se pueden utilizar para controlar el comportamiento de representación de un Visual. Por ejemplo, al configurar un corte sobre un Visual el contenido es cortado con la forma especificada. Otras propiedades son la transformación, el modo de mezcla, la opacidad, la muestra, etc. Todas aquellas propiedades se pueden controlar por vía de las propiedades de consecución y ajuste.

Una propiedad Show es utilizada para mostrar/ocultar el visual, por ejemplo, cuando falsear el visual es invisible, de otra manera el visual es visible. Adicionalmente, los objetos MIL (donde los Visuals en la capa de Visual API o los elementos en la capa de elementos existen en una jerarquía.

487

729

Un sistema de coordenadas es heredado hacia abajo a través de esta jerarquía. De esta manera, un padre puede poner una transformación de coordenadas que modifica el paso y consecución representados aplicados a ese hijo de padre.

La transformación para un visual está sobre la conexión a ese visual. En otras palabras, está se ajusta por vía del [Get|Set]ChildTransform sobre la propiedad VisualCollection Children del padre. Ver también el VisualCollection descrito posteriormente.

Nótese que las transformaciones coordinadas pueden ser aplicadas de una manera uniforme a todo, como si este fuera un mapa de bits. Nótese que esto no significa que las transformaciones siempre aplicaran al mapa de bits, sino que lo que se consigue representado se afecta por transformaciones igualmente. Por vía de ejemplo, si el usuario dibuja un círculo con un lápiz redondo que es de una pulgada de ancho entonces se aplica una escala en la dirección X de 2 a ese círculo, el lápiz tendrá dos pulgadas de ancho a la izquierda y derecha y solamente una pulgada de ancho de arriba abajo. Esto algunas se denomina como una composición o transformación del mapa de bits (opuesto al esqueleto o escala de geometría que afectan solamente a la geometría). La figura 8 es una representación de la

x 64

transformación de escalamiento, con una imagen no transformada 800 que aparece sobre la izquierda y una imagen transformada 802 con una escala no uniforme que aparece a la derecha. La figura 9 es una representación de la transformación de escalamiento, con la imagen no transformada 800 que aparece a la izquierda y una imagen transformada 904 con geometría de escalamiento que aparece a la derecha.

Con respecto a la transformación coordenada de un visual, el TransformDescendant regresa una transformación que refleja el cambio de espacio coordenado que va desde el visual de referencia al visual descendente. La transformación puede entonces ser utilizada para por ejemplo transformar un punto desde el espacio coordenado del visual de referencia al espacio coordenado del visual descendiente. El TransforFromDescendant regresa similarmente una transformación que describe el cambio de espacio que va desde el Visual descendente al Visual de referencia. La transformación puede ser entonces utilizada para transformar un punto en el espacio coordenado del visual descendiente al espacio coordenado del visual de referencia. Por conveniencia el Visual API también suministra el TransformToAncestor, TransformFromAncestor, TransformFromVisual, y TransformToVisual que también regresan transformaciones a los cambios de espacio coordenados respectivos. Nótese que en los dos API últimos, la relación entre los Visual no se

especifica. Ellos pueden ser aun homólogos en el árbol visual en tanto que ellas comparten unos ancestros comunes. La implementación encontrará el ancestro común y luego calculará la transformación de coordenada desde el Visual de referencia al ancestro común, y luego desde el ancestro común al visual objetivo. La transformación resultante puede ser utilizada para por ejemplo transformar un punto entre los Visual especificados.

Para conseguir las propiedades que están disponibles que se pueden utilizar para determinar la caja de unión del contenido del Visual, a saber el VisualDescendantBounds, que es la caja de unión de todo el contenido gráfico de los descendientes, y el VisualContentBounds que es la unión del contenido. Aplicar una Union a estos suministra las uniones totales del Visual.

Una propiedad de corte ajusta (y consigue) la región de corte de un Visual. Cualquier Geometry (la clase de geometría es mostrada en la figura 10 y se describe en la sección Geometry adelante) se puede utilizar como una región de corte. En una implantación, un conjunto por defecto para la región de corte es nula, es decir, sin corte, lo cual puede considerarse como un rectángulo de corte infinito desde $(-\alpha, -\alpha)$ a $(+\alpha, +\alpha)$.

Las propiedades de opacidad consiguen/ajustan el valor de opacidad del visual, de tal forma que el contenido del visual es mezclado con la superficie del dibujo con base en el valor de opacidad y el modo de mezcla seleccionado. La propiedad BlendMode se puede utilizar para ajustar (o conseguir) el modo de mezcla que es utilizado. Por ejemplo, un valor de opacidad (alfa) se puede ajustar entre 0,0 y 1,0, con el conjunto de mezcla alfa lineal como el modo, por ejemplo, $\text{Color} = \text{alfa} * \text{color delantero} + (1,0 - \text{alfa}) * \text{color de fondo}$. Otros servicios, tales como las propiedades de efectos especiales, se pueden incluir en el Visual, por ejemplo, achurados o borrosos, monocromos, y así sucesivamente.

El Visual tiene también una propiedad Children para manejar el conjunto de hijos. Está también suministra una propiedad HasChildren para revisar si el Visual tiene cualquier hijo después de todo. La propiedad children regresa un VisualCollection que le permite al usuario desarrollar operaciones como agregar, remover, insertar, etc. sobre el conjunto de hijos. Los siguiente muestra un ejemplo del VisualCollection:

733

```

public class Visual : ICollection, IEnumerable
{
    public Int32 Add(Visual visual)
    public Void Clear()
    public Boolean Contains(Visual visual)
    public virtual sealed Void CopyTo(Array array, Int32 index)
    public Int32 get_Capacity()
    public virtual sealed Int32 get_Count()
    public virtual sealed Boolean get_IsSynchronized()
    public Visual get_Item(Int32 index)
    public virtual sealed Object get_SyncRoot()
    public virtual sealed IEnumerator GetEnumerator()
    public Transform GetTransform(Int32 index)
    public Transform GetTransform(Visual child)
    public Int32 IndexOf(Visual visual)
    public Void Insert(Int32 index, Visual visual)
    public Void Remove(Visual visual)
    public Void RemoveAt(Int32 index)
    public Void RemoveRange(Int32 index, Int32 count)
    public Void set_Capacity(Int32 value)
    public Void set_Item(Int32 index, Visual value)
    public Void SetTransform(Int32 index, Transform transform)
    public Void SetTransform(Visual child, Transform transform)
    public Void TrimToSize()
}

```

El orden de los Visuals el VisualCollection determina en que orden son representados los Visuals, es decir, los Visuals son representados desde el índice más bajo al índice más alto

497

desde la parte posterior hacia la parte frontal (orden pintado).

Un ProxyVisual es un visual que se puede agregar más de una vez en el gráfico de escena, por ejemplo, por debajo de un Visual de contenedor. En razón a que ningún Visual referido por el ProxyVisual puede ser alcanzado por múltiples sendas desde la raíz, los servicios de lectura (TransformToDescendent, TransformFromDescendent y HitTest) no trabajan a través de un ProxyVisuals. En esencia, existe una senda canónica desde cualquier Visual a la raíz del árbol visual y esa senda no incluye ningunos ProxyVisuals.

La figura 4 muestra una gráfica de ejemplo 400 en la cual los Container Visual y los DrawingVisuals están relacionados en el gráfico de escena, y tienen datos asociados en la forma de de Instruction Lis, (por ejemplo, en contextos de dibujos correspondientes). El Container Visual es un Visual que tiene solamente contenido estructural que se deriva de la clase base Visual. Los Visuals pueden ser arbitrariamente alojados uno en el otro. En particular es legal alojar Container Visuals. El propósito principal del Container Visual es suministrar un contenedor para Visuals que pueda ser convenientemente accesado sin ir a través de la interfase IVisual. El Container Visual por lo tanto reimplementa todos los métodos IVisual como métodos públicos de nuevo. Los hijos

del container Visual para ser manipulados con los métodos de la propiedad del VisualCollection Children del Container Visual.

Regresando a la figura 5, todavía aún otro visual es un HwndVisual 505, que posicionan un Win32 hijo Hwnd en el gráfico de escena. Más particularmente, los programas de legado operaran todavía por vía del método WM_PAINT (o similar) que dibuja un Hwnd hijo (o similar) con base en tecnología de gráficos anteriores. Para soportar tales programas en el nuevo modelo de procesamiento de gráficos, el Hwnd Visual le permite al Hwnd estar contenido en un gráfico de escena y ser movido en la medida que el Visual padre es reposicionado. Otros tipos de visuales 506 también son factibles, tales como los visuales tridimensionales (3D) que posibilitan una conexión entre mundos bidimensionales y tridimensionales, por ejemplo, una vista similar a cámara es posible por vía de un visual bidimensional que tiene una vista en un mundo tridimensional. Tal visual 3D se describe adelante.

Como se describió anteriormente, los visuales pueden se dirigidos a poblar sus contextos de dibujo con baños primitivos de dibujo, incluyendo Geometry, ImageSource y MediaData. Adicionalmente existe un conjunto de recursos y clases que son compartidos a través de está pila completa.

Esto incluye Pens, Brushes, Geometry, Transforms y Effects. La clase abstracta `DrawingContext` expone un conjunto de dibujos y operaciones de estado de contexto que se pueden utilizar para poblar un `DrawingVisual`, `RetainedVisual`, `ImageData`, etc. En otras palabras, la clase abstracta de la clase abstracta de contexto de dibujo expone un conjunto de operaciones de dibujo y operaciones de introducción/extracción; para cada operación de dibujo e introducción existe dos métodos que toma constantes argumentos, y uno que toma animadores como argumentos. Ejemplos de las operaciones de empuje/extracción son los `PushTransform`, `PopTransform`, `PushClip`, `PopClip`, `PushOpacity`, `PopOpacity`, etc.

Los varios servicios (incluyendo transformación, opacidad, y corte) pueden ser introducidos y extraídos en un contexto de dibujo, y las operaciones de introducción/extracción pueden ser alojadas, en tanto exista una llamada de extracción apropiada para cada llamada de empuje.

El método `PushTransform` introduce una transformación. Las operaciones de dibujo subsecuentes son ejecutadas con respecto a la transformación introducida. La llamada de extracción extrae la transformación introducida por la llamada coincidente `PushTransform`:

```
Void PushTransform (Transform transform);  
Void PushTransform (Matriz matriz);  
Void Pop ();
```

Similarmente, el método PushOpacity introduce un valor de opacidad. Las operaciones de dibujo subsecuentes son desarrolladas sobre una superficie temporal con un valor de opacidad especificado y entonces se compone en la escena. El Pop () extrae la opacidad introducida por la llamada PushOpacity coincidente:

```
Void PushOpacity (float opacity);  
Void PushOpacity (floatAnimation opacity);  
Void Pop ();
```

El método PushClip introduce una geometría de corte. Las operaciones subsecuentes de dibujo son cortadas a la geometría. El corte es aplicado en espacio de post transformación. El Pop () extrae la región cortada introducida por el llamado de PushClip coincidente:

```
Void PushClip (Geometry clip);  
Void Pop ();
```

Nótese que las operaciones de extracción pueden ser arbitrariamente alojadas en tanto que las operaciones de

extracción son coincidentes con una introducción. Por ejemplo, lo siguiente es válido.

```
PushTransform(...);  
DrawLine(...);  
PushClip(...);  
DrawLine(...);  
Pop();  
PushTransform(...);  
DrawRect(...);  
Pop();  
Pop();
```

Geometry es un tipo de clase (figura 10) que define un esqueleto de gráficos de vector, sin trazo o el llenado. Cada objeto de geometría es una forma simple (LineGeometry, EllipseGeometry, RectangleGeometry), una forma simple compleja (PathGeometry) o una lista de tales formas GeometryCollection con una operación de combinación (por ejemplo, unión, intersección, y así sucesivamente) especificadas. Estos objetos forman una clase jerárquica como se representa en la figura 10.

Como se representa en la figura 11, el PathGeometry es una colección de objetos Figure. A su vez, cada uno de los objetos Figure está compuesto de uno o más objetos Segment que definen de hecho la forma de la figura. Una figura es una

subsección de un Geometry que define una colección de segmentos. Esta colección de segmentos es una serie simple conectada de objetos Segment bidimensionales. El Figure puede ser una forma cerrada con un área definida, o solo una serie conectada de Segments que definen una curva pero no un área encerrada.

Como se representó en la figura 12, cuando la geometría (por ejemplo un rectángulo) es dibujado, una brocha o lápiz se puede especificar, como se escribe adelante. Adicionalmente, el objeto lápiz también tiene un objeto brocha. El objeto brocha define como llenar gráficamente un plano, y existe una clase jerárquica de objetos brocha. Este se representa en la figura 12 por el rectángulo llenado 1202 que resulta cuando el visual que incluye el rectángulo y las instrucciones de brocha y los parámetros es procesado. Un objeto Pen se mantiene sobre un Brush junto con las propiedades para Thickness, LineJoin, LineCap, EndCap, MiterLimit, DashArray y DashOffset, como se describe adelante. También como se describe adelante, algunos tipos de Brushes (tales como gradientes y nueve rejillas) se ajustan en tamaño a sí mismas. Cuando se utiliza, el tamaño para estas brochas se obtiene de la caja de unión, por ejemplo, cuando el GradientUnits/DestinationUnits para el Brush se ajusta a RelativeToBoundingBox, la caja de unión del primitivo que está siendo dibujada se utiliza. Si aquellas propiedades son

ajustadas a Absolute, entonces el espacio coordenado se utiliza.

Como se mencionó anteriormente y como se describe adicionalmente adelante, el modelo de objeto de gráfico de la presente invención un modelo objeto Brush, y es generalmente dirigido hacia el concepto de cubrir un plano con píxeles. Ejemplos de tipos de brochas están representadas en la jerarquía de la figura 13, y, bajo una clase base Brush, incluir GradientBrush, NineGridBrush, SolidColorBrush y TileBrush. El GradientBrush incluye objetos LinearGradient y RadialGradient. El DrawingBrush y el ImageBrush derivan del TileBrush. Las disposiciones alternativas de las clases son factibles, por ejemplo, derivando el TileBrush puede estar el ImageBrush, el VisualBrush, el VideoBrush, NineGridBrush y DrawingBrush. Nótese que los objetos Brush pueden reconocer como ellos se relacionan con el sistema de coordenadas cuando ellos son utilizados, y/o como ellos se relacionan con la caja de unión de la forma en la cual ellos son utilizados. En general, la información tal como el tamaño puede ser inferida del objeto sobre el cual la brocha es dibujada. Más particularmente, muchos de los tipos de brocha utilizan un sistema de coordenadas para especificar algo de sus parámetros. Este sistema de coordenadas puede ser definido como relativo a la caja de unión simple de la forma a la cual la brocha se aplica, o puede ser relativo al espacio de

coordenadas que es activo en el momento en que la brocha se utiliza. Estos son conocidos, respectivamente, como el modo `RelativeToBoundingBox` y el modo `Absolute`.

VISUAL API

El Visual API es un punto de partida para dibujar por vía de la capa de integración media, y comprende múltiples tipos de objetos, incluyendo un objeto `VisualManager`, que conecta el `Visual Tree` a un medio. Los diferentes tipos de `VisualManager` (por ejemplo, `Screen`, `Printer`, `Surface`) manejan los procesos de reproducción de un `Visual Tree` a su medio particular. El `VisualManager` es adicionalmente descrito aquí en la sección titulada "objetos MIL de máximo nivel".

Un visual es donde el usuario hace dibujos. Este es un modo en el árbol visual (un objeto contenedor que es la estructura para una escena, como se describe adelante) y suministra un lugar para un programa para dibujar. Existen varios tipos de visuales cada uno ajustado para diferentes usos. Un visual es similar al lado visual/salida de un Win32 `hWnd`.

Los `Visuals` suministran un número de capacidades, incluyendo acceso visual padre, recolección visual hijo, corte basado en una geometría arbitraria, opacidad, modo de mezcla, transformación que afecta ese visual y su hijo, ensayo de

golpe, servicios de transformación de coordenada, servicios de caja de unión, y efectos (de línea y vector).

Para representar una escena visual, el árbol visual es atravesado, por ejemplo, de arriba hacia abajo, de izquierda hacia derecha, primero representando el contenido y luego atravesando el hijo del Visual de izquierda a derecha. Cualquiera de los hijos del Visual son dibujados antes del contenido de ese Visual mismo. Si el contenido requiere una devolución de llamada al usuario, esto ocurre sincrónicamente durante el tiempo de representación. Aquí está algún pseudocódigo para lo que el sistema hace (por vía de un VisualRenderer) para representar un árbol Visual a un dispositivo:

```
Visual.RenderToDevice(device)
{
    this.RenderContents(device);
    foreach (Visual child in this.Children)
    {
        Child.Walk(ctx);
    }
}
```

Un visual derivado puede llamar de nuevo al usuario durante la llamada RenderContentst.

La figura 13 representa un Visual Class Hierarchy en una implementación. Durante las llamadas de regreso que ocurren como parte del paso de representación (que comprende una llamada de regreso al `IRetainedVisual.Render` o una llamada de regreso al `PaintingVisual.RenderCore`) el árbol Visual es "asegurado" por razones de desempeño. Este aseguramiento ocurre en una base de contexto amplia, significando que no se puede modificar el contexto del árbol cuando se asegura, sin importar a cual árbol visual pertenece un visual. Cuando el árbol es asegurado, el hijo de un Visual no puede ser cambiado, y otros contenidos del visual no pueden ser cambiados de ninguna manera (por ejemplo `Open`, ajuste del modelo 3D de raíz, y así sucesivamente), `Transform`, `Opacity`, `Clip`, `BlendMode` o `Effect` sobre un Visual no pueden ser ajustados, ensayados por golpe, ni operará un trabajo de operación de información de unión `Get`.

Las capacidades sobre un Visual son expuestas por vía de la interfase `IVisual`, elaborando de esta manera las capacidades públicas, aunque protegiendo el modelo del objeto. La siguiente es la interfase `IVisual` en una implementación de ejemplo:

```

public interface IVisual
{
    bool HasChildren { get; }
    VisualCollection Children { get; }

    Visual Parent { get; }
    Geometry Clip { get; set; }
    bool Show { get; set; }
    double Opacity { get; set; }
    Matrix TransformToAncestor(Visual ancestor);
    Matrix TransformToDescendant(Visual descendant);
    Matrix TransformFromAncestor(Visual ancestor);
    Matrix TransformFromDescendant(Visual descendant);
    Matrix TransformToVisual(Visual visual);
    Matrix TransformFromVisual(Visual visual);
    bool IsAncestorOf(Visual descendant);
    bool IsDescendantOf(Visual ancestor);
    Visual FindCommonVisualAncestor(Visual otherVisual);
    PointHitTestResult HitTest(Point point);
    void HitTest(
        HitTestFilterDelegate filterHitDelegate,
        HitTestResultDelegates resultHitDelegate,
        HitTestParameters hitTestParams);
    Rect VisualContentBounds { get; }
    Rect VisualDescendantBounds { get; }
}

```

Un Visual comprende representar el contenido y una colección de hijos. Un número de propiedades, incluyendo transformación, corte, opacidad, mezcla de modo, y así sucesivamente pueden ser utilizados para controlar la

745

representación actual del árbol Visual. Nótese que el Visual no tiene que tener contenido e hijos al mismo tiempo. En una implementación, la representación del contenido y la colección de hijo pueden ser creadas por demanda para utilizar el uso de la memoria.

Las transformaciones sobre hijos son desarrolladas por vía de una transformación sobre la propiedad Children del tipo VisualCollection.

```
public void SetChildTransform(Visual visual, Transform transform);  
public Transform GetChildTransform(Visual visual);
```

La Transformation Read Services suministra métodos que les permiten a los usuarios obtener una Matriz que representa la transformación agregada de un cuadro de coordenada a otros:

```
Matrix TransformToDescendant(Visual descendant);  
Matrix TransformFromDescendant(Visual descendant);  
Matrix TransformToAncestor(Visual ancestor);  
Matrix TransformFromAncestor(Visual ancestor);  
Matrix TransformToVisual(Visual visual);  
Matrix TransformFromVisual(Visual visual);
```

Los métodos TransformToAncestor y TransformToDescendant son más eficientes pero requieren que el llamador conozca la

746

relación entre los dos Visuals. Los métodos más generales TransformTo/FromVisual encuentran un ancestro común y computan la transformación de ese visual. Nótese que estos pueden originar que los caches sean actualizados y OnRender para ser llamados sobre visuales arbitrarios. Si los visuales no están conectados o se encuentra una transformación degenerada, es lanzada una excepción.

Cálculos de unión también son suministrados:

```
protected Rect VisualDescendantBounds { get; }  
protected Rect VisualContentBounds { get; }
```

El VisualDescendantBounds regresa de las cajas de unión de contenido para los descendientes del visual de ese momento, pero no incluyendo los contenidos o el visual del momento. El VisualContentBounds regresa la caja de unión para los contenidos del Visual del momento.

Una propiedad de opacidad (por ejemplo una doble) especifica un valor de Opacity opcional para aplicarlo a un Visual cuando este es compuesto en su padre. Por defecto, este valor es 1,0, que originará que los contenidos aparezcan con la Opacity completa. Nótese que en razón a que este valor es multiplicado con cualquier otro dato de Opacity en el sub

gráfico, la Opacity de 1,0 no altera nada. La Opacity de 0,0 origina que los contenidos completos sean transparentes, el valor de 0,25 hace que la Opacity sea el 25% de sus valores nominales, y así sucesivamente. La Opacity es aplicada antes del BlendMode.

El BlendMode es una propiedad que especifica un BlendMode opcional para aplicar los contenidos a un sub gráfico y el destino cuando este Visual está compuesto. Por defecto, este valor es BlendMode.Normal, que desarrolla una composición entrada del canal alfa en el destino. Configurando esta propiedad algún otro valor se desarrollará la composición con los contenidos Visual como la fuente de los contenidos del objetivo de representación como el destino. Esto se aplica después de que se aplica la propiedad de opacidad.

La clase Visual base presenta interfases para las características que los Visuals tienen en común:


```

public class System.Windows.Media.Visuals.Visual :
    DependencyObject, IVisual
{
    // Constructors
    protected Visual();
    protected Visual(UIContext Context);

    // Finalizer
    ~Visual();

    protected virtual PointHitTestResult HitTestCore(
        PointHitTestParameters point);
    protected virtual GeometryHitTestResult HitTestCore(
        GeometryHitTestParameters geometry);

    protected virtual Rect HitTestBounds { get; }

```

```

// All methods for IVisual are implemented. These are implemented
// explicitly only for the interface. Here is an example:
bool IVisual.HasChildren { get; }
}

```

El ContainerVisual deriva directamente de la clase visual y promueve las propiedades protegidas para ser públicas. Esto le posibilita al usuario crear contenedor visual sin tener que derivar una nueva clase.

```

public class System.Windows.Media.Visuals.ContainerVisual :
    Visual
{
    // Constructors
    public Visual();
    public Visual(UIContext Context);

    // All IVisual methods are "reexposed" so that
    // they are public. This class is merely a convenience
    // above and beyond Visual
}

```

Un RetainedVisual es un visual que introduce una "corriente de instrucciones retenidas" que se puede utilizar para dibujar:

```

public class System.Windows.Media.Visuals.RetainedVisual : Visual
{
    protected RetainedVisual();
    protected RetainedVisual(UIContext Context);

    // Open/Close semantics
    protected DrawingContext RenderOpen();
    protected DrawingContext RenderAppend();
}

```

```

// Invalidation
protected bool IsVisualInvalid { get; }
protected bool InvalidateVisual();
Rect RenderBounds { set; get; }
}

public class System.Windows.Media.Visuals.IRetainedRender
{
    void Render(DrawingContext drawingContext);
}

```

La corriente de instrucción puede ser utilizada en un modo OnDemand, en el cual el usuario es llamado de nuevo para representar según sea necesario. El usuario es requerido para implementar IRetainedRender. La corriente de instrucción puede ser utilizada en un modo Imperative, en el cual el usuario puede directamente llamar el RenderOpen y conseguir un contexto de dibujo. Generalmente un usuario utilizará uno de estos modos a la vez, pero es posible para ellos ser utilizados de una forma mezclada.

El RenderOpeny el RenderAppend afectará la corriente del momento y estarán disponibles en varios escenarios. Ellos se generarán si este Visual está habitualmente en una llamada de retorno Render. El RenderOpen limpia todo el contenido previo que estaba en el RetainedVisual, aunque el RenderAppend unirá un nuevo contenido al extremo de la corriente. Si el usuario

ha implementado `IRetainedRender` en el `Visual`, entonces las el usuario señala el sistema que el modo `OnDemand` también deberá utilizar. El sistema utilizará el valor establecido en la propiedad `RenderBounds` en la medida en que las uniones para el contenido son suministradas por la llamada del `Render`. El sistema puede decidir optimizar la escena y lanzar el contenido en cualquier momento, cuando se implementa el `IRetainedRender`. El `RenderBounds` actuará por defecto a una recta vacía, aunque el `Rect.Infinite` o un valor no ajustado son alternativas factibles. Para conseguir las ganancias del desempeño de actualización implicadas con la llamada de regreso, el usuario tendrá que establecer un valor razonable.

Cuando se representa la escena, el sistema examinará cada visual en concepto (nótese que en la realidad el sistema será capaz de ignorar la mayoría de los visuales la mayor parte del tiempo). Si ese visual ajustado el `IsVisualInvalid` como cierto, y, con base en el `RenderBounds`, ese contenido de visual será necesario, entonces el sistema llamará al `IRetainedVisual.Render` para llenar en el contenido de ese visual. Esto reemplazará cualquier contenido que ya este allí. El usuario puede llamar manualmente al sistema para lanzar la corriente de contenido al llamar el `Invalidate`.

Si el `IRetainedRender` no es implementado entonces el `IsVisualInvalid` siempre regresará como falso. El `Invalidate`

752

no hará nada. Nótese que el `IRetainedRender` es así llamado. (por ejemplo en lugar de `Prender`), porque no es suficientemente general para ser utilizado en todos los casos de llamada de retorno de representación. Por ejemplo, el `PaintVisual` llama de retorno con un rectángulo inválido.

Un `DrawingVisual` es muy similar a un `RetainedVisual` pero está diseñado para ser utilizado sin derivación. Los métodos protegidos son "promovidos" al público. Adicionalmente, no existe llamada de regreso `Render` o necesidad de implementar la interfase `IRetainedRender`. En razón de esto, el contenido es siempre retenido, similar a cuando el `IRetainedRender` no es implementado sobre un `RetainedVisual`.

571.

753

```
public class System.Windows.Media.Visuals.DrawingVisual :
    ContainerVisual
{
    // Constructors
    public DrawingVisual();
    public DrawingVisual(UIContext Context);

    // Tree interfaces
    public bool HasVisualChildren { get; }
    public Visual VisualParent { get; }
    public bool IsVisualAncestorOf(Visual descendant);
    public bool IsVisualDescendantOf(Visual ancestor);
    public Visual FindCommonVisualAncestor(Visual otherVisual);

    // Standard compositing properties
    public new Geometry VisualClip { get; set; }

    // Default value is 1.0
    public new double VisualOpacity { get; set; }
    public new DoubleAnimationCollection VisualOpacityAnimations { get; set; }
```

512

```

// Default value is BlendModes.Normal
public new BlendMode VisualBlendMode { get; set; }

// Transformations
public Transform TransformToDescendant(Visual descendant);
public Transform TransformFromDescendant(Visual descendant);
public Transform TransformToAncestor(Visual ancestor);
public Transform TransformFromAncestor(Visual ancestor);
public Transform TransformToVisual(Visual visual);
public Transform TransformFromVisual(Visual visual);

// Bounds calculation
public new Rect VisualDescendantBounds { get; }
public new Rect VisualContentBounds { get; }

// Open/Close operations
public new DrawingContext RenderOpen();
public new DrawingContext RenderAppend();
}

```

Un `PaintingVisual` es también suministrado:

```

public class System.Windows.Media.Visuals.PaintingVisual : Visual
{
    protected SurfaceVisual();
    protected SurfaceVisual(UIContext Context);

    // Surface properties
    protected Rect PaintingBounds { get; set; }
    protected Color FillColor { get; set; } // defaults to transparent -- more efficient if
        opaque

    // Open/Close semantics
}

```

```

protected StaticDrawingContext PaintingOpen();
protected StaticDrawingContext PaintingAppend();

// RenderCore semantics
protected abstract void RenderCore(Rect invalidRect);
protected void InvalidateRender();
protected void InvalidateRender(Rect invalidationRect);
}

```

Aunque el RetainedVisual agrega una corriente de instrucción retenida, el PaintingVisual es esencialmente respaldado por una superficie. El sistema puede virtualizar la superficie y aún mantener instrucciones de representación en tanto que los requisitos de desempeño se cumplan. En razón de esto, la superficie no es accesible por el usuario.

Una diferencia del PaintingVisual con relación al RetainedVisual es que los métodos suministran un StaticDrawingContext que no permite animación. Si un argumento animado se utiliza en cualquier parte será arrojada una excepción. Otra diferencia es que el "Append" crecerá a la corriente de instrucciones mucho más barato en términos de costo de memoria. Adicionalmente, el PaintingBounds, que esencialmente ajusta un corte duro sobre el usuario, es requerido y es necesario. Nótese que este es diferente de la propiedad de corte, y esta se une al contenido del visual mientras que Clip cortará el contenido de este visual y todos

sus hijos. El `RenderCore` (análogo a `IRetainedVisual.Render`) también se implementa en donde sí los cambios de resolución o si el sistema necesita representar el contenido por alguna razón, el usuario suministra este mecanismo.

Un `PaintingVisual` es potencialmente mucho más ligero que un `SurfaceVisual`, y no existe un respaldo explícito de este visual por una superficie. En su lugar, este es un nodo en un árbol de despliegue que es en algún punto inferior respaldado por una superficie.

Para lograr un `SurfaceVisual`, el usuario debe crear un `RetainedVisual`, llamar el `DrawImage` y luego cambiar la imagen detrás de las escenas. En ese caso el usuario está explícitamente controlando la pasterización en lugar de permitirle al sistema llamar de regreso. Habrá un modo inmediato API para trabajar en la imagen directamente. Este API le permitirá al usuario conseguir un `StaticDrawingContext` que trabaja sobre esa imagen. Nótese que el API para una `SurfaceVisual` es similar a un nodo de árbol de pantalla `hWnd`, `DUser Gadget` o un `Trident`. El contenido "Appending" (de hecho-haciendo un pequeño delta que componen sobre lo que ya está allí) es una operación barata. Existe por mucho, o nom existe por mucho pena de memoria para el contenido de apéndice, a menos a diferencia de un `RetaindenVisual` por medio del cual el `RenderAppend` originará que una corriente de

instrucción consiga más y más, potencialmente conduciendo a un crecimiento. En razón a que la superficie que puede ser respaldada este visual puede venir e ir, el visual es requerido para implementar un virtua "por demanda" RenderCore.

Un escenario de uso principal para el PaintingVisual es para portar código de aplicación que es grandemente estructurado alrededor de un modelo de pintura WM_PAINT. Este es también útil para contenido estático que es raramente cambiante y denso. Nótese que el PaintingVisual puede ser respaldado por un meta archivo o puede ser a la superficie cierta. El sistema puede decidir un tiempo de corrido que es más apropiado, por ejemplo, con base en la memoria y las preocupaciones de desempeño. Sin embargo, se garantiza que, pasado cierto punto, con el apéndice del nuevo contenido no costará más memoria. Nótese que el sistema puede cambiar entre almacenamiento de meta archivo y superficie según se necesite.

OBJETOS MIL DE MÁXIMO NIVEL

Como se puede apreciar fácilmente, varios objetos son suministrados para operar en un escenario con ventana típica. Nótese que estos no son necesariamente clases formales (por

ejemplo no existe una interfase u objetor de programador explicita).

Uno de tales objetos comprende un árbol visual, que es un objeto que contiene el contenido principal a ser dibujado. Los controles de derivarán de los visuales del árbol directamente. Los Visuals son dispositivo y contexto independientes.

Un objetivo de representación es el dispositivo al cual el visor es dibujado. Este objeto (por ejemplo pantalla) puede tener su propia suciedad o mecanismo de invalidación, que es necesario para regresar el sistema visual por medio de un hWnd. Varios objetivos de representación incluyen una pantalla en una ventana, un Printer, un Metafile, un Surface, una "Sub-Window" tal como una parte de la escena que es dibujada separadamente del resto de la decena. Este es el mecanismo para posibilitar dibujo de hilo cruzado, y es equivalente a un objeto componible de motor de más bajo nivel.

Otros objetos relacionados con dibujo incluyen un VisualRender, que comprende un objeto que está configurado par dibujar el árbol visual sobre el objetivo de representación, y un objeto Display Scheduler que conoce cuando dibujar el árbol visual sobre el objetivo de

representación. Un Time Manager es un objeto de contexto para un conjunto de nodos de tiempo, y es el objeto que el programador llama a pegarse.

El siguiente es un flujo de ejemplo de control para dibujar en la pantalla:

1. El usuario adquiere el UiContext de alguna manera e inicia modificando el Visual Tree. Esto puede ser durante la aplicación de arranque quizás en respuesta a un evento en cada UI.
2. La modificación sucia propaga hasta el árbol Visual. El Visual raíz conoce cual visual representará a su asociado y envía la ratificación sucia. Esta ratificación es privada.
3. el representador visual, por vía de un evento público, reporta que este ha sido cambiado y está afuera de sync con su objetivo de representación.
4. El programador decide cuando y donde rectificará de hecho está situación y originará que ocurra el dibujo. Típicamente esto se hace al enviar por correo un ítem de trabajo al despachador. Sin embargo, el usuario puede hacer algo más.

5. El usuario produce el UiContext y le permite al despachador correr.

6. El despachador corre y llama un ítem de trabajo diferido del programador. (más probablemente cualquiera ítems de trabajo diferidos en un contexto simple serán colegidos. También, este puede correr paso seguro con el sistema de diseño para reducir desperdicio).

7. El programador corre su loop principal de actualización:

a. Pega el TimeManager

b. Corre el diseño, entre otras cosas

c. Llama el representador visual para representar los cambios en el objetivo representador. El representador entonces:

i. camina las partes sucias del Visual Tree y actualiza las cajas de unión con cache interno.

ii. Llama todos los visores necesarios "por demanda" para representación. (por defecto los visuales "por demanda" tendrán una recta vacía en sus uniones y así

71

ellas no serán llamadas hasta que el diseño corra y se ajuste).

- iii. Camina las partes sucias del huevo y envía las actualizaciones de representación necesarias a los sistemas gráficos de nivel más bajo.

Nótese que el sistema visual no conoce nada acerca del despachador. Depende del objeto del programador tener cuidado de aquellos detalles. El programador puede hacer cualquier flujo de control apropiado.

Adicionalmente, existe la idea de un representador visual creciente y un representador visual instantáneo. Puede ser deseable para un visual pertenecer a uno y solamente a un representador visual creciente a la vez. Esta restricción es necesaria para el cacheo eficiente de los datos sobre el visual mismo. Sin embargo, es razonable también tener manera de "hacer instantánea" un árbol visual completo a un objetivo representador. En este caso no existen conexiones persistentes entre el árbol visual y el representador. Esto se podría utilizar para conseguir una captura de pantalla de alta resolución o para enviar el árbol visual (en la medida en que este en la pantalla) directamente a la impresora.

520

76²

La ventana juega el papel del objetivo representador, anterior. Esta es también un reemplazo manejado para hWnd.

```
class System.Windows.Media.WindowContext
: IDisposable
{
    public WindowContext();

    // IDispose
    public void Dispose();
    public bool IsDisposed { get; set; }

    // Window size
    public Size Size { get; }
    public event EventHandler SizeChanged { add; remove; }

    // Window validation
    public bool IsInvalid { get; }
    public event EventHandler IsInvalidChanged { add; remove; }

    // Resolution information
    public ResolutionInformation ResolutionInformation { get; }
```

```
    public event EventHandler ResolutionInformationChanged { add; remove; }
```

```
}
```

El control manejador de ventana está por fuera de este objeto, sin embargo puede ser integrado con el WindowContext al hacer las propiedades (por ejemplo, tales como Size)

521

valores de lectura/escritura, y suministrar propiedades adicionales tales como sitio, título de ventana, y así sucesivamente. Nótese que Size representa el tamaño de la ventana en unidades físicas (1/96° de cada pulgada). Este no es un tamaño de píxel. Nótese que podrían haber situaciones en donde los datos representados en la ventana se pierdan por alguna razón, tal como el interruptor de modo de video o un instructor de una consola local a una sesión de servidor de terminal remoto.

Los VisualRenderers y los VisualManagers son otros objetos, y responsables por representar un árbol de Visuals a un objeto representador. Un VisualRenderer suministra un modelo simple "de un disparo" que representará un medio aunque el VisualManager establezca una conexión retenida entre el árbol de visuales y el objetivo a que ellos están representando. Esta soporta un representador "creciente" al medio.

El siguiente es un ejemplo de lo que el VisualRenderer base semeja en una implementación:

264

```

public class System.Windows.Media.Visuals.VisualRenderer :
    System.Threading.ContextAffinityObject,
    IDisposable
{
    // Constructors
    internal VisualRenderer();
    internal VisualRenderer(System.Threading.UiContext context);

    // ContextAffinityObject
    public UiContext Context { get; }
    public bool CheckContext();
    public void VerifyContext();
    public void SetContext(System.Threading.UiContext context);

    // IDisposable+

```

```

    public void Dispose();
    public bool IsDisposed { get; }

    // Properties
    public Color BackgroundColor { get; set; }
    public Visual RootVisual { get; set; }

    // Device information
    public Size RootSize { get; }
    public ResolutionInformation ResolutionInformation { get; set; }

    // Render control
    public void RenderNow();
}

```

523

705

PCT (la versión de papel la cual se actualiza usualmente dos veces al año y la versión de Internet la cual se actualiza usualmente sobre una base semanal). Finalmente, una tabla acumulativa de todos los límites de tiempo aplicables para entrar a la fase nacional está disponible del sitio de Internet de la WIPO, por vía de enlaces a varias páginas, el sitio incluye aquellas de la Gaceta; Diario y Guía, en <http://www.wipo.int/pct/en/index.html>.

Información acerca de los requerimientos para presentar una petición para examen preliminar internacional se establece en la Guía del Solicitante PCT, Volumen I/A, Capítulo IX. Note que solo un solicitante quien es un nacional o residente de un Estado Contratante del PCT que se une mediante el capítulo II tiene el derecho de registrar una petición par examen preliminar internacional (actualmente, todos los Estados contratantes están unidos por el Capítulo II).

REQUERIMIENTOS CON RESPECTO A LOS DOCUMENTOS DE PRIORIDAD

Para solicitantes quienes aún no han cumplido con los requerimientos con respecto a los documentos de prioridad, se recuerda lo siguiente.

443

Cuando la prioridad de una solicitud internacional, nacional o regional anterior es reivindicada, el solicitante debe presentar una copia de dicha solicitud anterior, certificada por la autoridad con la cual esta fue registrada ("el documento de prioridad") a la Oficina Receptora (la cual transmitirá a la Oficina Internacional) o directamente a la Oficina Internacional, antes del tiempo de expiración de 16 meses a partir de la fecha de prioridad, dado que cualquier documento de prioridad puede todavía ser presentado a la Oficina Internacional antes que la fecha de publicación internacional de la solicitud internacional, en cuyo caso tal documento será considerado haber sido recibido por la Oficina Internacional en el último día del límite de tiempo de 16 meses (Regla 17.1(a)).

Cuando el documento de prioridad es publicado por la Oficina receptora, el solicitante puede, en lugar de presentar el documento de prioridad, solicitar a la Oficina receptora preparar y transmitir el documento de prioridad a la Oficina Internacional. Tal solicitud debe ser hecha antes de la expiración del límite de tiempo de

767

16 meses y puede ser sometida por la Oficina receptora al pago de una tarifa (Regla 17.1 (b)).

Si el documento de prioridad concerniente no es presentado a la Oficina Internacional o se la solicitud a la Oficina receptora para preparar y transmitir el documento de prioridad no ha sido hecho (y la tarifa correspondiente, si existe pagada) dentro del límite de tiempo aplicable indicado de acuerdo con los párrafos precedentes, cualquier Estado designado puede desatender la prioridad reivindicada, dado que ninguna Oficina designada puede desatender la prioridad reivindicada relacionada antes de dar al solicitante una oportunidad, al momento de la entrada en fase nacional, para presentar el documento de prioridad dentro del límite de tiempo que es razonable de acuerdo con las circunstancias (Regla 17.1 (c)).

Cuando son reivindicadas varias prioridades, la fecha de la prioridad a ser considerada para los propósitos de computar el límite de tiempo de 16 meses (y todos los otros límites de tiempo PCT) es la fecha de presentación de la solicitud más reciente cuya prioridad sea reivindicada (Artículo 2(xl)(b)).

Formulario PCT/IB/301 (Anexo) (Enero 2004)

441

768

CAPA DE INTEGRACIÓN DE MEDIOS

CAMPO DE LA INVENCIÓN

La invención se relaciona de manera general con sistemas de computadora, y más particularmente con el procesamiento de información gráfica y otra información de video para desplegar sobre los sistemas de computadora.

ANTECEDENTES DE LA INVENCIÓN

Los límites del modelo de modo intermedio tradicional de acceder gráficas sobre sistemas de computadora están siendo alcanzados, en parte porque las velocidades de la memoria y el bus no sean mantenido con el avance en los procesadores principales y/o los procesadores gráficos. En general, el modelo habitual (por ejemplo, WM_PAINT) para preparar un marco requieren demasiado procesamiento de datos para mantenerse con la proporción de refresco del equipo cuando son deseados efectos gráficos complejos. Como resultado, cuando los efectos gráficos complejos se intentan con modelos gráficos convencionales, en lugar de completar los cambios que dan como resultado efectos visuales percibidos en tiempo para el siguiente cuadro, los cambios se pueden agregar sobre diferentes cuadros, originando resultados que son visual y notoriamente indeseables.

944

Un nuevo modelo para controlar salida de gráficos se describe en la solicitud de patente estadounidense No 10/184,795, 10/184,796, 10/185,775, 10/401,717, 10/401,322, y 10/401,268, traspasada al cesionario de la presente invención e incorporada aquí como referencia. Este nuevo modelo suministra un número de significativas mejoras en la tecnología de procesamiento de gráficos. Por ejemplo, la U.S. Serie No.10/184,795 esta dirigida de manera general hacia un sistema y método de procesamiento de gráficos de nivel múltiple, en el cual un componente de mayor nivel (por ejemplo de un sistema operativo) desarrolla aspectos computacionalmente intensivos de construcción de un gráfico de escena, actualiza parámetros de animación y atravesando las estructuras de gráficos a una proporción de operación relativamente baja, con el fin de pasar estructuras de datos simplificadas y/o comandos de gráficos a un componente de bajo nivel. En razón aunque el pensamiento de alto nivel simplifica grandemente los datos, el componente de bajo nivel puede operar a una proporción más rápida, (con relación al componente de alto nivel), tal como una proporción que corresponde a la proporción refresco del cuadro del sub sistema de gráficos, para procesar los datos en datos de salida constantes para el sub sistema de gráficos. Cuando se utiliza animación, lugar de tener que redibujar una escena completa con cambios, el procesamiento de bajo nivel puede interpolar intervalos de parámetro según sea necesario para

x70

obtener valores instantáneos que cuando se producen suministran una escena ligeramente cambiada para cada cuadro, suministrando animación suave.

El documento U.S. Serie No 10/184,796 describe una gráfica de escena parametrizada que suministra valores mutables (animados) y contenedores de gráfico parametrizado tal como los códigos de programa que quieren dibujar gráficos (por ejemplo un programa de aplicación o un componente de sistema operativo) puede cambiar selectivamente ciertos aspectos de la descripción del gráfico de escena, aunque dejando otros aspectos intactos. El código de programa también puede reutilizar porciones ya construidas del gráfico de escena, con parámetros posiblemente diferentes. Como se puede apreciar, la capacidad para cambiar fácilmente la apariencia de desplegar ítems por vía de la parametrización y/o la reutilización de partes existentes de un gráfico de escena suministra ganancias sustanciales en toda la eficiencia de procesamiento de gráficos.

El documento U.S. Serie No.10/185,775 describe de manera general una estructura de datos de cache y mecanismos relacionados para almacenar información visual por vía de objetos y datos en un gráfico de escena. La estructura de datos está generalmente asociada con mecanismos que controlan inteligentemente como la información visual allí es poblada y

446

771

utilizada. Por ejemplo, a menos que específicamente se solicite por el programa de aplicación, la mayoría de la información en la estructura de datos no tiene referencia externa a está lo que posibilita que está información se optimizada o procesada de otra manera. Como se puede apreciar, esto suministra eficiencia y conservación de recursos, por ejemplo, los datos en la estructura de datos cache puede ser procesada en un formato diferente que es más compacto y/o reduce la necesidad de procesamiento subsecuente, repetido, tal como un mapa de bits u otro resultado post procesamiento.

Aunque las mejoras anteriores suministran beneficios sustanciales en la tecnología de procesamiento de gráficos, subsiste la necesidad de una forma de programas para utilizar efectivamente este modelo de gráficos mejorado y sus otras mejoras relacionadas de una manera directa. Lo que se requiere es un modelo comprensivo aunque directo para programas para tomar ventaja de las muchas características y capacidades de procesamiento de gráfico suministrada por el modelo de gráfico mejorado y sacar de esta manera gráficos complejos y datos audiovisuales de una manera eficiente.

4449

772

RESUMEN DE LA INVENCION

La presente invención suministra una capa de integración de medios que suministra un API para los programadores para lograr la composición de efectos posiblemente compleja dentro de sus aplicaciones de una manera directa, aunque apalancando la unidad de procesamiento de gráficos de una manera que no impacte adversamente el desempeño de la aplicación normal. Un aspecto suministra la capacidad de convidar diferentes tipos de medio (por ejemplo, 2D, 3D, Video, Audio, texto, imágenes y así sucesivamente) y animarlas tanto suave como sin uniones.

El MIL suministra una arquitectura de gráficos para composición multi-etapa, y un modelo de programación que permite la pariedad funcional en las interfases programática y de scripted. Un API y un script permiten la creación de una estructura o escenas retenidas que se compone cuando se genera, aun incluye áreas que tienen más naturaleza de modo inmediato.

Por vía de las interfases, el MIL suministra acceso a una estructura de datos para almacenar información visual de tal forma que las aplicaciones pueden tomar ventaja de las capacidades gráficas suministradas por el equipo de computadora. Las interfases soportan un modelo objeto de

950

78

elemento y un lenguaje demarcado da gráficos de vector para utilizar ese modelo objeto de elemento de una manera que le permita a los desarrolladores de código de programa hacer interfase consistentemente con una estructura de gráficos de escena para producir gráficos. La estructura de datos también se puede utilizar para hacer directamente o para "compilar" la información visual de tal forma que está se pueda suministrar a un sistema de gráficos de más bajo nivel para una composición y animación rápida.

El modelo de objeto de elemento de gráficos de vector corresponde generalmente a unos elementos de forma y a otros elementos que incluyen elementos de imagen y video que se correlacionan con un modelo de objeto gráfico de escena del gráfico de escena. El marcado puede ser analizado sintácticamente en datos que incluyen elementos en un árbol de elementos que es traducido en los objetos de una estructura de datos de gráfico de escena. Otro marcado puede ser producido directamente en datos y llamadas que crean los objetos gráficos de escena. El lenguaje demarcado suministra formas distintas de describir un elemento, incluyendo un formato de cuerda simple o sintaxis de propiedad compleja, que puede ser nombrada, posibilitando la reutilización en otros sitios en el marcado. Un aspecto del MIL es la integración de animación y tiempo a través del conjunto API, suministrando la animación como un concepto de nivel base

451

274

inherente. Para facilitar la animación suave, el MIL suministra un sistema y un método de procesamiento de gráficos multi-nivel (por ejemplo de un sistema operativo). Uno de tales sistemas de procesamiento de gráficos multi-nivel comprende dos componentes, que incluye un tick por demanda o un componente de alto nivel de tick lento, y un componente de bajo nivel rápido (por ejemplo en los equipos a proporción de refresco de cuadro de equipo de gráficos. (En general, el componente de alto nivel menos frecuente desarrolla aspectos computacionalmente intensivos para actualizar los parámetros de animación y atravesar estructuras de datos de escena, con el fin de pasar estructuras de datos simplificadas al componente de bajo nivel. El componente de bajo nivel opera a una frecuencia mayor, de tal forma que la proporción de refresco de cuadro del subsistema de gráficos, procesa las estructuras de datos en datos de salida constante para el subsistema de gráficos. El procesamiento de bajo nivel incluye interpolar cualquier intervalo de parámetro según sea necesario para obtener valores instantáneos para hacer la escena para cada cuadro de animación.

Los objetos MIL de alto nivel incluyen un árbol visual, que es un objeto que contiene el contexto principal a ser dibujado. Los controles se derivarán de los visuales del árbol directamente. Los visuales son dispositivos y contextos

272

295

padres independientes. Un objeto producido es el dispositivo al cual el árbol es dibujado. Este objeto (por ejemplo pantalla) puede tener su propio mecanismo sucio o de invalidación. Varios objetivos de representación incluyen una pantalla en una ventana, una impresora, un metaarchivo, una superficie, un archivo de medios de cuerda (por ejemplo, DVD) y una "sub ventana" que es parte de la escena que es dibujada separadamente del resto de la escena. Otros objetos relacionados influyen un Visual Render, que comprende un objeto que es configurado para dibujar un árbol visual sobre un objetivo de representación, y un objeto de Display Scheduler que sabe cuando dibujar el árbol visual sobre el objetivo de representación. Un Time Manager es un objeto de contexto para un conjunto de nodos de tiempo, y es el objeto que el planificador llama "tick on".

Un API Visual es suministrado, el cual es esencialmente un punto de partida para dibujar por vía de la capa de integración de medios, y comprende múltiples tipos de objetos, incluyendo un objeto VisualManager, que conecta el Visual Tree aun medio. Los diferentes tipos de VisualManager (por ejemplo Screen, Printer, y Surface) son responsables por representar un Visual Tree en su medio particular. Un visual es donde el programador hace el dibujo; este es un nodo del árbol visual y suministra un lugar para un programa a dibujar.

457

/ 776

Los API DrawingContext representan un modelo de programación basado en contexto para como construir contenido visual que pueble un visual o son representados a un ImageData. Las clases de DrawingContext son suministradas, así como también las clases y puntos de entrada necesarios para adquirir un DrawingContext y enumerar el contenido visual en un RetainedVisua/DrawingVisual.

Para posibilitar la mutabilidad, se suministra un conjunto simple de tipos que se derivan de una clase básica común Changeable. Cualquier tipo para el cual mutablemente se desee cambiar puede derivar de la clase Changeable. Por ejemplo, en una programación de gráficos, el modelo objeto incluye Brushes, Pens, Geometries, FloatAnimations, GradientStops, Segments, y así sucesivamente. Una propiedad IsChangeable especifica si el objeto cambiabile puede ser modificado o no, dependiendo de su valor habitual, que define el estado.

Una brocha es un objeto aunque representa un método para llenar un plano. Además de ser capaz de llenar un plano de una manera absoluta, las brochas de la capa de integración de medios también son capaces de adaptarse como ellas llenan el plano con relación al tamaño y el objeto que ellas están llenando. Ejemplos de tipos de brochas influyen SolidColorBrush, VisualBrush (que puede referenciar un

4541

227

recurso de gráficos vector/Visual), DrwingBrush, LinearGradient, RadialGradient, ImageBrush y NineGridBrush. Ciertos objetos de brocha tendrán una idea de cómo ellos relacionan para coordinar el sistema cuando ellos son utilizados, y una idea de cómo ellos se relacionan con la caja de unión de la geometría con la cual ellos son utilizados. Este tamaño está basado en el objeto que la brocha está llenando. Cierta tipo de brochas, (por ejemplo VisualBrush) también pueden ser inclinadas para crear patrones definidos de programador. La clase base Brush tiene un Transform, una opacidad general, un modo de mezcla. Brush (y otros frentes de objeto en VectorGraphics y el MIL API) los objetos son Changeables y son escribibles después que ellos ha sido creados, y siguen el patrón Changeable general para como ellos se comportan después de que ellos son utilizados en uso calificado.

Una clase Geometry de objetos puede ser utilizada para enganchar, ensayar con un golpe, y representar unos datos basados en vector 2D con el Pen y el Brush. Las clases de Geometry derivadas suministran más construcciones específicas y semánticas de la numeración. Un número de tipos de Geometry específicos en forma son suministrados, así como también una PathGeometry generalizada que permite la definición explícita de Geometry de forma más compleja. Geometry es una clase base abstracta. Una GeometryCollection es una colección de objetos

63

de geometría múltiple que han sido combinados utilizando operaciones `CombineMode` en su área definida. Este objeto permite combinaciones visuales de construcción de más fáciles de objetos de geometría de que lo que es posible utilizando estrictamente `PathFigure` dentro de un `PathGeometry`.

`ImageSource` es una clase abstracta, que comprende un bloque de construcción básico para imágenes. Un `ImageSource` conceptualmente representa un conjunto simple, constante de píxeles a un cierto tamaño y resolución. Por ejemplo, una `ImageSource` puede ser un cuadro simple en un archivo de imagen que un decodificador podría suministrar, o puede ser el resultado de una transformación que opera sobre una cierta `ImageSource` de su propiedad. Un `ImageSource` es cambiable, no porque sus propias propiedades puedan ser cambiadas, sino por que las propiedades de sus subclases pueden ser potencialmente cambiadas.

Una clase `Transform` de objetos es suministrada para escalar, rotar, trasladar y sesgar vectores y gráficos rasterizados. Las clases derivadas de `Transform` suministran uso amigable y semánticas de enumeración.

Los efectos suministran unos medios para alterar los contenidos visuales de una escena de una manera céntrica de representación. Por ejemplo, `ImageEffects` (efecto de mapa de

77A

bits basado en pasterizado) opera sobre una representación basada en imagen completamente compuesta de una porción de una escena. Los efectos son rotos en varios tipos incluyendo ImageEffects, BlendModes y VectorEffects. ImageEffects puede ser utilizado en una escena de modo retenido al aplicarlo a un sub gráfico o a un Element, o puede ser utilizado en la cadena de procesamiento de imagen que permanece sola. BlendModes son una forma específica de efectos basados en imagen, y se puede aplicar para escena de modo retenido en generalmente la misma manera que ImageEffects. Los modos combinados desarrollan una combinación de la fuente y colores de destinación en la medida en que la fuente es compuesta, por ejemplo, multiplicado o agregado.

El ensayo de golpe es utilizado para recoger visuales en una escena y opera al arrancar desde la parte superior del árbol de control, y regresar un control o conjunto de controles por medio de un punto geometría. Un control puede definir si este es golpe o no con servicios de soporte que incluyen geometría representada, caja de unión, geometría fuera de banda (región de golpe), opacidad de imagen o máscara o su propia lógica. El control puede regresar datos relacionados con golpes específicos sobre el golpe. El mecanismo de ensayo de golpe puede filtrar los resultados de ensayo de golpe de una manera eficiente. El recorrido de ensayo de golpe es un recorrido de derecha a izquierda profundo del árbol visual, con golpes

457

780

recortados a través de un devolvedor de llamadas en forma de orden Z de arriba hacia el fondo. Cuando se desciende, el proveedor del filtro en términos de relaciones de nivel de elemento, por ejemplo, un lienzo con formas, o un panel de acoplar con un lienzo interior. Cuando ocurre un golpe el probador de golpe puede { continuar procesando golpes adicionales (si existe alguno) o parar.

Se suministra un sistema de animación, comprendido de un motor de control de tiempo y un conjunto de objetos de animación. El motor de tiempo es un servicio que se puede utilizar por cualquier objeto que exhiban comportamientos variantes de tiempo, animación y audio u objetos de medios de video. Los objetos de animación implementan un conjunto de funciones que el tiempo de mapa expande en otros tipos de datos, que son entonces utilizados como entradas en otros objetos de nivel mayor. La animación gráfica se logra al asociar una colección de animación con una operación de representación. Cada animación utilizada en una operación de representación puede ser corrida sobre un reloj separado, denominado como un llamado una "línea de tiempo". Las líneas de tiempo múltiples se pueden organizar en árboles de tiempo o soportar tiempo jerárquico. Una vez que el primitivo animado es dibujado y los parámetros de animación especificados, el sistema de representación de bajo nivel tiene cuidado de redibujar la escena de intervalos regulares.

450

781

Cada vez que un cuadro es representado el valor habitual de las animaciones involucradas en la escena es computado, con base en el tiempo vencido (en la mayoría de los casos debido por el reloj del sistema) y luego los primitivos animados son redibujados.

Varios tipos de primitivo, características de color y soporte de medios son también suministrados por vía del MIL. El MediaData puede ser utilizado para jugar cualquier contenido de audio/video.

Otros beneficios y ventajas serán evidentes de la siguiente descripción detallada cuando se toma en conjunto con los dibujos, de los cuales:

BREVE DESCRIPCIÓN DE LOS DIBUJOS

La figura 1 es un diagrama de bloque que representa un sistema de computadora de ejemplo en la cual puede ser incorporada la invención presente;

La figura 2 es un diagrama de bloque que representa de manera general una arquitectura de capa de gráficos en la cual la presente invención se puede incorporar;

784

27

La figura 3 es una representación de un gráfico de escena de visuales y de componentes asociados para procesar el gráfico de escena tal como atravesar el gráfico de escena para suministrar comandos gráficos y otros datos de acuerdo con un aspecto de la presente invención;

La figura 4 es una representación de un gráfico de escena de visuales de validación, dibujar visuales y Instruction List asociados contruidos de acuerdo con un aspecto de la presente invención;

La figura 5 es una representación de una clase visual, de un modelo de objeto, de acuerdo con un aspecto de la presente invención;

La figura 6 es una representación de varios otros objetos del modelo objeto, de acuerdo con un aspecto de la presente invención;

La figura 7 es una representación de una jerarquía de clase de transformación, de acuerdo con un aspecto de la presente invención;

Las figuras 8 y 9 son representaciones de transformaciones de unos datos visuales en una escala de geometría y una escala

460

no uniforme, respectivamente, de acuerdo con un aspecto de la presente invención;

La figura 10 es una representación de las clases de geometría del modelo objeto, de acuerdo con un aspecto de la presente invención;

La figura 11 es una representación de una estructura PathGeometry, de acuerdo con un aspecto de la presente invención;

La figura 12 es una representación de un gráfico de escena de visuales Instruction List que muestran gráficos de ejemplo producidos por los primitivos, de acuerdo con un aspecto de la presente invención;

La figura 13 es una representación de clases de brocha del modelo objeto, de acuerdo con un aspecto de la presente invención;

La figura 14 es un diagrama de bloque que representa generalmente una arquitectura cambiante en la cual las solicitudes son procesadas mediante una máquina de estado para controlar la mutabilidad del tipo, de acuerdo con un aspecto de la presente invención;

Las figuras 15-17 son diagramas de estado que representan los estados de propiedad controlan el comportamiento de los tipos mutables, de acuerdo con un aspecto de la presente invención;

Las figuras 18-23 son representaciones jerárquicas de objetos en una gráfica de escena de ejemplo que muestra como las propiedades de estado de control hacen transición y clonan los comportamientos del código de ejemplo, de acuerdo con un aspecto de la presente invención;

La figura 24 es una representación de gráficos representados que resultan de los datos en un objeto de brocha de un gradiente lineal, de acuerdo con un aspecto de la presente invención;

La figura 25 es una representación de gráficos representados que resultan de los datos en un objeto de brocha de gradiente radial, de acuerdo con un aspecto de la presente invención;

La figura 26 es una representación de gráficos representados que resultan de tener varios valores estirados, de acuerdo con un aspecto de la presente invención;

La figura 27 es la representación de gráficos representados que resultan de tener varios valores de mosaico, de acuerdo con un aspecto de la presente invención;

785

La figura 28 es una representación que un objeto de brocha de 9 parrillas representado de acuerdo con un aspecto de la presente invención;

Las figuras 29-40 son representaciones gráficas de líneas de tiempo de ejemplo utilizadas en animación, de acuerdo con un aspecto de la presente invención;

La figura 41 es una representación de una imagen tridimensional de ejemplo construida por vía de un visual tridimensional de acuerdo con un aspecto de la presente invención;

La figura 42 es una representación de conceptos tridimensionales para suministra soportes tridimensionales de acuerdo con un aspecto de la presente invención.

DESCRIPCIÓN DETALLADA

AMBIENTE OPERATIVO DE EJEMPLO

La figura 1 ilustra un ejemplo de un ambiente de sistema de cómputo adecuado 100 en el cual la invención puede ser implementada. El ambiente de sistema de cómputo 100 es solamente un ejemplo de un ambiente de computo adecuado y no

405

pretende sugerir ninguna limitación en el alcance de uso o funcionalidad de la invención. Ni se debe interpretar el ambiente de cómputo 100 como teniendo alguna dependencia o requisito que relacione a cualquiera combinación de componentes ilustrados en el ambiente operativo de ejemplo 100.

La invención es operacional con otros numerosos propósitos generales o ambientes de sistema de cómputo con propósito especial o configuraciones. Los ejemplos de sistemas de cómputo bien conocidos, ambientes, y/o configuraciones que pueden ser adecuadas para uso con la invención incluyen, pero no están limitados, computadores personales, computadores servidores, dispositivos de agendas de mano o computadores portátiles, dispositivos tableta, sistemas multiprocesador, sistemas basados en microprocesador, cajas de codificadores, electrónicos de consumo programables, PCs de red, minicomputadores, computadores gigantes, ambientes de computo distribuido que incluyen cualquiera de los sistemas o dispositivos anteriores, y similares.

La invención de puede describir en el contexto general de las instrucciones ejecutables por computadora, tal como, los módulos de programa, siendo ejecutados por una computadora. Generalmente, los módulos de programa incluyen rutinas, programas, objetos, componentes, estructuras de datos, y así

787

sucesivamente, lo que desarrolla tareas particulares o implementa tipos de datos abstractos particulares. La invención también se puede practicar en ambientes de cómputo distribuido cuando las tareas son desarrolladas mediante dispositivos de procesamiento remoto que están ligados a través de una red de comunicaciones. En un ambiente de cómputo distribuido, los módulos de programa se pueden localizar tanto en medios de almacenamiento de cómputo locales como remotos que incluyen dispositivos de almacenamiento de memoria.

Con referencia a la figura 1 un sistema de ejemplo para implementar la invención incluye un dispositivo de cómputo de propósito general en la forma de una computadora 110. Los componentes de la computadora 110 pueden incluir, pero no están limitados a, una unidad de procesamiento 120, una memoria del sistema 130, un bus de sistema 121 que acopla varios componentes del sistema que incluyen la memoria del sistema a la unidad de procesamiento 120. El bus del sistema 121 puede ser cualquiera de varios de tipos de estructuras de bus que incluyen un bus de memoria o un controlador de memoria, un bus periférico, y un bus local que utiliza cualquiera de una variedad de arquitecturas de bus. Por vía de ejemplo, y no de limitación, tales arquitecturas incluyen el bus Industry Standard Architecture (ISA), el bus Micro Channel Architecture (MCA), el bus Enhanced ISA (EISA), el bus

458

788

local Video Electronics Standards Association (VESA), el bus Accelerated Graphics Port (AGP), y el bus Peripheral Component Interconnect (PCI) conocido bus Mezznine.

El computador 110 típicamente incluye de medios por medios leíbles por computadora. Los medios leíbles por computadora pueden ser cualesquier medios disponibles que se puedan acceder por la computadora 110 e incluye tanto medios volátiles como no volátiles, y medios removibles y no removibles. Por vía de ejemplo, y no de limitación, los medios leíbles por computadora pueden comprender medios de almacenamiento de computadora y medios de comunicación. Los medios de almacenamiento de computadora incluyen tanto medios volátiles como no volátiles, removibles como no removibles implementados en cualquier método o tecnología para almacenar información tal como instrucciones leíbles por computadora, estructuras de datos, módulos de programa u otros datos. Los medios de almacenamiento de computadora, incluyen, pero no están limitados a, RAM, ROM, EEPROM, memoria flash u otra tecnología de memoria, CD-ROM, discos versátiles digitales (DVD), otros almacenamientos de disco óptico, cassettes magnéticos, cintas magnéticas, almacenamiento de disco magnético u otros dispositivos de almacenamiento magnético, o cualquier otro medio que se pueda utilizar para almacenar la información deseada y que pueda ser acezada por la computadora 110. Los medios de comunicación típicamente

466

289

tienen modalidades de instrucciones leíbles por computadora, estructuras de datos, módulos de programa u otros datos en una señal de datos modulada tal como una onda portadora otro mecanismo de transporte e incluye cualesquier medios de suministro de información. El termino "señal de datos modulada" significa una señal que tiene una o más de sus características ajustada o cambiada de tal manera que codifica información en la señal. Por vía de ejemplo, y no de limitación, los medios de comunicación incluyen medios alámbricos tales como una red alámbrica o una conexión alámbrica directa, y medios inalámbricos tal como medios acústicos, RF, infrarrojos y otros medios inalámbricos. Las combinaciones de cualquiera de los anteriores también se debe incluir dentro del alcance de los medios leíbles por computadora.

La memoria del sistema 130 incluye medios de almacenamiento por computadora de la forma de memoria volátil y/o no volátil tal como memoria de solo lectura (ROM) 131 y memoria de acceso aleatorio (RAM) 132. Un sistema de entrada/salida básica 133 (BIOS), que contiene las rutinas básicas que ayudan a transferir información entre elementos dentro de la computadora 110, tal como durante el arranque, es típicamente almacenado en la ROM 131. La RAM 132 típicamente contiene módulos de datos y/o programas que son inmediatamente accesibles y/o actualmente son operados por una unidad de

402

procesamiento 120. Por vía de ejemplo, y no de limitación, la figura 1 ilustra un sistema operativo 134, programas de aplicación 135, otros módulos de programa 136 y datos de programa 137.

La computadora 110 también puede incluir otros medios de almacenamiento por computadora removibles/no removibles, volátiles/no volátiles. Por vía de ejemplo solamente, la figura 1 ilustra una unidad de disco duro 141 que lee o escribe a un medio magnético no removible, no volátil, una unidad de disco 151 que lee o escribe a un disco magnético removible, no volátil 152, y una unidad de disco óptico 155 que lee o escribe a un disco óptico removible, no volátil 156 tal como un CD-ROM u otro medio óptico. Otros medios de almacenamiento por computadora removibles/no removibles, volátiles/no volátiles que se puede utilizar en el ambiente operativo de ejemplo incluyen, pero no están limitados a, cassettes de cinta magnética, tarjetas de memoria flash, discos versátiles digitales, cintas de video digital, RAM de estado sólido, ROM de estado sólido, y similares. La unidad de disco duro 141 está típicamente conectada al bus del sistema 121 a través de una interfase de memoria no removible tal como la interfase 140, y la unidad de disco magnético 151 y una unidad de disco óptico 155 que son típicamente conectadas al bus del sistema 121 mediante una interfase de memoria tal como la interfase 150.

Las unidades y sus medios de almacenamiento de computadora asociados, discutidos anteriormente ilustrados en la figura 1, suministran almacenamiento de instrucciones leíbles por computadora, estructuras de datos, módulos de programa y otros datos para la computadora 110. En la figura 1, por ejemplo, la unidad de disco duro 141 se ilustra como sistema operativo de almacenamiento 144, los programas de aplicación 145, otros módulos de programa 146, y los datos de programa 147. Nótese que estos componentes pueden ser iguales o diferentes del sistema operativo 134, los programas de aplicación 135, otros módulos de programa 136, y los datos de programa 137. El sistema operativo 144, los programas de aplicación 145, otros módulos de programa 146, y los datos de programa 147 le son dados diferentes números aquí para ilustrar que, como mínimo, ellas son diferentes copias. Un usuario puede ingresar con mandos de información en la computadora 110 a través de los dispositivos de entrada tal como una tableta (digitalizador electrónico) 164, un micrófono 163, un teclado 162 un dispositivo de puntero 161 comúnmente denominado como ratón, un "trackball, o un touch pad". Otros dispositivos de entrada (no mostrados) pueden incluir un "joystick", "game pad", un disco satelital, un escáner, o similar. Estos y otros dispositivos de entrada están a menudo conectados a la unidad e procesamiento 120 a través de una interfase de entrada de usuario 160 que está

acoplada al bus del sistema pero pueden estar conectadas por otras estructuras de interfase y de bus, tal como el puerto paralelo, el puerto de juegos o el bus serial universal (USB). Un monitor 191 u otro tipo de dispositivo de juego también esta conectado al bus del sistema 121 por vía de una interfase, tal como una interfase de video 190. el monitor 191 también puede estar integrado con un panel de pantalla de toque 193 o similar que puede ingresar entradas digitalizadas tales como escritura manual en el sistema de computadora 110 por vía de una interfase, tal como la interfase de pantalla de toque 192. Nótese que el monitor y/o el panel de pantalla de toque pueden estar físicamente acoplados a una carcasa en la cual el dispositivo de computo 110 está incorporado, tal como una computadora personal tipo tableta, en donde el panel de pantalla de toque 193 esencialmente sirve como una tableta 164. Además, las computadoras tales como el dispositivo de computo 110 también pueden incluir otros dispositivo de salida periféricos tales como los parlantes 195 y la impresora 196, que puede estar conectada a través de la interfase periférica de salida 194 o similar.

La computadora 110 puede operar en un ambiente de red utilizando conexiones lógicas a una o más computadoras remotas, tal como la computadora remoto 180. la computadora remoto 180 puede ser una computadora personal, un servidor, un agrupador, un PC de red, un dispositivo homólogo u otros

nodos de red común, y típicamente incluyen muchos o todos los elementos descritos anteriormente con relación a la computadora 110, aunque solamente un dispositivo de almacenamiento de memoria 181 ha sido ilustrado en la figura 1. Las conexiones lógicas descritas en la figura 1 incluyen una red de área local (LAN) 171 y una red de área amplia (WAN) 173, pero también se pueden incluir otras redes. Tales ambientes de red son sitios comunes en oficinas, redes de computadoras amplias de empresa, intranets y la Internet.

Cuando se utiliza en un ambiente de red LAN, la computadora 110 está conectada a la LAN 171 a través de la interfase o adaptador de red 170. Cando se utiliza en un ambiente de red WAN, la computadora 110 típicamente incluye un módem 172 u otros medios para estabilizar las comunicaciones sobre el WAN 173, tal como la Internet. El módem 172, que puede ser interno o externo puede estar conectado al bus del sistema 121 por vía de la interfase de entrada de usuarios 160 u otro mecanismo apropiado. En un ambiente de red, los módulos de programa descritos con relación a la computadora 110, o a porciones de está, se pueden almacenar en el dispositivo de almacenamiento de memoria remoto. Por vía de ejemplo, y no de limitación, la figura 1 ilustra programas de aplicación remota 185 como residentes en el dispositivo de memoria 181. Se apreciará que las conexiones mostradas son ejemplos y

otros medios de establecer un enlace de comunicaciones entre las computadoras se pueden utilizar.

CAPA DE INTEGRACIÓN MEDIA (MIL)

Un aspecto de la presente invención está dirigido de manera general a una arquitectura denominada como la capa de integración media (MIL) que incluyen una interfase de programación de aplicación de gráficos de modo inmediato (API), una estructura de datos de partición de pantalla y API que también juega un papel en las instrucciones de dibujos de cache, un conjunto de objetos de nivel de control, y un lenguaje demarcadores. En general, la arquitectura le permite al código de programa, tal como una aplicación o un componente del sistema operativo, comunicar instrucciones de dibujo y otra información (por ejemplo, mapas de bits de imágenes) a componentes gráficos con el fin de hacer representar la salida gráfica sobre la pantalla del sistema. Un aspecto de la presente invención suministra un número de funciones definidas y métodos, por ejemplo, en la forma de los API a un modelo de objeto, para posibilitarle a los programas poblar un gráfico de escena con estructuras de datos listas de instrucciones (primitivas de dibujo/comando), y otros datos relacionados con gráficos. Cuando se procesa, el gráfico de escena, da como resultado gráficos que son mostrados sobre la pantalla.

De acuerdo con un aspecto de la presente invención, el MIL es un sistema compuesto que trabaja en capas. Cada ítem hacia el compuesto es conceptualmente dibujado en un mapa de bits que tiene un canal alfa completo. El canal alfa es entonces utilizado cuando se compone el mapa de bits al buffer de la espalda. Los objetos individuales son compuestos desde atrás hacia el frente. Nótese que mientras este es un modelo conceptual, en la actualidad, el sistema entiende cuando una superficie de mapa de bits intermedia completa no es necesaria y se compone directamente al amortiguador de respaldo u otras superficies de respaldo. El sistema también puede entender cambios de incremento y hace un mínimo repintado.

La figura 2 representa una arquitectura general en capas (MIL) 200 en la cual la presente invención puede ser implementada. Como se representa en la figura 2, el código de programa 202 (por ejemplo un programa de aplicación o un componente de sistema operativo o similar) se puede desarrollar a los dato gráficos de salida de una o varias maneras, incluyendo por vía de imágenes 204, por vía de elementos gráficos de vector 206, y/o llamadas de método ubicadas directamente en una capa de interfase de programación de aplicación visual (API) 212. En general, las imágenes 204 suministran el código de programa 202 con un

mecanismo para cargar, editar y guardar imágenes; por ejemplo, el mapa de bits. Como se describe adelante, estas imágenes se pueden utilizar por otras partes del sistema, y existe también una manera de utilizar el código de dibujo primitivo para dibujar una imagen directamente. Los elementos gráficos de vector 206 suministran otra forma de dibujar gráficos, consistentes con el resto del modelo objeto (descrito adelante). Los elementos gráficos de vector 206 pueden ser creados por vía de un lenguaje de marcadores, cuyo elemento/sistema de propiedad 208 y sistema de diseño 210 interpretan para hacer llamadas apropiadas a la capa API visual 212. Los elementos gráficos de vector 206, junto con el elemento/sistema de propiedad 208 y el sistema de diseño 210, son descritos en la solicitud de patente comprendiente serie No. 10/401,717.

Así, el MIL 200 incluye diferentes niveles que un programador puede programar al incluir imágenes 204, la cual es la cadena de procesamiento para cargar, editar y guardar imágenes. Estas imágenes pueden ser utilizadas en el resto del sistema como se desee. Adicionalmente, existe una forma de utilizar el código de dibujo del primitivo para dibujar una imagen directamente. Otro nivel comprende el visual API 212, que es un API que primariamente suministra acceso a la estructura de datos 216 para organizar ítems a ser dibujados. Cada uno de

797

aquellos ítems se puede cargar con instrucciones de dibujo que pueden ser cacheadas por el sistema.

Existe una variedad de formas para especificar esta estructura de datos y que es dibujada dentro de una aplicación MIL-aware típica, este API se puede utilizar desde dentro de un sistema de diseño 210.

Un tercer nivel para programar comprende un nivel de elementos de gráficos de vector 206, que es un lenguaje de marcadores para describir y dibujar gráficos de una manera que sea consistente con el resto del control/modelo objeto Element. Los Vectors Graphics Elements exponen el sistema gráfico a través del sistema de elemento. Esto incluye un conjunto de elementos para representar y un conjunto de propiedades que trabajan sobre un elemento arbitrario. En una implementación, existen dos subconjuntos, que incluyen Element level Vector Graphics, que consiguen ser analizados sintácticamente y crear elementos, y gráficos de vector level Resource, que consiguen ser analizados sintácticamente y almacenados de una manera eficiente. El modelo objeto de nivel Element se refiere a operar al mundo de nivel mayor que trae en el árbol de elemento, el sistema de propiedad y el sistema de diseño 210. con respecto al análisis sintáctico, muchas propiedades dinámicas en el nivel Element son de los tipos MIL en general, el marcador es resuelto a los objetos,

476

798

en el cual el esquema XML para un marcador XAML que es visualmente declarado en la parte superior de un archivo marcador como sigue:

```
<Canvas xmlns="http://schemas.microsoft.com/2003/xaml"
xmlns:df="Definition"
def:Class="Surfin.ChannelBrowser"
def:CodeBehind="ChannelBrowser.xaml.cs"
ID="mainCanvas" Background="Black" Width="100%"
Height="100%"
Loaded="PageLoaded">
```

Cuando la etiqueta <Path> es utilizada por ejemplo, el analizador sintáctico utiliza el esquema de mirar el espacio del hombre relevante (por ejemplo, System.Windows.Shapes) para resolver y construir el objeto. Como generalmente se describió en la solicitud de patente copendiente anteriormente mencionada serie No 10/401,717, el analizador sintáctico confía el tipo de convertidores para convertir desde una cuerda a una instancia de un objeto MIL. Estos tipos que requieren sintaxis compleja tiene cada uno CLR escribible (tiempo de corrido de lenguaje común) propiedad expuesta como un atributo XLM opcional, que es analizado sintácticamente de la misma manera como una propiedad dinámica. Algunos tipos (notablemente Brushes) pueden ser analizados sintácticamente en cualquier forma simple o compleja.

:

798

Se debe notar que las llamadas de función dirigidas a cualquiera de estas capas o a cualquiera de estas clases se pueden manejar directamente o indirectamente. Por ejemplo, un manejador de solicitud puede comprender código intermedio que convierte las solicitudes recibidas en sistema operativo a llamadas API manejadas por otro sistema operativo. Así, como se utiliza aquí, las funciones son llamadas por programas que causan el comportamiento de la solicitud ocurra, sin importar de donde ocurre el procesamiento actual o donde son suministradas las estructuras de datos y las clases.

Como se entenderá, y también está representado en la figura 2, un sistema de animación 220 penetra el API completo. Como se describió aquí, agrupadores animados pueden ser pasados esencialmente a donde quiera, incluyendo en el nivel de propiedad del elemento 208, dentro del API visual 212 y en cualquiera de los otros recursos. El sistema de temporización también esta expuesto a ambos niveles de elemento y el visual.

En una implementación, la arquitectura de capa de gráficos 200 incluyen una composición de alto nivel y un motor de animación 214, que incluye o esta asociado de otra manera con una estructura datos de cache 216. La estructura de datos de cache 216 contiene un gráfico de escena que comprende objetos

800

dispuestos jerárquicamente que son manejados de acuerdo a un modelo objeto definido, como o se describe adelante. En general, la capa API visual 212 le suministra al código de programa 202 (el sistema de capa al sistema de diseño 210) on una interfase a la estructura de datos cache 216 incluyendo la capacidad de crear objetos, abrir y cerrar objetos para suministrarles datos, y así sucesivamente. En otras palabras, la composición de alto nivel y el motor de animación 214 exponen una capa API de medios unificado 212 por la cual los desarrolladores pueden expresar intenciones a cerca de los gráficos y medios para desplegar información de gráficos, y suministrar una plataforma subyacente con suficiente información de tal forma que la plataforma pueda optimizar el uso del equipo para el código de programa. Por ejemplo, la plataforma subyacente será responsable por la negociación por cachear, la negociación de recursos y la integración de medios.

En una implementación, la composición de alto nivel y el motor de animación 214 pasan una cuerda de instrucción y posiblemente otros datos (por ejemplo punteros a los mapas de bits) a una composición rápida, de bajo nivel y a un motor de animación 218. En general, la composición de bajo nivel y el motor de animación/representador 218, suministra un conjunto de sistemas que manejan el dibujo y la composición actuales sobre la pantalla. Nótese que como se utiliza aquí los

1576

términos "a alto nivel" y "bajo nivel" son similares a aquellos utilizados en otros escenarios de cómputo, en donde en general, entre más bajo es el componente de software con relación a los componentes más altos, más cercano es ese componente al hardware. Así, por ejemplo, la información de gráficos enviada a la composición de alto nivel y al motor de animación 214 puede ser recibida en el motor de composición y animación de bajo nivel 218, donde la información es utilizada para enviar datos de gráficos al subsistema de gráficos que incluye el hardware 222. Nótese que la presente invención puede ser extendida a una composición multinivel más haya de dos capas.

Adicionalmente, nótese que con el fin de facilitar la resolución y las interfases de usuario dependientes de dispositivo, el concepto de un píxel no es expuesto como una unidad fundamental en los API principales. En su lugar, el sistema de coordenadas inicial es ajustado de tal forma que una unidad en el sistema de coordenadas es igual a $1/96^{\circ}$ de cada pulgada. Esto puede ser denominado como un "dip", o alternativamente un px, aunque en algunos sistemas (tal como el monitor o una impresora de alta resolución) este no mapee a un píxel. En longitud, la unidad dip es traducida directamente a una unidad de usuario. El multiplicador entre otras unidades físicas (pulgada, cm, pt, y similares) es asegurado en $1/96^{\circ}$ de pulgada. Esto significa que si una

882

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América
- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

☒ Solicitantes adicionales y/o inventores (adicionales) se indican en una hoja a continuación.

Formulario PCT/RO/101 (hoja de continuación) (Enero 2004)

Ver notas al formulario de solicitud

391

Hoja No. 3

Continuación de la Casilla No. III

SOLICITANTE(ES) Y/O INVENTOR(ES) ADICIONAL(ES)

Si ninguna de las siguientes subcasillas se usa, esta hoja no debe ser incluida en la solicitud.

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).

**Schechter, Greg D.
1127 37th Avenue East
Seattle, Washington 98112
US**

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)

801

Estado (que es, el país) de nacionalidad:

US

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América
- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).*

Smith, Adam M..

12310 NE 92 Street #205

Kirkland, Washington 98033

US

313

205

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor *(si esta casilla de verificación es marcada, no llene adelante)*

Estado (que es, el país) de nacionalidad:

US

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América
- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de*

2201

806

la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).

Vandenberg, Eric S.
1419 2nd Avenue W #201
Seattle, Washington 98119
US

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)

Estado (que es, el país) de nacionalidad:

US

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados

371

303

- ☐ Todos los estados designados excepto los Estados Unidos de América
- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).*

**Calkins, Matthew W.
503 E Thomas Street #204
Seattle, Washington 98102
US**

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor *(si esta casilla de verificación es marcada, no llene adelante)*

376

880

Estado (que es, el país) de nacionalidad:

US

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América
- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

☒ Solicitantes adicionales y/o inventores (adicionales) se indican en una hoja a continuación.

Formulario PCT/RO/101 (hoja de continuación) (Enero 2004)

Ver notas al formulario de solicitud

347

809

Hoja No. 4

Continuación de la Casilla No. III**SOLICITANTE(ES) Y/O INVENTOR(ES) ADICIONAL(ES)**

Si ninguna de las siguientes subcasillas se usa, esta hoja no debe ser incluida en la solicitud.

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).

**Gallo, Kevin T.
19235 222nd Way NE
Woodinville, Washington 98072
US**

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor *(si esta casilla de verificación es marcada, no llene adelante)*

398

910

Estado (que es, el país) de nacionalidad:

US

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América
- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).*

**Stokes, Michael
1946 E. Stonybrook Court
Eagle, Idaho 83616
US**

319

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)

Estado (que es, el país) de nacionalidad:

US

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América
- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del

400

solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).

Goel, Rajat
121 Vine St. #2206
Seattle, Washington 98121
US

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)

Estado (que es, el país) de nacionalidad:

Canadá

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América

201

813

☐ Solamente los Estados Unidos de América

☐ Los estados indicados en la casilla complementaria

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).

Esta persona es:

☐ Solo solicitante

☐ Solicitante e inventor

☐ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)

Estado (que es, el país) de nacionalidad:

Estado (que es, el país) de residencia:

Esta persona es solicitante para los propósitos de:

402

- ☐ Todos los estados designados
 - ☐ Todos los estados designados excepto los Estados Unidos de América
 - ☐ Solamente los Estados Unidos de América
 - ☐ Los estados indicados en la casilla complementaria
-
- ☐ Solicitantes adicionales y/o inventores (adicionales) se indican en una hoja a continuación.

Formulario PCT/RO/101 (hoja de continuación) (Enero 2004)

Ver notas al formulario de solicitud

Hoja No. 5 .

Casilla No. V DESIGNACIONES

El registro de esta solicitud constituye de acuerdo con la Regla 4,9(a), la designación de todos los Estados Contratantes unidos por el TLC en la fecha de presentación internacional, para el otorgamiento de cada clase de protección disponible y, cuando aplique, para el otorgamiento de ambas patentes naciones y regional.

Sin embargo,

☐ DE Alemania no es designado para cualquier clase de protección nacional

☐ KR República de Corea no es designado para cualquier clase de protección nacional

☐ RU Rusia no es designado para cualquier clase de protección nacional

(Las casillas marcadas arriba pueden ser usadas para excluir (irrevocablemente) las designaciones

for

concernientes con el fin de evitar el cese del efecto, de acuerdo con la ley nacional, de una solicitud nacional anterior de la cual se reivindica prioridad. Ver las notas a la casilla No. V así como las consecuencias de tales provisiones de la ley nacional en estos y ciertos otros estados.)

Casilla No. VI REIVINDICACIÓN DE PRIORIDAD

☐ Por medio de la presente se reivindica la prioridad de la(s) siguiente(s) solicitud(es) anterior(es):

Fecha de presentación de la solicitud anterior (día/mes/año)	Número de la solicitud anterior	En donde está la solicitud anterior es:		
		Solicitud nacional: país o miembro de WTO	Solicitud regional:* Oficina regional	Solicitud internacional: Oficina receptora
Ítem (1) 23 Octubre 2003	10/693,630	US		
Ítem (2)				
Ítem (3)				

☐ Reivindicaciones de prioridad adicionales se indican en la casilla Complementaria.

Se solicita a la Oficina receptora preparar y transmitir a la Oficina Internacional una copia certificada de la(s) solicitud(es) anterior(es) *(solo si la solicitud anterior fue presentada con la Oficina para propósitos de esta solicitud internacional es la Oficina receptora)* identificada arriba como:

- ☒ Todos los ítems
- ☐ ítem (1)
- ☐ ítem (2)
- ☐ ítem (3)
- ☐ Otros, ver casilla complementaria

* Cuando la solicitud anterior es una solicitud ARIPO, es obligatorio indicar en la casilla complementaria al menos un estado parte de la Convención de París para la Protección de Propiedad Industrial o un Miembro de la Organización Mundial del Comercio para el cual se presentó la solicitud anterior (Regla 4.10(b)(ii)).

Casilla No. VII AUTORIDAD DE BÚSQUEDA INTERNACIONAL

206

818

Elección de la Autoridad de Búsqueda Internacional (ISA)
(si dos o más Autoridades de Búsqueda Internacionales son competentes para llevar a cabo la búsqueda internacional, indique la autoridad escogida; se puede utilizar el código de dos letras):

ISA/ US

Solicitud para utilizar los resultados de la búsqueda anterior; Investigación de tal búsqueda (si una búsqueda anterior se ha llevado a cabo por o solicitado de la Autoridad de Búsqueda Internacional):

Fecha (día/mes/año):

Número:

País (u Oficina Regional):

Casilla No. VIII DECLARACIONES

Las siguientes **declaraciones** están contenidas en las casillas Nos. VIII (i) a (v) (marque las casillas aplicables adelante e indique en la columna de la derecha el número de cada tipo de declaración):

407

Número de
declaraciones

☐ Casilla No. VIII (i) Declaración en cuanto a la identidad del inventor :	
☐ Casilla No. VIII (ii) Declaración en cuanto al derecho del solicitante, como en la fecha de registro, para aplicar para y otorgarse una patente:	
☐ Casilla No. VIII (iii) Declaración en cuanto al derecho del solicitante, como en la fecha de registro, para reivindicar la solicitud de la reivindicación anterior:	
☐ Casilla No. VIII (iv) Declaración de inventoria (solo para propósitos de la designación de los Estados Unidos de América):	
☐ Casilla No. VIII (v) Declaración en cuanto a descripción no perjudicial o excepciones a falta de novedad:	

Formulario PCT/RO/101 (segunda hoja) (Enero 2004)

Ver notas al formulario de solicitud

810

Hoja No. 6

Casilla No. IX

LISTA DE VERIFICACIÓN; IDIOMA DE PRESENTACIÓN

Esta solicitud internacional contiene:

(a) en formulario de papel el siguiente número de hojas:

Petición (incluye hojas de declaración): 6

Descripción (que excluye listado de
Secuencias y/o tablas relacionadas

Con esta) : 323

Reivindicaciones : 9

Resumen : 1

Dibujos : 29

Subtotal del número de hojas : 368

listado de secuencias :

tablas relacionadas con esta :

(para ambos, número actual de hojas si se registró en
formulario de papel, sea o no también registrado en forma
legible por computador; ver (c) adelante) _____

Número total de hojas : 368

209

gu

(b) ☐ solo en forma legible en computador

(Sección 801(a)(i))

(i) ☐ lista de secuencias

(ii) ☐ tablas relacionadas con estas

(c) ☐ también en forma legible en computador

(Sección 801(a)(ii))

(i) ☐ lista de secuencias

(ii) ☐ tablas relacionadas con estas

Tipo y número de portadores (Diskette, CD-ROM, CD-Ru
otro) en el cual está contenida la

☐ Lista de secuencias

☐ Tablas relacionadas con esta

(copias adicionales a ser indicaras de acuerdo con los
ítems 9(ii) y/o 10(ii), en la columna de la derecha)

Figura de los dibujos que debe acompañar el resumen: 2

410

022

Esta solicitud internacional está acompañada por los siguientes ítems (marque las casillas que apliquen adelante e indique en la columna de la derecha el número de cada ítem):

	No. De Ítems
1. ☒ hoja de cálculo de tarifas	: 1
2. ☐ poder de abogado separado original	:
3. ☐ poder general de abogado	:
4. ☒ copia del poder general de abogado; número de referencia si existe:	: 1
5. ☐ declaración que explica la falta de firma	
6. ☐ documento(s) de prioridad identificado en la Casilla VI como ítem(s)	:
7. ☐ traducción de la solicitud internacional en (idioma)	:
8. ☐ indicaciones separadas relacionadas con microorganismos depositados u otro material biológico	
9. ☐ listado de secuencias en forma legible por computador (indique tipo y número de portadores)	:

411

0.13

- (i) ☐ copia presentada para propósitos de búsqueda internacional de acuerdo solo con la Regla 13ter (y no como parte de la solicitud internacional) :
- (ii) ☐ (solo cuando una casilla de verificación (b) (i) o (c) (i) sea marcada en la columna izquierda) copias adicionales incluyen, donde aplique, la copia para los propósitos de búsqueda internacional de acuerdo con la Regla 13ter :
- (iii) ☐ junto a declaración relevante en cuanto a la identidad de la copia o copias con la lista de secuencias mencionadas en la columna de la izquierda :

10. ☐ Tablas en forma legible por computador relacionadas con la lista de secuencias (indique tipo y número de portadores) :

- (i) ☐ copia presentada para propósitos de búsqueda internacional de acuerdo solo con la Sección 802 (b-cuarto) (y no como parte de la solicitud internacional) :

4(2)

824

(ii) ☐ (solo cuando una casilla de verificación (b)(ii) o (c)(ii) sea marcada en la columna izquierda) copias adicionales incluyen, donde aplique, la copia para los propósitos de búsqueda internacional de acuerdo con la Sección 802(b-cuarto) :

(iii) ☐ junto a declaración relevante en cuanto a la identidad de la copia o copias con las tablas mencionadas en la columna de la izquierda :

11. ☒ Otro (especifique): estampilla de vuelta de correo : 1

Idioma de presentación de

la solicitud internacional

Inglés

Casilla No. IX

FIRMA DEL SOLICITANTE, O AGENTE O REPRESENTANTE COMÚN

413

025

Junto a cada firma, indique el nombre de la persona firmante y la capacidad en la cual la persona firma (si tal capacidad no es obvia de la lectura de la solicitud)

(Aparece Firma ilegible)

Albert S. Michalk, Agente

Para uso exclusivo de la Oficina Receptora

1. Fecha de recibo actual de la solicitud internacional propuesta:
2. Dibujos: ☐ Recibido:
☐ No recibido:
3. Fecha corregida de recibo actual debido a posterior pero oportuno recibo de documentos o dibujos que completan la solicitud internacional propuesta:
4. Fecha de recepción oportuna de las correcciones requeridas de acuerdo con el Artículo 11(2) PCT:
5. Autoridad de Búsqueda Internacional
(si dos o más son competentes): ISA/
6. ☐ Transmisión de la copia de búsqueda retrasada hasta que la tasa de búsqueda se pague

454

Para Uso de la Oficina Internacional solamente

Fecha de recibo de la copia de registro por la Oficina Internacional:

Formulario PCT/RO/101 (última Hoja) (Enero 2004)

Ver notas al formulario de solicitud

Esta hoja no hace parte de y no cuenta como una hoja de la solicitud internacional

PCT

HOJA DE CÁLCULO DE TARIFAS

Anexo a la Solicitud

Archivo de referencia del agente o solicitante

4121

Solicitante

MICROSOFT CORPORATION

Para uso solo de la Oficina receptora

Solicitud Internacional No. _____

Estampilla de fecha de la Oficina Receptora _____

827

CÁLCULO DE LAS TARIFAS PRESCRITAS

1. TARIFA DE TRANSMISIÓN \$300,00 T

2. TARIFA DE BÚSQUEDA \$300,00 S

Reporte de Búsqueda a ser llevado a cabo por _____

(Si dos o más Autoridades de Búsqueda Internacional son competentes para llevar a cabo el reporte de búsqueda internacional, indique el nombre de la autoridad que se escoge para llevar a cabo la búsqueda internacional.)

3. TARIFA DE REGISTRO INTERNACIONAL

Cuando los ítems (b) y/o (c) de la Casilla No. IX aplica, entre el **Número subtotal de Hojas**

Cuando los ítems (b) y (c) de la Casilla No. IX no aplica, entre el **Número total de Hojas**

368

i1	Primeras 30 hojas	\$1134.00	i1

i2	<u>338</u>	x	<u>\$12.00</u>	=	\$4056.00	i2

No. De hojas	Tarifa por hoja
en exceso de 30	

i3

416

827

Componente adicional (solo si la lista de secuencia y/o tablas relacionadas con esta están registradas en forma legible en computador de acuerdo con la Sección 801(a)(i), o ambos en tal forma y en papel, de acuerdo con la Sección 801(a)(ii)):

$$400 \times \underline{\hspace{2cm}} = \boxed{i3}$$

Tarifa por hoja

Sume las cantidades entradas en i1, i2, e i3 e ingrese el total en I I

(Los solicitantes de ciertos Estados tienen derecho a una reducción de 75% de la tarifa de presentación internacional. Cuando el solicitante tiene derecho (o todos los solicitantes tienen), el total a ser ingresado en I es 25% de la tarifa de presentación internacional.)

4. Tarifa para documento de prioridad

(si aplica)

\$20.00

P

5. TOTAL DE TARIFAS PAGABLES

417

gty

Sume las cantidades ingresadas en T, S; I y P e ingrese el total en la casilla

\$5.810,00

TOTAL

MODO DE PAGO

- ☐ Autorización a cargar a cuenta de depósito (ver adelante)
- ☐ Giro postal
- ☐ Efectivo
- ☐ Cupones
- ☐ Cheque
- ☐ Giro bancario
- ☐ Timbre fiscal
- ☒ Otro (especifique): **Pago con Tarjeta de Crédito**

AUTORIZACIÓN A CARGAR (O ACREDITAR) CUENTA DE DEPÓSITO

- ☐ Autorización a cargar las tarifas totales indicadas anteriormente.
- ☒ (esta casilla de verificación puede ser marcada solo si las condiciones para las cuentas de depósito de la

418

80

Oficina receptora así lo permita) Autorización a cargar cualquier deficiencia o acreditar cualquier sobrepago en el total de tarifas indicado anteriormente

☐ Autorización para cargar la tarifa para documento de prioridad.

Oficina Receptora:	RO/ US
Cuenta de Depósito No.:	50-1618
Fecha:	Julio 28, 2004
Nombre:	Albert S. Michalk
Firma:	(aparece firma ilegible)

Formulario PCT/RO/101 (Anexo) (Enero 2004)

Ver notas a la hoja de cálculo de tarifas

419

PCT**PODER GENERAL DE ABOGADO**

**(para varias solicitudes internacionales presentadas de
acuerdo con el Tratado de Cooperación de Patentes)**

(Regla 90.5 PCT)

La(s) persona(s) abajo firmante(s):

*(apellido seguido por nombre: para una entidad legal,
designación oficial completa. La dirección debe incluir
código postal y nombre del país)*

Microsoft Corporation

One Microsoft Way

Redmond, Washington, USA 98052-6399

Por medio de la presente señalamos a la siguiente persona
como:

☒ Agente ☐ Representante común

Nombre y dirección

*(apellido seguido por nombre: para una entidad legal,
designación oficial completa. La dirección debe incluir
código postal y nombre del país)*

832

MICHALIK, Albert S. (Reg. No. 37,395); WYLIE, Roger D.
(Reg. No. 36,974); whitaker, John E. (Reg. No. 42,222)

MICHALIK& WYLIE, PLLC
14645 Bel-Red Road
Suite 103
Redmond, Washington USA 98007

Para representar el abajo firmante ante

- ☒ Todas las Autoridades internacionales competentes
☐ Solo la Autoridad de Búsqueda Internacional
☐ Solo la Autoridad Examinadora Preliminar Internacional

Con relación con cualquiera y todas las solicitudes
internacionales presentadas por el abajo firmante con la
siguiente Oficina

Oficina de Marcas y Patentes de los Estados Unidos

como oficina receptora y para hacer o recibir pagos en
nombre del abajo firmante.

421

03

Firma(s) (cuando haya varias personas cada una de ellas debe firmar; junto a cada firma, indicar el nombre de la persona firmante y la capacidad en la cual la persona firma, si tal capacidad no es obvia de la lectura de este poder).

Microsoft Corporation

Por: (aparece firma ilegible)

KATE SAKO

Título: **SECRETARIA ASISTENTE**

Fecha: **2/29/2000**

422

834

TRATADO DE COOPERACIÓN DE PATENTES**PCT****NOTIFICACIÓN RELACIONADA CON EL PAGO DE TARIFAS****PRESCRITAS****(Reglas 14, 15 y 16 e Instrucciones administrativas,****Secciones 102bis(c), 304, 323(b), 707(b) y 803)****De la OFICINA RECEPTORA****A:****ALBERT S. MICHALIK****704-228TH AVENUE NE, SUITE 193****SAMMAMISH, WASHINGTON 98074****25-APP****ACCT 306233.26****PROCESAMIENTO****Fecha de Correspondencia (Día/mes/año)****29 Agosto 2004****Referencia de archivo del agente o solicitante****4121****Pago debido****Ver ítem 3 para límites de tiempo**

428

05

Solicitud Internacional No.

PCT/US2004/024570

Fecha de Presentación Internacional/Fecha de Recibo

(Día/mes/año)

28 Julio 2004

Fecha de la Prioridad (Día/mes/año)

23 Octubre 2003

Solicitante

Microsoft Corporation

1. Por medio de la presente se notifica al solicitante
que esta Oficina receptora ha recibido:

- ☒ **El pago de todas las tarifas prescritas**
- ☐ **Un sobrepago que será devuelto en forma debida.**
- ☐ **Ningún pago o pago insuficiente de las tarifas**
prescritas y por medio de la presente **se invita a**
pagar el debido balance, como se resumió en el ítem 2,
dentro del (los) límite(s) de tiempo indicado(s) en el
ítem 3.

2. Cálculo de tarifas y pago

42-1

026

<u>5.810,00</u>	-	<u>5.810,00</u>	=	0,00
Total tarifas pagable		cantidad pagada		Balance

☐ Los detalles del cálculo se dan en el anexo

3. Límite(s) de tiempo y cantidad(es) pagables (Regla 14.1, 15.4 y 16.1 (f)):

☐ dentro de UN MES a partir de la fecha de recibo de la solicitud internacional (para la tarifa de transmisión (si existe), la tarifa de búsqueda y la tarifa de presentación internacional). La cantidad pagable para cada tarifa es la cantidad aplicable en la fecha de recibo de la solicitud internacional.

☐ dentro de 16 MESES a partir de la fecha de la prioridad (solo para la tarifa para documento de prioridad). Se dirige la atención del solicitante la hecho de que se hace la solicitud por el solicitante de acuerdo con la Regla 17.1(b) no será considerado a menos que se la tarifa sea pagada dentro de tal límite de tiempo.

4. Observaciones adicionales (si es necesario)

42

837

☐ la copia de búsqueda no será transmitida a la Autoridad de Búsqueda Internacional hasta que la tarifa de búsqueda sea pagada (por lo tanto se retrasará el inicio de la búsqueda internacional) (Regla 23.1(a) y (b)).

Nombre y dirección de correspondencia de la Oficina receptora

Mail Stop PCT, Commissioner for Patents
P.O. Box 1450, Alexandria, VA 22313-1450
Facsímile No. 703-305-3230

Oficial Autorizado

Nhu Tuy Tran

Teléfono No. 703-605-1537

428

032

ANEXO AL FORMULARIO PCT/RO/102

CÁLCULO DE TARIFAS PRESCRITAS

Solicitud Internacional No.

PCT/US2004/024570

☐ T Tarifa de TransmisiónCantidad prescrita: 300.00 ☐ T ☒ Cantidad correctaCantidad pagada: 300.00 ☐ SobrepagoBalance: 0.00 ☐ Balance debido☐ S Tarifa de BúsquedaCantidad prescrita: 300.00 ☐ S ☒ Cantidad correctaCantidad pagada: 300.00 ☐ SobrepagoBalance: 0.00 ☐ Balance debido

Tarifa de Presentación Internacional

Cantidad fijada para las primeras 30 hojas 1,134.000 ☐ i1338 x 12.00 4.056,00 ☐ i2Número de hojas Tarifa x
En exceso de 30 Hoja

424.

839

Componente adicional

400 x 0.00 i3

Tarifa por hoja

Reducción cuando la solicitud internacional se registra
(ver Guía del Solicitante PCT, Volumen I, Parte General,
para detalles sobre la disponibilidad de esta reducción):

Utilizando el software PCT-EASY - 0,00 r

O

En forma electrónica cuando el texto de la descripción,
reivindicaciones y resumen no está en formato codificado
de carácter - 0,00 r

O

En forma electrónica cuando el texto de la descripción,
reivindicaciones y resumen está en formato codificado de
carácter - 0,00 r

Subtotal = 5.190,00 i1+i2+i3-r

Cantidad total prescrita (La cantidad a ser ingresada en
1 es el subtotal ingresado en (i1+i2+i3-r), excepto
cuando el solicitante tiene derecho (o todas las
solicitudes tienen) a una reducción del 75%, en cuyo caso
la cantidad a ser ingresada en 1 es 25% del subtotal

422

840

(i1+i2+i3-r); ciertas solicitudes de ciertos estados tienen derecho a una reducción del 75% de la tarifa de presentación internacional; ver notas a la hoja de cálculo de tarifas; ver Notas a la Hoja de Cálculo de Tarifas como se anexó al formulario de solicitud, PCT/RO/101, para detalles): = 5.190,00 ☒ I

Cantidad pagada: - 5.190,00

Balance: = 0,00

☒ Cantidad correcta

☐ Sobre pago

☐ Balance debido

☒ **Tarifa para documento de prioridad**

Cantidad prescrita: 20,00 ☒ ☒ Cantidad correcta

Cantidad pagada: 20,00 ☐ Sobre pago

Balance: 0,00 ☐ Balance debido

429

TRATADO DE COOPERACIÓN DE PATENTES**PCT****NOTIFICACIÓN DEL NÚMERO DE SOLICITUD INTERNACIONAL Y DE****LA FECHA DE PRESENTACIÓN INTERNACIONAL****(Regla 20.5(C) PCT)****De la OFICINA RECEPTORA****A:****ALBERT S. MICHALIK****704-228TH AVENUE NE, SUITE 193****SAMMAMISH, WASHINGTON 98074****Fecha de Correspondencia (Día/mes/año)****24 Agosto 2004****Referencia de archivo del agente o solicitante****4121****NOTIFICACIÓN IMPORTANTE****Solicitud Internacional No.****PCT/US2004/024570****Fecha de Presentación Internacional/Fecha de Recibo****(Día/mes/año)****28 Julio 2004**

giz

Fecha de la Prioridad (Día/mes/año)

23 Octubre 2003

Solicitante

MICROSOFT CORPORATION

Título

CAPA DE INTEGRACIÓN DE MEDIOS

1. Por medio de la presente se notifica al solicitante que la solicitud ha estado acorde con el número de solicitud internacional y la fecha de presentación internacional indicada anteriormente.
2. se notifica al solicitante además que la copia de registro de la solicitud internacional:

☒ fue transmitida a la oficina Internacional en

24 Agosto 2004

☐ Aún no ha sido transmitida a la oficina internacional por la razón indicada adelante y una copia de esta notificación ha sido enviada a la Oficina Internacional*:

☐ ya que no se ha obtenido todavía la claridad de seguridad nacional necesaria

☐ debido a (razón a ser especificada)

437

* La Oficina Internacional monitorea la transmisión de la copia de registro por la Oficina receptora y notificará al solicitante (con Formulario PCT/IB/301) de su recibo. La copia de registro no debe haber sido recibida al término de la expiración de 14 meses a partir de la fecha de prioridad, la Oficina Internacional notificará al solicitante (Regla 22.1(c)).

3. INFORMACIÓN DE LICENCIA DE TRANSMISIÓN EXTRANJERA

Terminado por: NT

☐ Licencia adicional para transmisión extranjera no requerida. Esta materia objeto está cubierta por una licencia ya otorgada o el equivalente a solicitud nacional U.S. Refiérase a esa licencia para información relacionada con su alcance.

☐ Licencia para transmisión extranjera no requerida. 37 CFR. 5.11(e)(1) o 37 CFR. 5.11(e)(2). Sin embargo, se puede requerir una licencia para materia objeto adicional. Ver 37 CFR. 5.15(b).

☒ Otorgada licencia de transmisión extranjera. 35 U.S.C. 184; 37 CFR 5.11 el 19 Agosto 2004:

844

☒ 37 CFR 5.15(a)☐ 37 CFR 5.15(b)

Nombre y dirección de correspondencia de la oficina
receptora

Mail Stop PCT, Commisioner for Patents
P.O. Box 1450, Alexandria, VA 22313-1450
Facsimile No. 703-305-3230

Oficial Autorizado

Nhu Tuy Tran

Teléfono No. 703-605-1537

Formulario PCT/RO/105 (Julio 1992)

TRATADO DE COOPERACIÓN DE PATENTES

PCT

NOTIFICACIÓN DE RECIBO DE COPIA DE BÚSQUEDA

(Regla 25.1 PCT)

De la AUTORIDAD DE BÚSQUEDA INTERNACIONAL

537

645

A:

ALBERT S. MICHALIK

704-228TH AVENUE NE, SUITE 193

SAMMAMISH, WASHINGTON 98074

Fecha de Correspondencia (Día/mes/año)

24 Agosto 2004

Referencia de archivo del agente o solicitante

4121

NOTIFICACIÓN IMPORTANTE

Solicitud Internacional No.

PCT/US2004/024570

Fecha de Presentación Internacional/Fecha de Recibo
(Día/mes/año)

28 Julio 2004

Fecha de la Prioridad (Día/mes/año)

23 Octubre 2003

Solicitante

MICROSOFT CORPORATION

1. Cuando la Autoridad de Búsqueda Internacional y la
Oficina receptora no son la misma Oficina:

Por medio de la presente se notifica al solicitante
que la copia de búsqueda de la solicitud

587

internacional fue recibida por esta Autoridad de Búsqueda Internacional en la fecha indicada adelante.

Cuando la Autoridad de Búsqueda Internacional y la Oficina receptora son la misma Oficina:

Por medio de la presente se notifica al solicitante que la copia de búsqueda de la solicitud internacional fue recibida en la fecha indicada adelante

24 Agosto 2004

(Fecha de recibo)

2. ☐ la copia de búsqueda estuvo acompañada por una lista de secuencia de ácido amino y/o nucleótido o tablas relacionadas con estas en forma legible por computador
3. **Límite de tiempo para establecimiento de reporte de búsqueda internacional y concepto escrito de la Autoridad de Búsqueda Internacional.** Se informa al solicitante que el límite de tiempo para establecer el reporte de búsqueda internacional y el concepto escrito de la Autoridad de Búsqueda Internacional es tres meses a. partir de la fecha de recibo

247

indicada anteriormente o nueve meses a partir de la fecha de prioridad, cualquier límite de tiempo que expire después. (Reglas 42.1 y 43bis.1(a))

4. Una copia de esta notificación ha sido enviada a la Oficina Internacional y, cuando la primera oración del párrafo 1 aplica, a la Oficina receptora.

Nombre y dirección de correspondencia del ISA/

Mail Stop PCT, Commissioner for Patents

P.O. Box 1450, Alexandria, VA 22313-1450

Facsímile No. 703-305-3230

Oficial Autorizado

Nhu Tuy Tran

Teléfono No. 703-605-1537

Formulario PCT/ISA/202 (Enero 2004)

231

847

TRATADO DE COOPERACIÓN DE PATENTES**PCT****NOTIFICACIÓN DE RECIBO DE COPIA DE REGISTRO**

(Regla 24.2(a))

306.233.26

De la OFICINA INTERNACIONAL

A:**MICHALIK, Albert S.**

Suite 193

704-228th Avenue NE,

Sammamish, WA 98074

Estados Unidos de América

Fecha de Correspondencia (Día/mes/año)

22 Septiembre 2004 (22.09.2004)

Referencia de archivo del agente o solicitante

4121**NOTIFICACIÓN IMPORTANTE**

Solicitud Internacional No.

PCT/US2004/024570

430

849

Por medio de la presente se notifica al solicitante que la Oficina Internacional ha recibido la copia de registro de la solicitud internacional como se detalla adelante.

Nombre(s) del solicitante(s) y Estado(s) para el cual ellos son solicitantes:

MICROSOFT CORPORATION (para todos los Estados excepto US)

Fecha de Presentación Internacional:

28 Julio 2004 (28.07.2004)

Fecha de prioridad reivindicada:

23 Octubre 2003 (23.10.2003)

Fecha de recibo de la copia de registro por la Oficina Internacional:

30 Agosto 2004 (30.08.2004)

Lista de Oficinas designadas:

AP: BW, GH, GM, KE, LS, LW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW

EA: AM, AZ, BY, KG, KZ, MD, RU, TJ, TM

436

EP:

AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, HU, IE, IT, LU, MC, NL,
PL, PT, RO, SE, SI, SK, TR

OA: BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG

Nacional:

AE, AG, AL, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU,
CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL,
IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK,
MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG,
SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW

ATENCIÓN

El solicitante debe revisar cuidadosamente los datos que aparecen en esta Notificación. En caso de cualquier discrepancia entre estos datos y las indicaciones en la solicitud internacional, el solicitante debe informar inmediatamente a la Oficina Internacional.

Adicionalmente, se dirige la atención del solicitante a la información contenida en el Anexo, con relación a:

851

☒ límites de tiempo para la entrada en la fase nacional -
ver información importante actualizada (en fecha de Abril
2002)

☐ requerimientos con respecto a los documentos de
prioridad (si aplica)

Una copia de esta Notificación está siendo enviada a la
Oficina receptora y a la Autoridad de Búsqueda
Internacional.

La Oficina Internacional de WIPO
34, chemin des Colombettes
1211 Ginebra 20, Suiza

Facsímile No. (41-22) 388.89.75

Oficial Autorizado:

A. ZOLTANSKI (Fax: 338 89 75)

Teléfono No. (41-22) 338 8608

Formulario PCT/IB/301 (Enero 2004)

006439906

ucl

852

ANEXO AL FORMULARIO PCT/IB/301

Solicitud Internacional No.

PCT/US2004/024570

**INFORMACIÓN SOBRE LOS LÍMITES DE TIEMPO PARA ENTRAR EN LA
FASE NACIONAL**

Se le recuerda al solicitante que **debe** entrar a la "fase nacional" antes de cada una de las Oficinas designadas indicadas en la portada de esta Notificación mediante el pago de tarifas nacionales y suministro de traducciones, como está prescrito por los Artículos 22 y 39 y las leyes nacionales aplicables. Adicionalmente, el solicitante también puede cumplir con otros requerimientos especiales aplicables en ciertas oficinas. Es de responsabilidad del solicitante asegurar los pasos necesarios se tomen en una forma oportuna para entrar a la fase nacional. La mayoría de las Oficinas no publican recordatorios a los solicitantes en relación con la entrada en la fase nacional.

El límite de tiempo aplicable para entrar a la fase nacional, sujeto a lo que se dice en el siguiente

441

853

párrafo, será de 30 MESES a partir de la fecha de la prioridad, no solo con respecto a cualquier Oficina elegida si una petición para examen preliminar internacional se registra antes de la expiración de 19 meses a partir de la fecha de la prioridad (ver Artículo 39(1)), pero también con respecto de cualquier Oficina designada, en la ausencia de registro de tal petición, donde el Artículo 22(1) como se modificó con efecto de 1 Abril 2002 aplica con respecto a tal oficina designada. Para detalles adicionales, ver Gaceta PCT No. 44/2001 de 1 Noviembre 2001, páginas 19926, 19932 y 19934, así como también las publicaciones del Diario PCT, Octubre y Noviembre 2001 y Febrero 2002.

En la práctica, los límites de tiempo diferentes al límite de tiempo de 30 meses continuarán aplicándose, durante varios periodos de tiempo, con respecto a ciertas Oficinas elegidas o designadas. Para actualizaciones regulares sobre los límites de tiempo aplicables (20, 21, 30 o 31 meses, u otro límite de tiempo), Oficina por Oficina, refiérase a la Gaceta PCT ("Sección IV" parte publicada sobre una base semanal), al Diario PCT (sobre una base mensual), y a los Capítulos Nacionales relevantes en el Volumen II de la Guía del Solicitante

442

859

```

bool IsPaused { get; }
bool IsReversed { get; }
double Progress { get; }
bool IsChanging { get; }
bool IsEnabled { get; }
bool IsOverridingBaseValue { get; }
bool HasChanged { get; }

void Enable();
void Enable(Timeline parentTimeline);
void Disable();

void BeginIn(Time offset);
void EndIn(Time offset);
void Pause();
void Resume();
void Seek(Time offset, TimeSeekOrigin origin);

event EventHandler Begun { add; remove; }
event EventHandler Ended { add; remove; }
event EventHandler Paused { add; remove; }
event EventHandler Repeated { add; remove; }
event EventHandler Resumed { add; remove; }
event EventHandler Reversed { add; remove; }
event EventHandler Searched { add; remove; }
}

```

The **AudioData** object is provided for audio data:

```

public System.Windows.Media.AudioData : MediaData
{
    public AudioData (string source);
}

```

335

055

```

//
// MediaData
//
public MediaState State { get; }
public Time MediaDuration { get; }
public bool HasAudio { get; }
public bool HasVideo { get; }
public long Width { get; }
public long Height { get; }
public double Volume { get; set; }
public bool Mute { get; set; }

public void Play();

public void Dispose();

public bool IsChangeable { get; } // defaults to true
public Changeable Copy();
public void MakeUnchangeable();
public UseStatus StatusOfNextUse { get; set; } // defaults to false
public event EventHandler Changed { add; remove; }
public UIContext UIContext { get; }

double Acceleration { get; set; }
bool AutoReverse { get; set; }
TimeSyncValue Begin { get; set; }
double Deceleration { get; set; }
Time Duration { get; set; }
TimeSyncValue End { get; set; }
TimeEndSync EndSync { get; set; }
TimeFill Fill { get; set; }
TimeFill FillDefault { get; set; }

```

```

Timeline ParentTimeline { get; set; }
double RepeatCount { get; set; }
Time RepeatDuration { get; set; }
TimeRestart Restart { get; set; }
TimeRestart RestartDefault { get; set; }
double Speed { get; set; }
int CurrentRepeat { get; }
Time CurrentTime { get; }
bool IsForwardProgressing { get; }
bool IsPaused { get; }
bool IsReversed { get; }
double Progress { get; }
bool IsChanging { get; }
bool IsEnabled { get; }
bool IsOverridingBaseValue { get; }
bool HasChanged { get; }

void Enable();
void Enable(Timeline parentTimeline);
void Disable();

void BeginIn(Time offset);
void EndIn(Time offset);
void Pause();
void Resume();
void Seek(Time offset, TimeSeekOrigin origin);

event EventHandler Begun { add; remove; }
event EventHandler Ended { add; remove; }
event EventHandler Paused { add; remove; }
event EventHandler Repeated { add; remove; }
event EventHandler Resumed { add; remove; }

```

857

```

event EventHandler Reversed { add; remove; }
event EventHandler Seeked { add; remove; }
}

```

The VideoData object is provided for video data:

```

public System.Windows.Media.VideoData : MediaData
{
    public VideoData (string source);

    //
    // MediaData
    //
    public MediaState State { get; }
    public Time MediaDuration { get; }
    public bool HasAudio { get; }
    public bool HasVideo { get; }
    public long Width { get; }
    public long Height { get; }
    public double Volume { get; set; }
    public bool Mute { get; set; }

    public void Play();

    public void Dispose();

    public bool IsChangeable { get; } // defaults to true
    public Changeable Copy();
    public void MakeUnchangeable();
    public UseStatus StatusOfNextUse { get; set; } // defaults to false
    public event EventHandler Changed { add; remove; }
    public UIContext UIContext { get; }
}

```

338

858

```

double Acceleration { get; set; }
bool AutoReverse { get; set; }

bool IsOverridingBaseValue { get; }
bool HasChanged { get; }

void Enable();
void Enable(Timeline parentTimeline);
void Disable();

void BeginIn(Time offset);
void EndIn(Time offset);
void Pause();
void Resume();
void Seek(Time offset, TimeSeekOrigin origin);

event EventHandler Begun { add; remove; }
event EventHandler Ended { add; remove; }
event EventHandler Paused { add; remove; }
event EventHandler Repeated { add; remove; }
event EventHandler Resumed { add; remove; }
event EventHandler Reversed { add; remove; }
event EventHandler Searched { add; remove; }
}

```

An element level Video object is also provided:

```

public System.Windows.Controls.Video : FrameworkElement
{
    public Video();
    //

```

859

```

// Video Element properties
//
public VideoData Source { get; set; }
public MediaState MediaState { get; }
public double Volume { get; set; }
public bool Mute { get; set; }
public int NativeVideoHeight { get; }
public int NativeVideoWidth { get; }
public Time MediaDuration { get; }

public TimeSyncValue Begin { get; set; }
public TimeSyncValue End { get; set; }
public Time Duration { get; set; }
public double RepeatCount { get; set; }
public Time RepeatDuration { get; set; }

public HorizontalAlignment HorizontalAlignment { get; set; }
public VerticalAlignment VerticalAlignment { get; set; }
public Stretch Stretch { get; set; }

//
// Video Element methods
//
public void BeginIn(Time offset);
public void EndIn(Time offset);
public void Pause();
public void Resume();
public void Seek(Time offset, TimeSeekOrigin origin);
}

```


260

An element level Audio object is also provided:

```
public System.Windows.Controls.Audio: FrameworkElement
{
    public Audio();

    //
    // Audio Element properties
    //
    public AudioData Source { get; set; }
    public MediaState MediaState { get; }
    public double Volume { get; set; }
    public bool Mute { get; set; }
    public Time MediaDuration { get; }

    public TimeSyncValue Begin { get; set; }
    public TimeSyncValue End { get; set; }
    public Time Duration { get; set; }
    public double RepeatCount { get; set; }
    public Time RepeatDuration { get; set; }

    //
    // Audio Element methods
    //
    public void BeginIn(Time offset);
    public void EndIn(Time offset);
    public void Pause();
    public void Resume();
    public void Seek(Time offset, TimeSeekOrigin origin);
}
```

661

CONCLUSION

As can be seen from the foregoing detailed description, there is provided a media integration layer including interfaces and an object model that provide program code with the ability to interface with a scene graph. The system, method and object
5 model are straightforward to use, yet powerful, flexible and extensible.

While the invention is susceptible to various modifications and alternative constructions, certain illustrated embodiments thereof are shown in the drawings and have been described above in detail. It should be understood, however, that there is no intention to limit the invention to the specific forms disclosed, but on the contrary, the
10 intention is to cover all modifications, alternative constructions, and equivalents falling within the spirit and scope of the invention.

342

862

WHAT IS CLAIMED IS:

1. A method for arranging computer graphics data for processing into an output, comprising:

5 receiving a function call via an interface of a media integration layer, the function call corresponding to graphics-related data; and causing data in a scene graph data structure to be modified based on the function call.

10 2. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to initialize a new instance of a visual class.

15 3. The method of claim 2 further comprising, receiving a function call via an interface corresponding to a transform associated with the visual.

4. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to initialize a new instance of a drawing visual class.

20 5. The method of claim 4 further comprising, receiving a function call via an interface to open the drawing visual instance for rendering, and in response, causing a drawing context to be returned, the drawing context providing a mechanism for rendering into the drawing visual.

25 6. The method of claim 1 further comprising, receiving brush data in association with the function call, and wherein causing data in a scene graph data structure to be modified comprises invoking a brush function to modify a data structure in the scene graph data structure such that when a frame is rendered from the scene graph, an area will be filled with visible data corresponding to the brush data.

30 7. The method of claim 6 wherein receiving brush data comprises receiving data corresponding to a solid color.

34?

803

8. The method of claim 6 wherein receiving brush data comprises receiving data corresponding to a linear gradient brush and a stop collection comprising at least one stop.

5

9. The method of claim 6 wherein receiving brush data comprises receiving data corresponding to a radial gradient brush.

10. The method of claim 6 wherein receiving brush data comprises receiving data corresponding to an image.

11. The method of claim 10 further comprising, receiving a function call via an interface corresponding to an image effect to apply to the image.

12. The method of claim 1 further comprising, receiving pen data in association with the function call, and wherein causing data in a scene graph data structure to be modified comprises invoking a pen function that defines an outline of a shape.

13. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a geometry-related function to represent an ellipse in the scene graph data structure.

14. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a geometry-related function to represent a rectangle in the scene graph data structure.

15. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a geometry-related function to represent a path in the scene graph data structure.

324

981

16. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a geometry-related function to represent a line in the scene graph data structure.
- 5 17. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function related to hit-testing a visual in the scene graph data structure.
- 10 18. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function related to transforming coordinates of a visual in the scene graph data structure.
- 15 19. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function related to calculating a bounding box of a visual in the scene graph data structure.
- 20 20. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function via a common interface to a visual object in the scene graph data structure.
21. The method of claim 1 further comprising invoking a visual manager to render a tree of at least one visual object to a rendering target.
- 25 22. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to place a container object in the scene graph data structure, the container object configured to contain at least one visual object.
- 30 23. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to place image data into the scene graph data structure.

349

685

24. The method of claim 23 wherein causing data in a scene graph data structure to be modified comprises invoking a function to place an image effect object into the scene graph data structure that is associated with the image data.

5 25. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to place data corresponding to text into the scene graph data structure.

10 26. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to provide a drawing context in response to the function call.

15 27. The method of claim 26 wherein the function call corresponds to a retained visual, and further comprising, calling back to have the drawing context of the retained visual returned to the scene graph data structure.

20 28. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to place a three-dimensional visual into the scene graph data structure.

29. The method of claim 28 wherein causing data in a scene graph data structure to be modified comprises mapping a two-dimensional surface onto the three dimensional visual.

25 30. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to place animation data into the scene graph data structure.

30 31. The method of claim 30 further comprising communicating timeline information corresponding to the animation data to a composition engine at another layer of the media integration layer.

345

888

32. The method of claim 31 wherein the composition engine interpolates graphics data based on the timeline to animate an output corresponding to an object in the scene graph data structure.

5 33. The method of claim 1 wherein receiving a function call via an interface of a media integration layer comprises receiving markup, and wherein causing data in a scene graph data structure to be modified comprises parsing the markup into a call to an interface of an object.

10 34. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function to place an object corresponding to audio and/or video data into the scene graph data structure.

15 35. The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises changing a mutable value of an object in the scene graph data structure.

20 36. In a computing environment, a system comprising:
a scene graph data structure of a layered system for containing data that can be rendered into output that for subsequent integrated output that can be viewed; and
an object model including objects and other data that can be contained in the scene graph data structure, at least some of the objects of the object model having interfaces for invoking functions to modify contents of the scene graph data structure.

25 37. The system of claim 26 wherein at least one function is invoked to place a tree of visual objects into the scene graph data structure.

30 38. The system of claim 37 further comprising a visual manager that when invoked renders the tree of visual objects to a rendering target.

39. The system of claim 37 wherein the tree of visual objects is contained in a visual collection object.

40. The system of claim 26 wherein at least one function is invoked to place a visual object into the scene graph data structure.

5 41. The system of claim 40 wherein at least one function is invoked to associate a brush with the visual object.

42. The system of claim 40 wherein at least one function is invoked to associate a geometry with the visual object.

10

43. The system of claim 42 wherein the geometry comprises at least one of a set containing an ellipse geometry, a rectangle geometry, a line geometry and a path geometry.

15 44. The system of claim 40 wherein at least one function is invoked to associate a transform with the visual object.

45. The system of claim 44 wherein the transform comprises a rotate transform for changing a perceived angle of the visual object.

20

46. The system of claim 44 wherein the transform comprises a scale transform for changing a perceived size of the visual object.

25 47. The system of claim 44 wherein the transform comprises a translate transform for changing a perceived position of the visual object.

48. The system of claim 44 wherein the transform comprises a skew transform for changing a perceived skew of the visual object.

30 49. The system of claim 44 further comprising animation information associated with the transform, and wherein the animation information causes transformation data associated with the transform to change over time thereby animating the transformation of the visual object over time.

868

50. The system of claim 40 wherein at least one function is invoked to associate a color with the visual object.

5 51. The system of claim 40 wherein at least one function is invoked to associate gradient data with the visual object.

52. The system of claim 40 wherein at least one function is invoked to associate a tile brush with the visual object.

10

53. The system of claim 40 wherein at least one function is invoked to associate an image with the visual object.

15 54. The system of claim 40 wherein at least one function is invoked to associate three-dimensional data with the visual object.

55. The system of claim 40 wherein at least one function is invoked to associate a drawing comprising drawing primitives with the visual object.

20 56. The system of claim 40 wherein at least one function is invoked to associate audio and/or video media d with the visual object.

57. The system of claim 40 wherein at least one function is invoked to associate an image effect with the visual object.

25

58. The system of claim 40 wherein at least one function is invoked to associate a pen with the visual object, to describe how a shape is outlined.

30 59. The system of claim 40 wherein at least one function is invoked to obtain a drawing context associated with the visual object.

318

289

60. The system of claim 40 wherein at least one function is invoked to associate hit testing data with the visual object.

5 61. The system of claim 40 wherein at least one function is invoked to associate a rectangle with the visual object.

62. The system of claim 61 wherein at least one function is invoked to describe how a source rectangle should be stretched to fit a destination rectangle corresponding to the visual object.

10

63. The system of claim 61 wherein at least one function is invoked to describe how content is positioned vertically within a container corresponding to the visual object.

15

64. The system of claim 61 wherein at least one function is invoked to describe how content is positioned horizontally within a container corresponding to the visual object.

20

65. In a computing environment, a system comprising:
interface means for receiving function calls;
high-level composition means for integrating graphics-related data and and/or media-related data received via the interface means into a scene graph; and
rendering means for converting the scene graph into output that may be transmitted or displayed.

25

66. The system of claim 65 wherein the rendering means includes low-level composition means for constructing a frame for viewing based on data received from the high-level composition engine.

30

67. The system of claim 65 further comprising animation means, the high-level composition engine providing timeline data to the low-level composition means for

349

370

interpolating the appearance of visible data across at least two frames to animate the visible data over time.

350

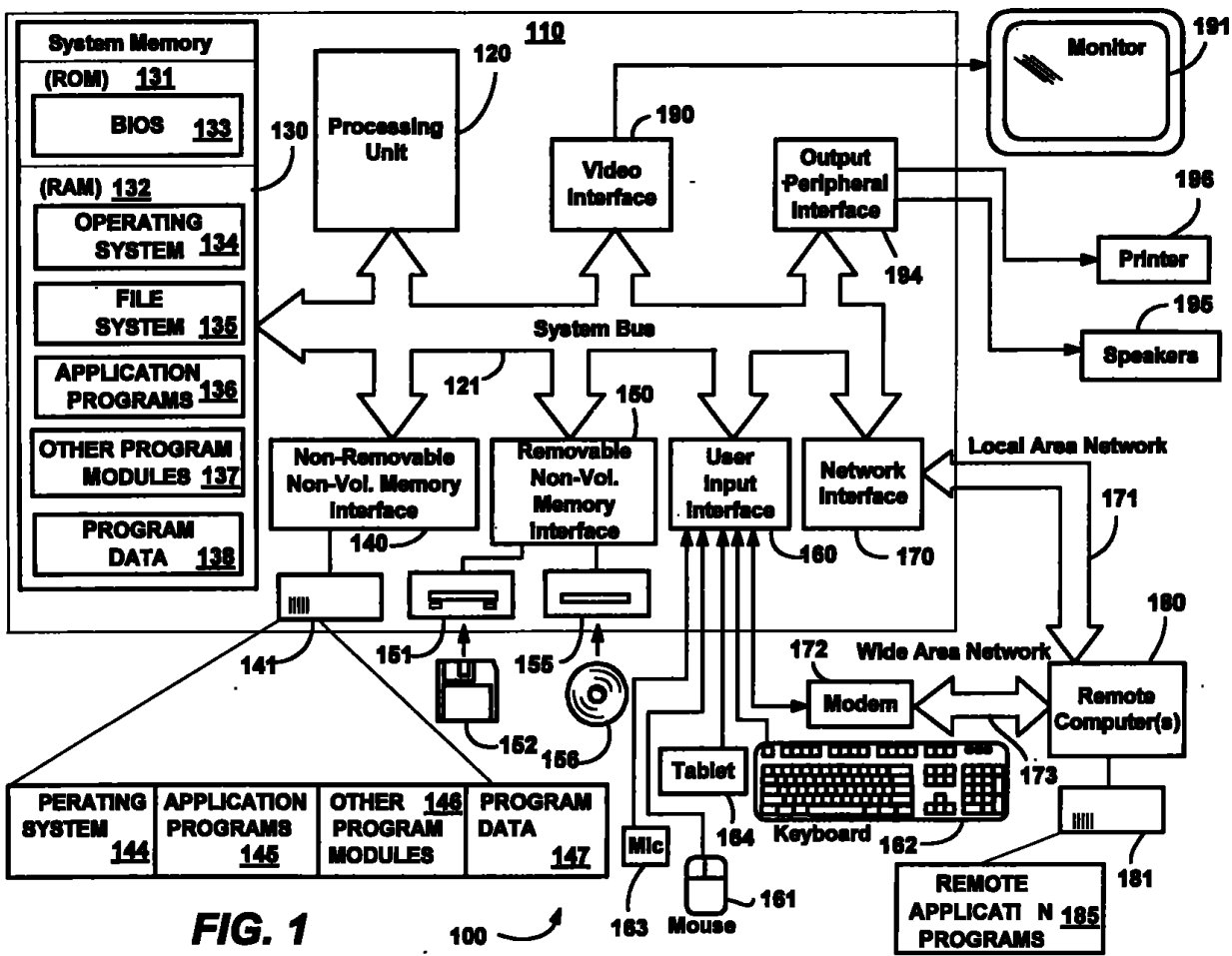
821

ABSTRACT

A media integration layer including an application programming interface (API) and an object model allows program code developers to interface in a consistent manner with a scene graph data structure in order to output graphics. Via the
5 interfaces, program code adds child visuals to other visuals to build up a hierarchical scene graph, writes Instruction Lists such as geometry data, image data, animation data and other data for output, and may specify transform, clipping and opacity properties on visuals. The media integration layer and API enable programmers to accomplish composition effects within their applications in a straightforward manner, while
10 leveraging the graphics processing unit in a manner that does not adversely impact normal application performance. A multiple-level system includes the ability to combine different media types (such as 2D, 3D, Video, Audio, text and imaging) and animate them smoothly and seamlessly.

15

351



873

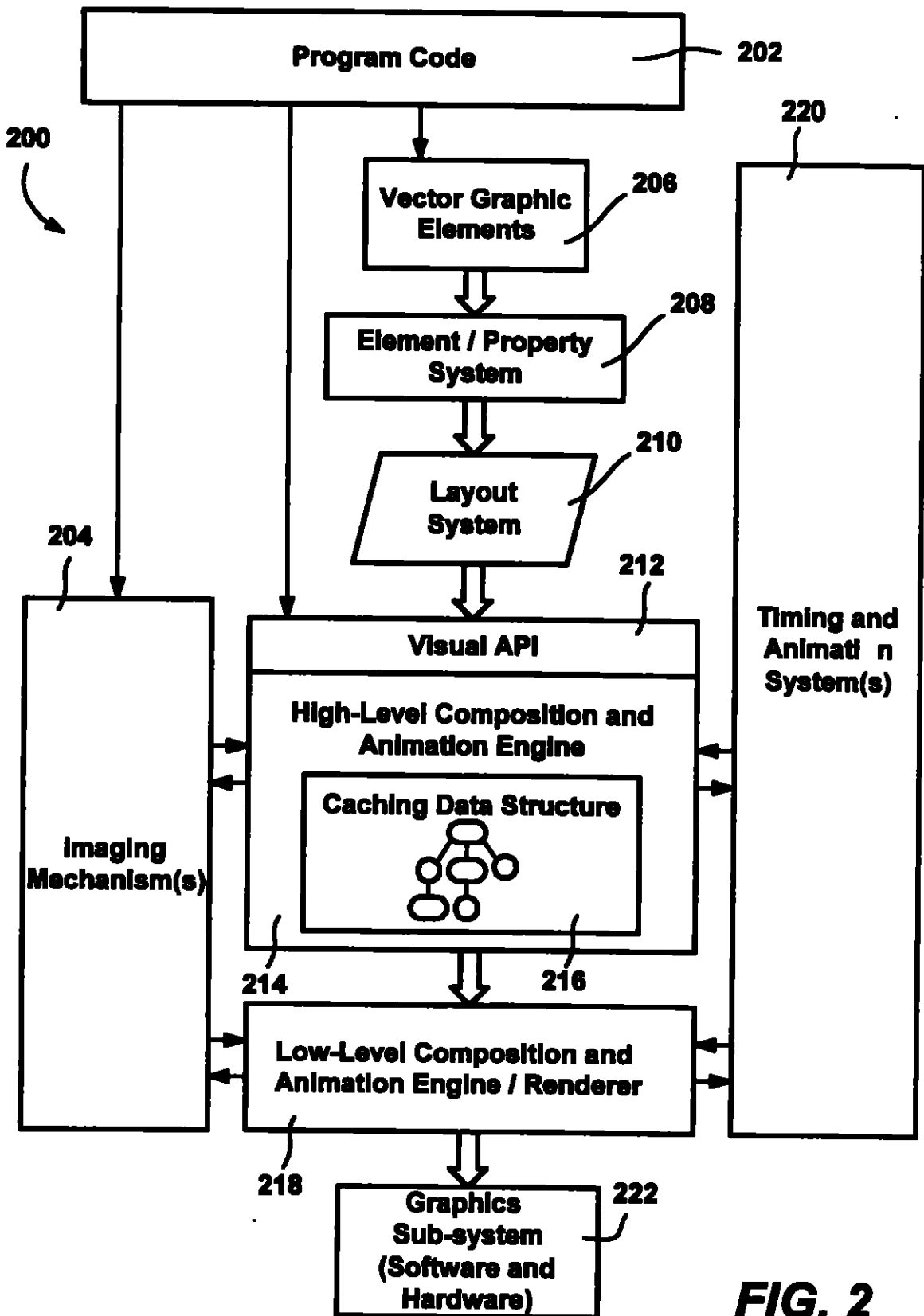


FIG. 2

353

674

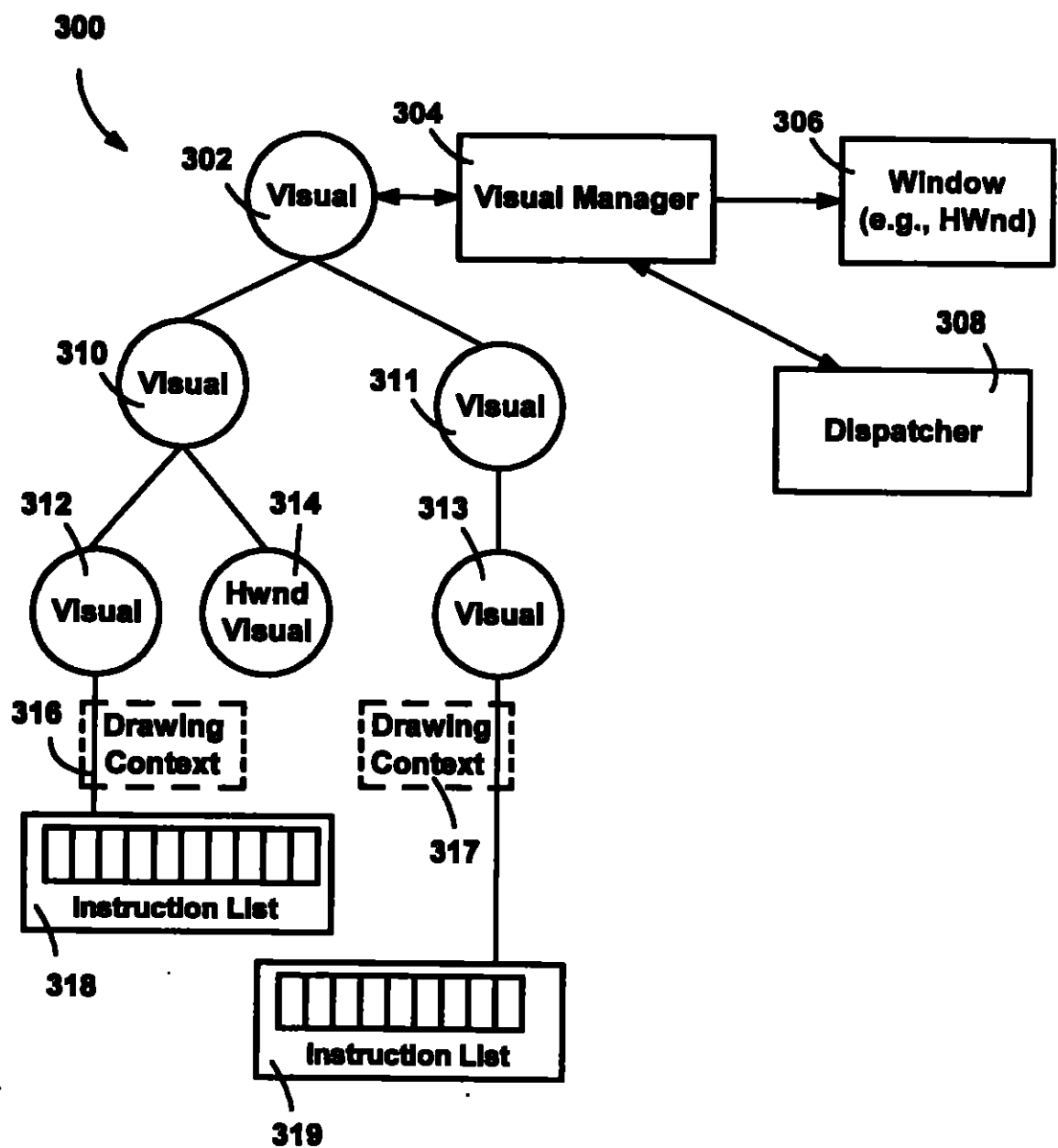
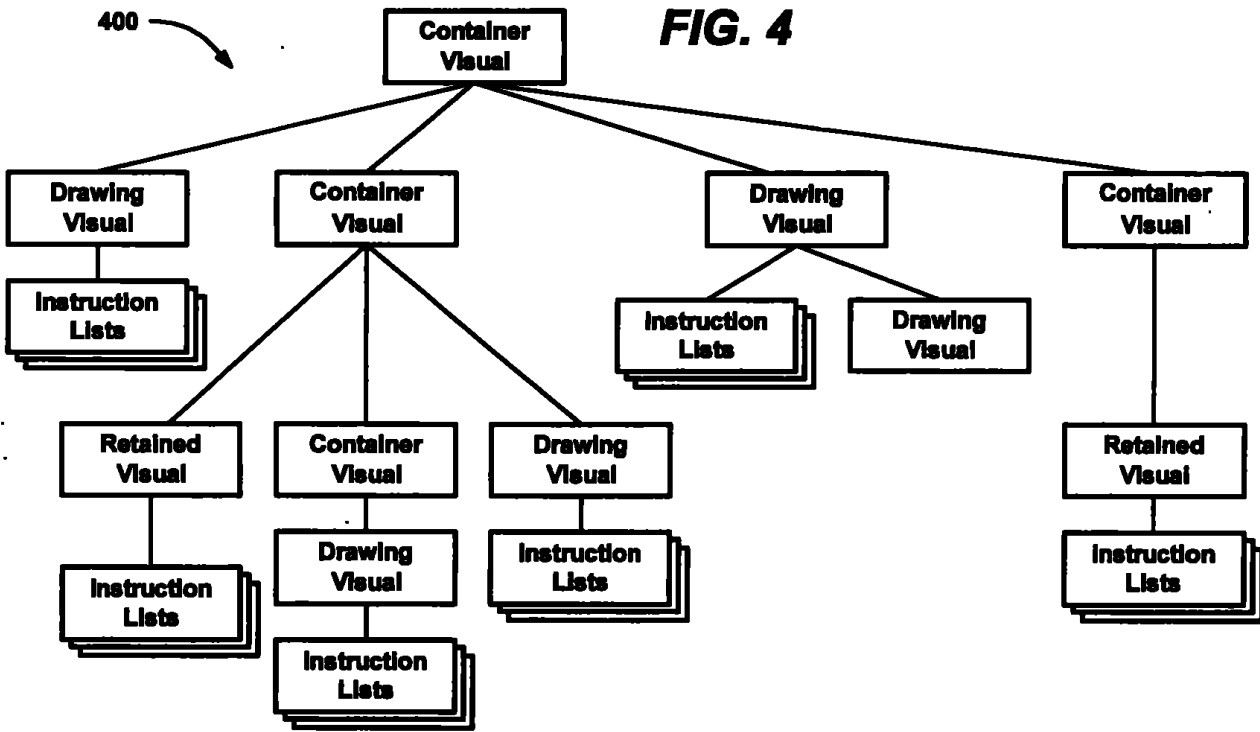


FIG. 3

959

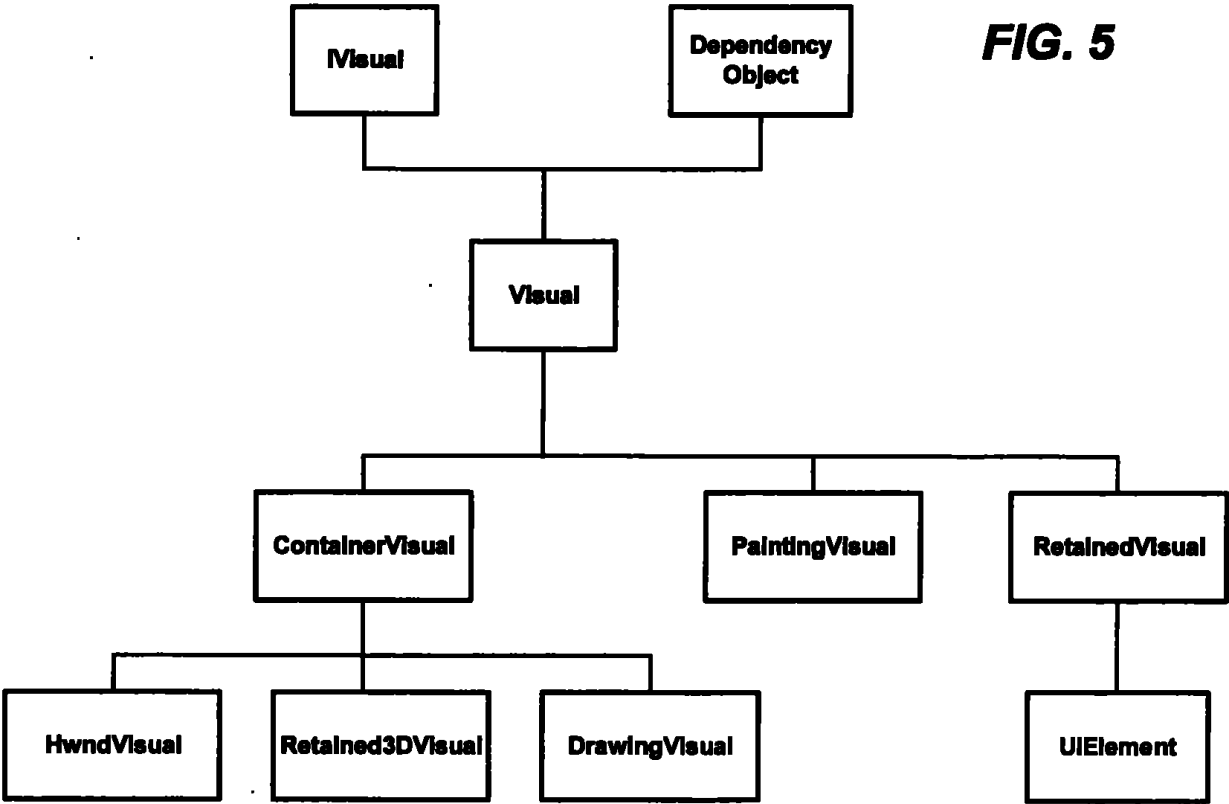
400

FIG. 4



875

FIG. 5



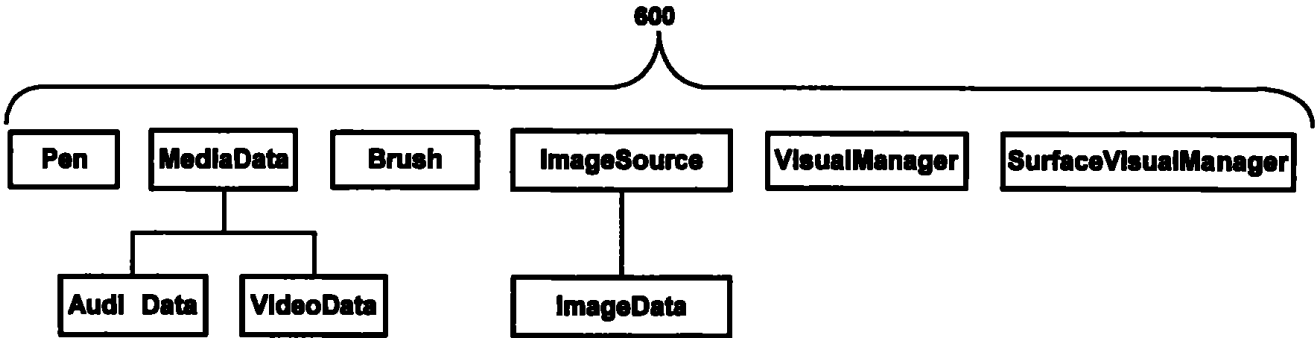


FIG. 6

6 / 29

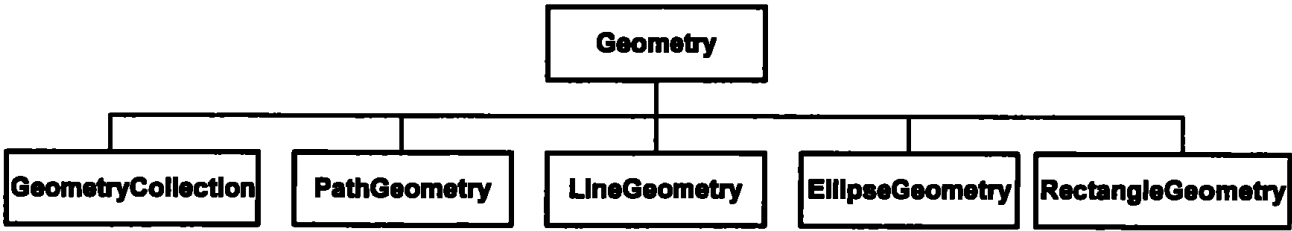


FIG. 10

857

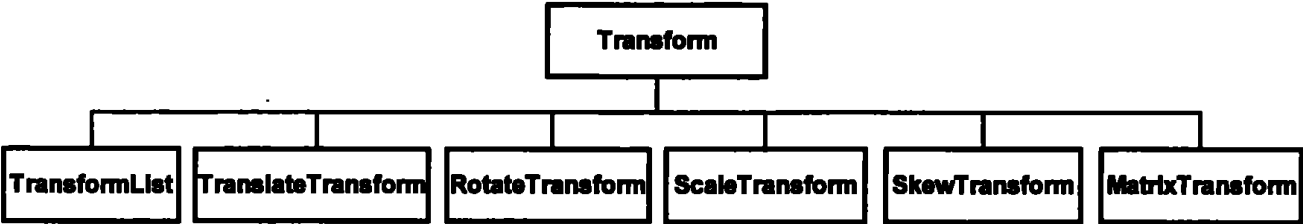


FIG. 7

7 / 29

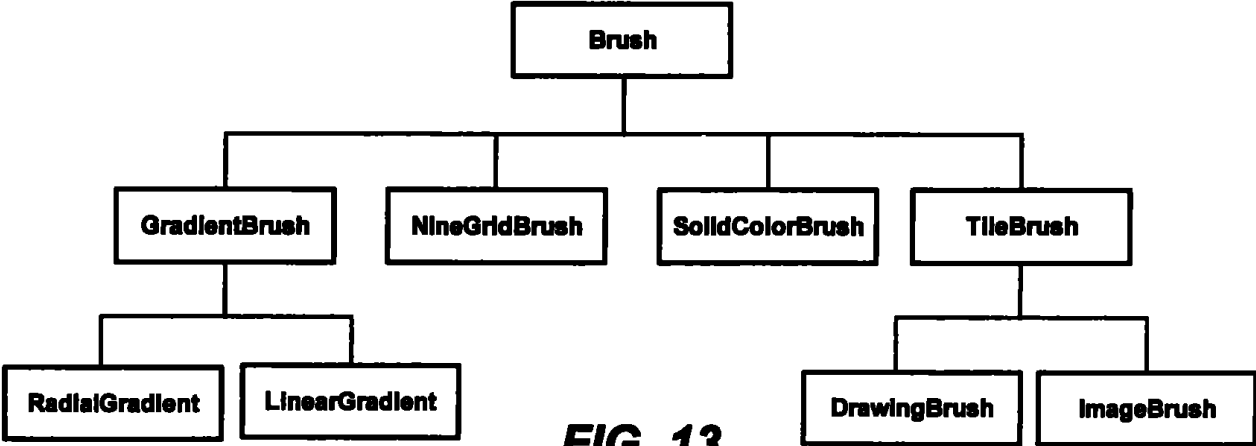


FIG. 13

Fig 13

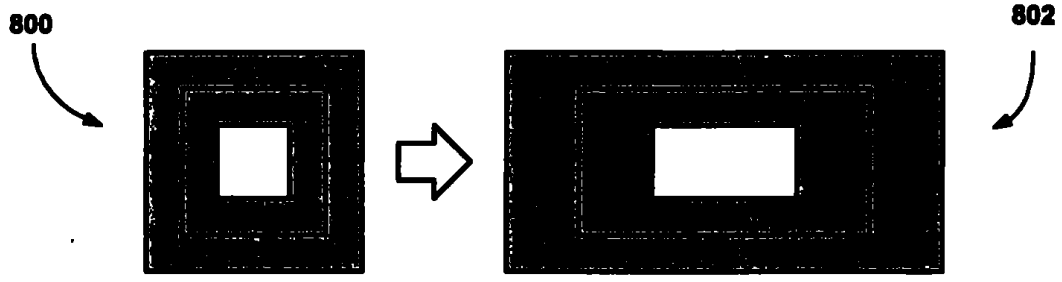


FIG. 8

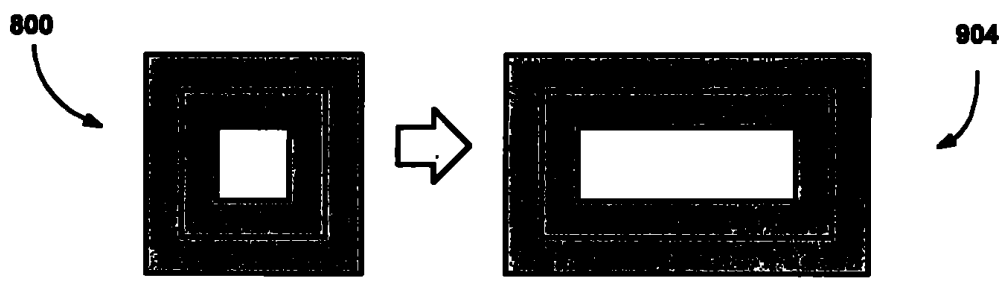


FIG. 9

356

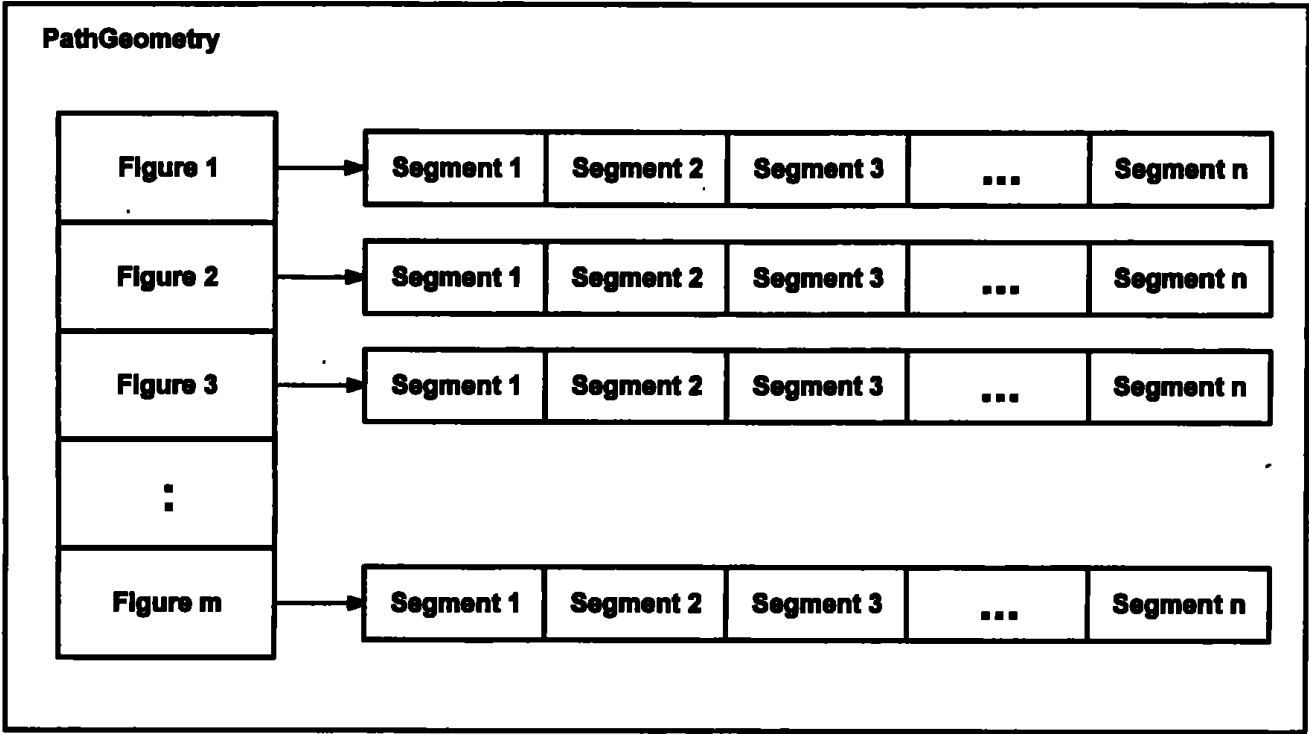


FIG. 11

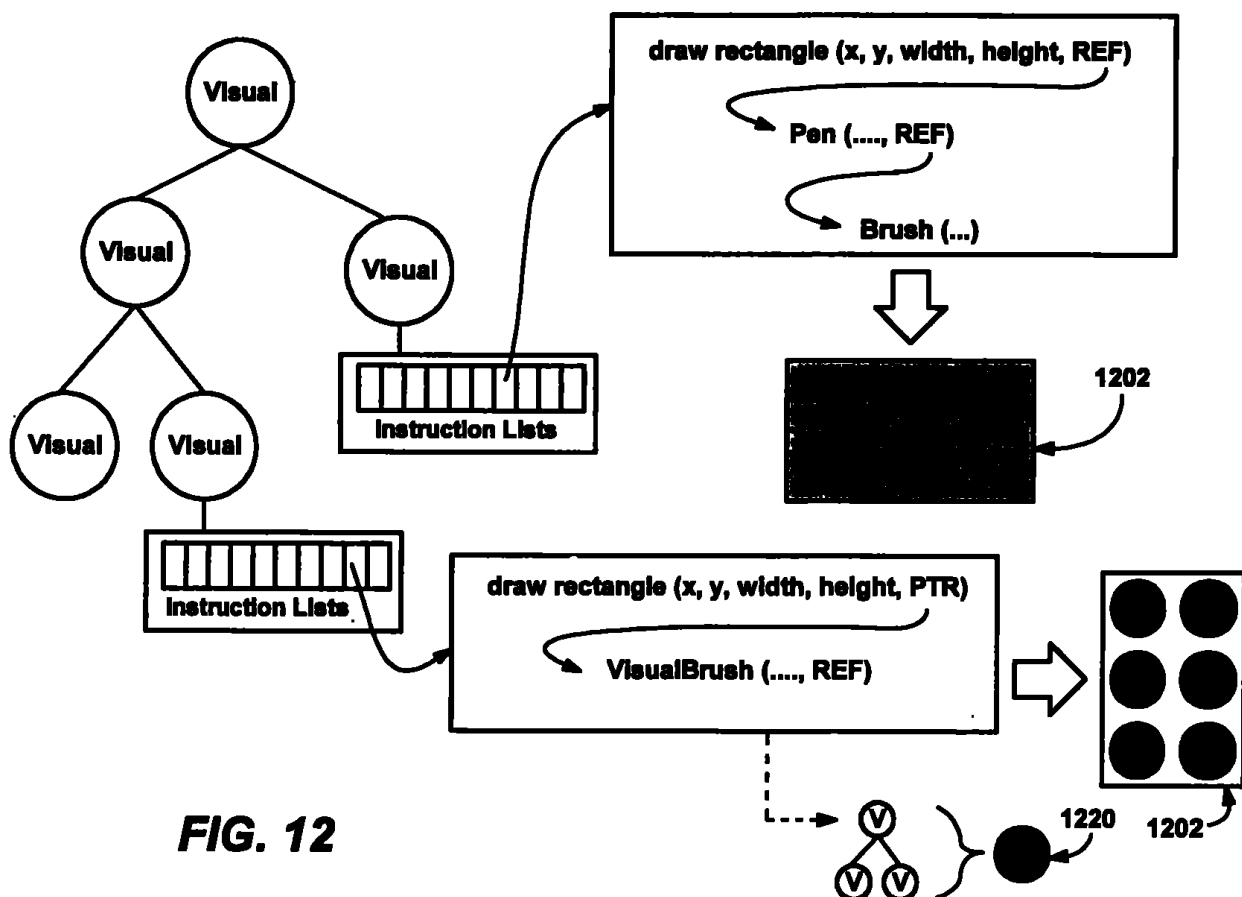


FIG. 12

002

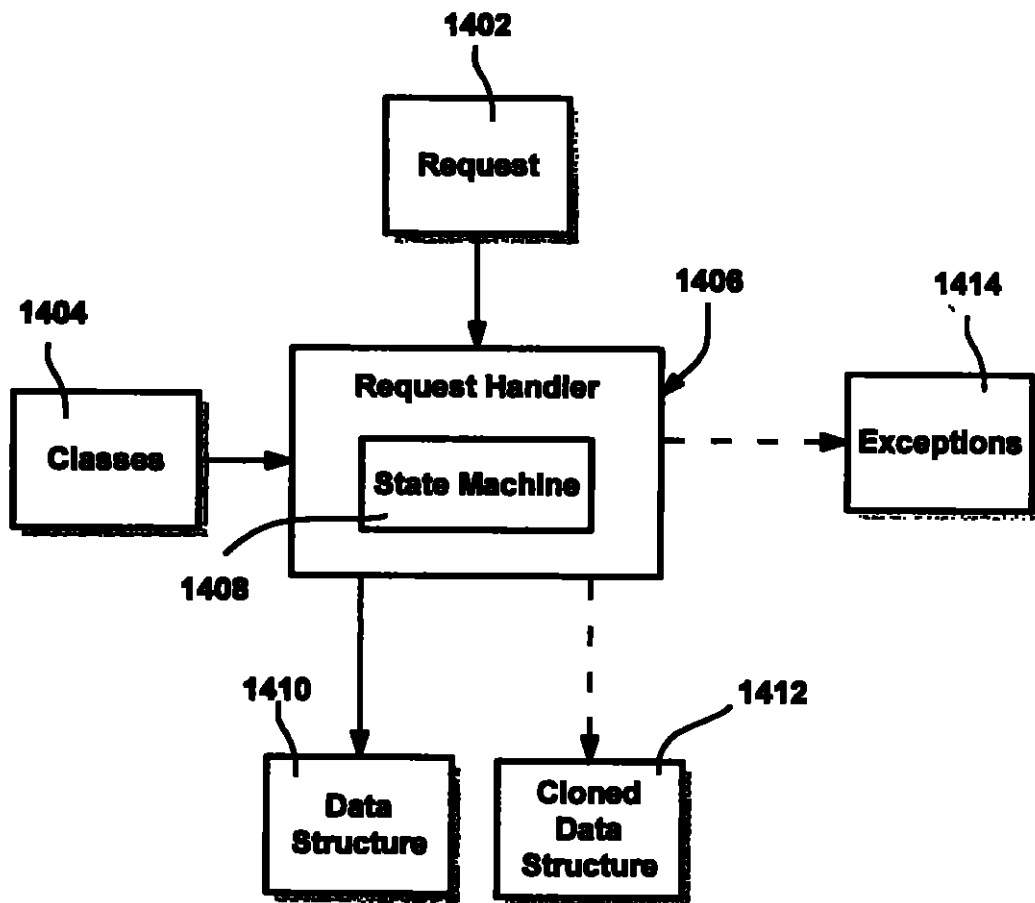


FIG. 14

3/2

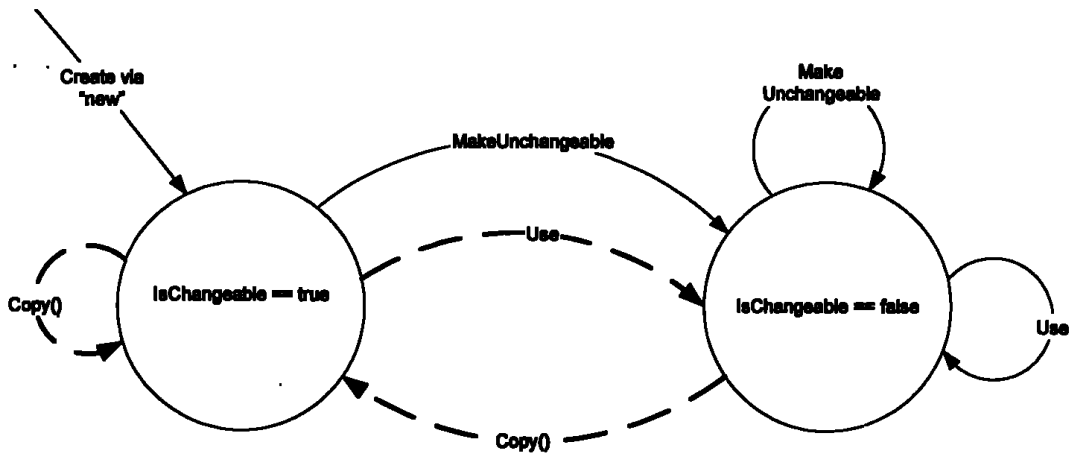


FIG. 15

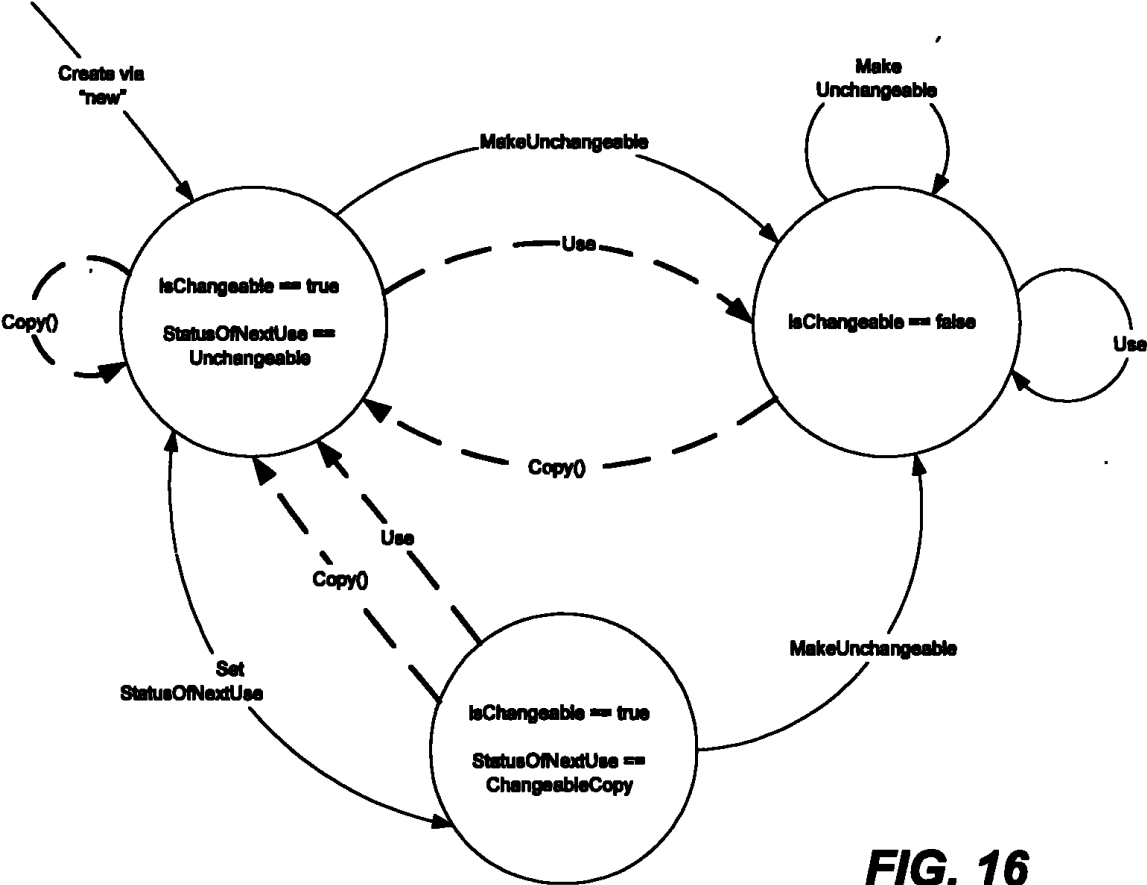


FIG. 16

26-1

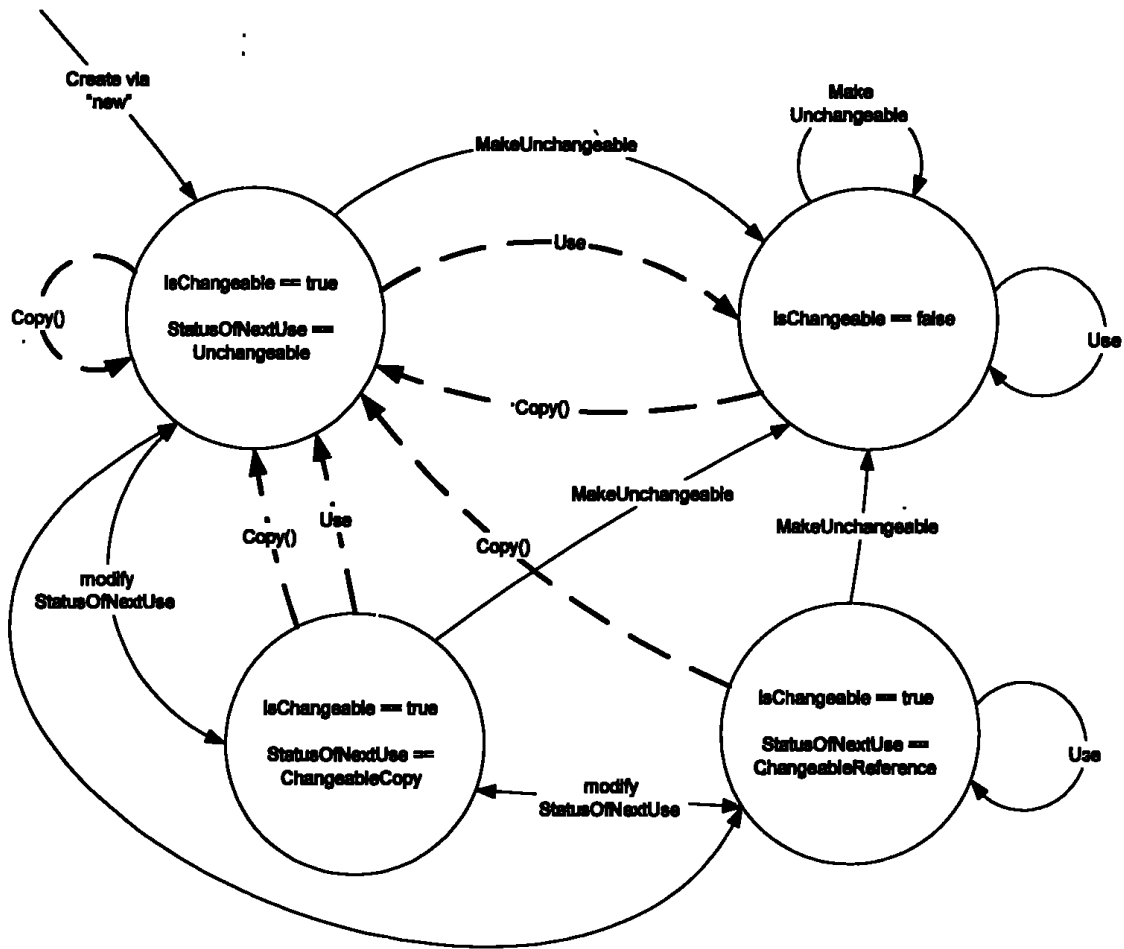


FIG. 17

30

FIG. 18

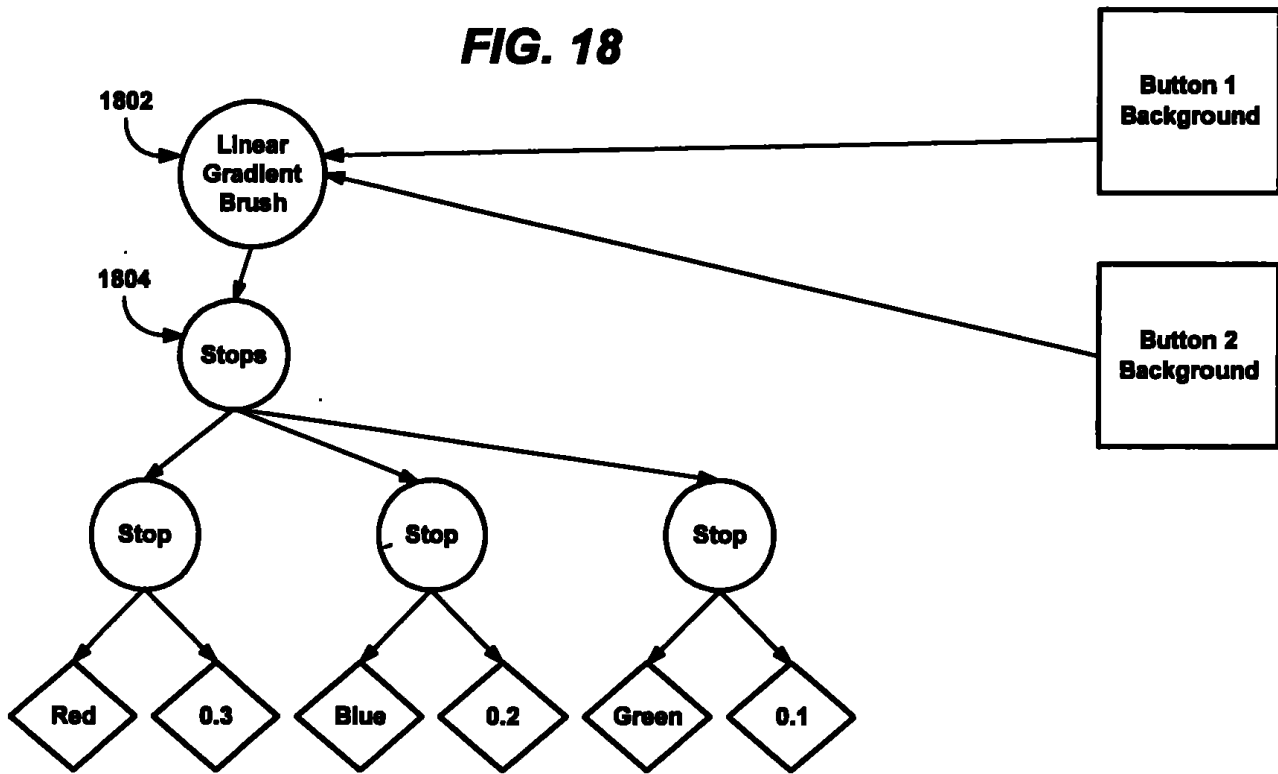


FIG. 19

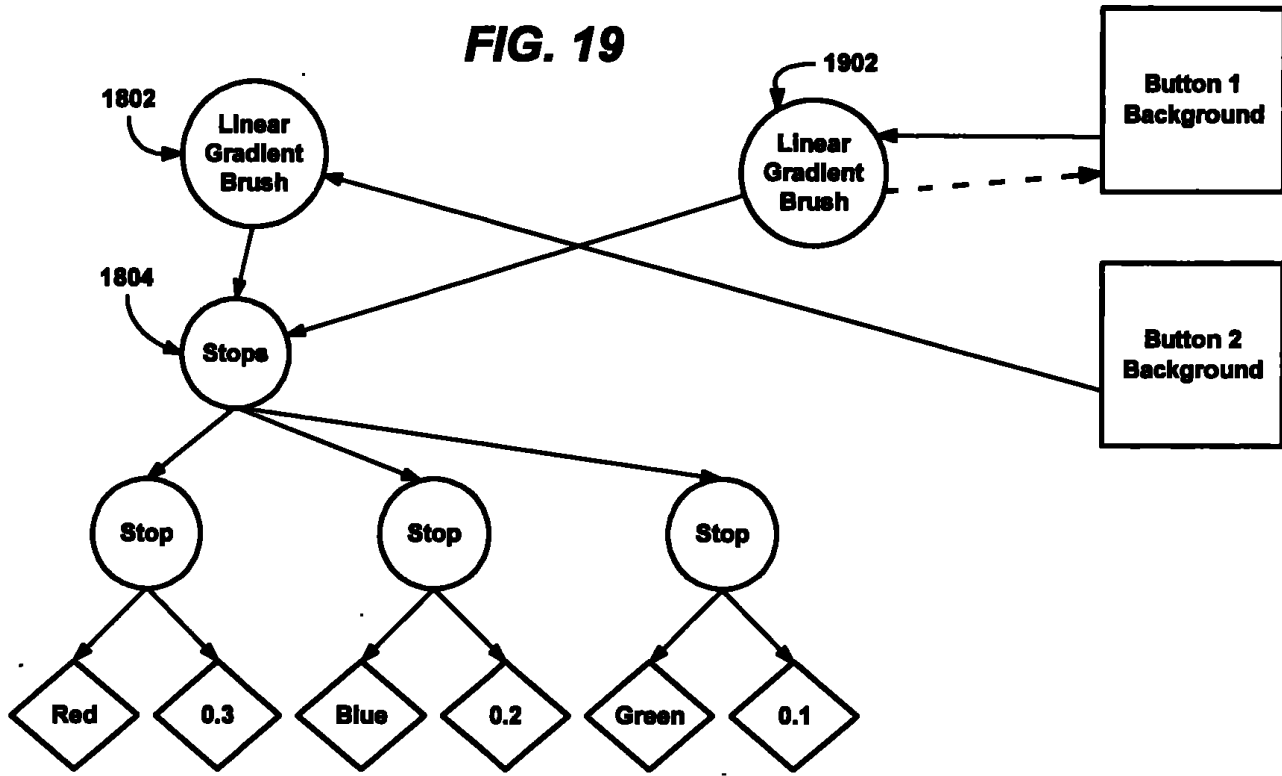
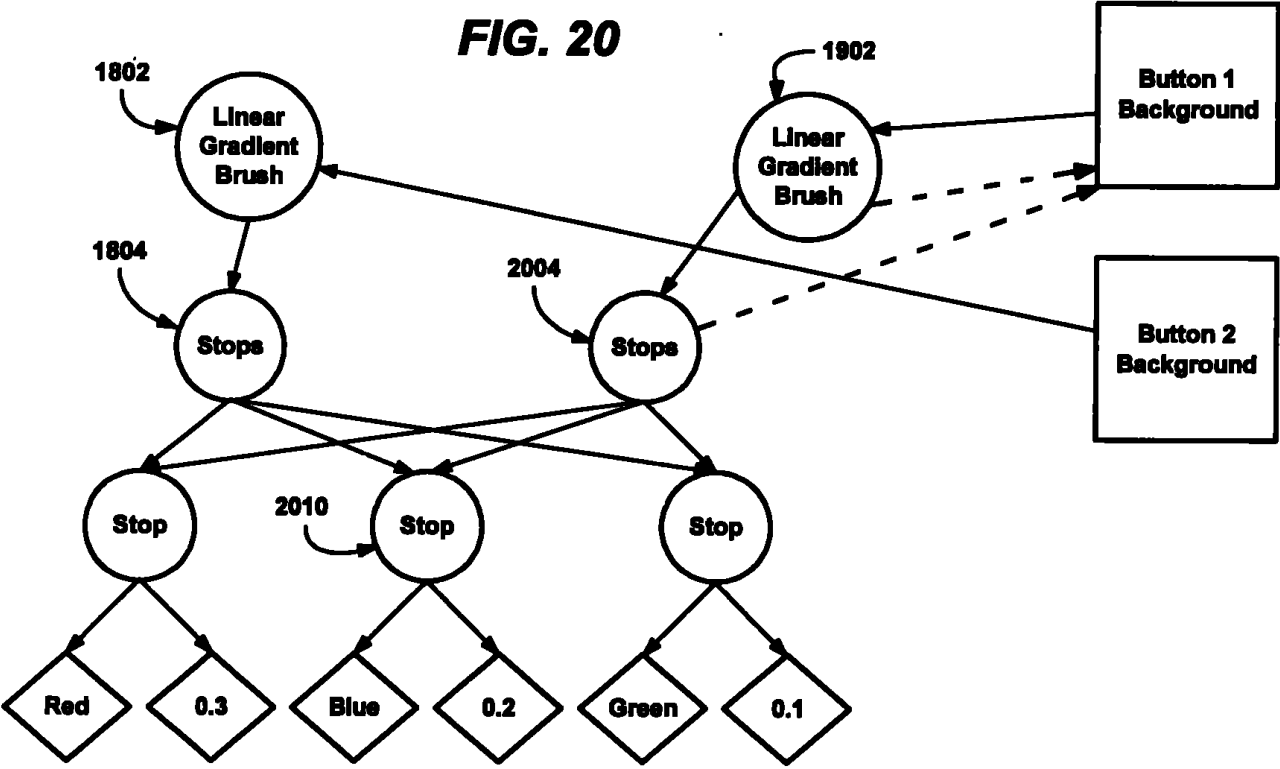
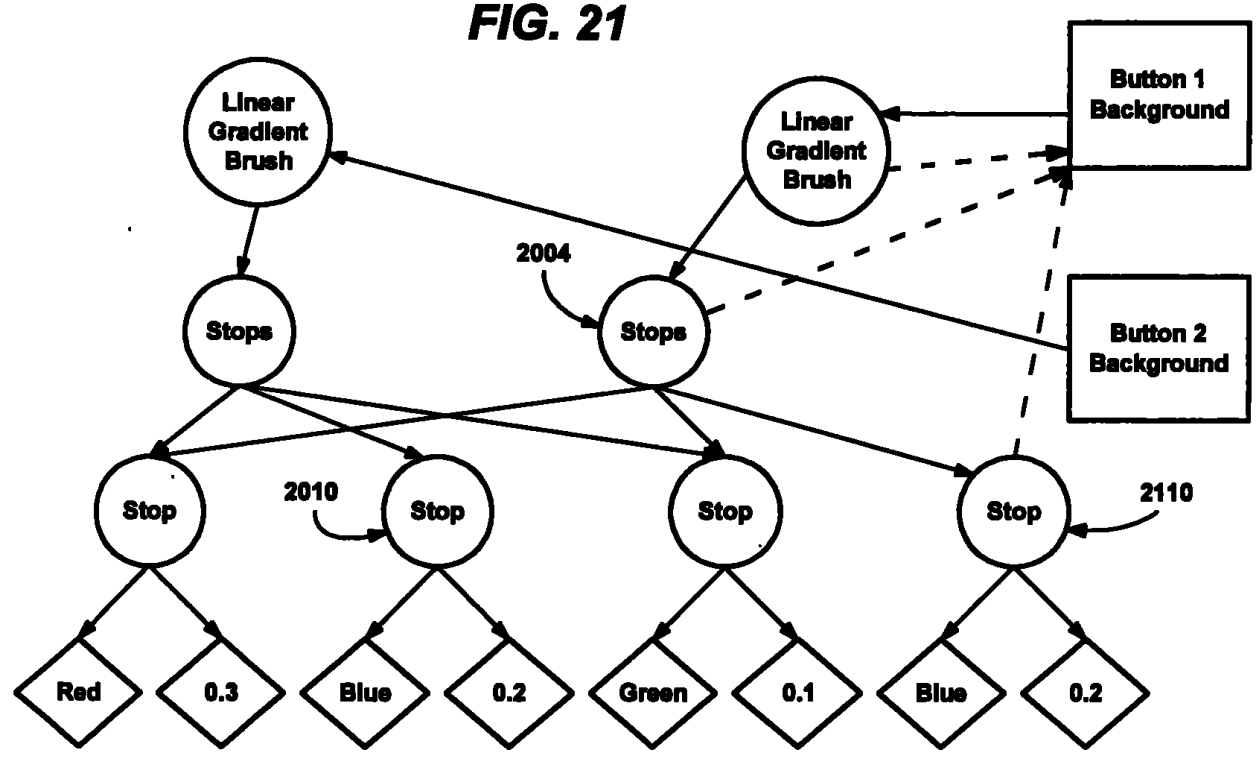


FIG. 20



368

FIG. 21



369

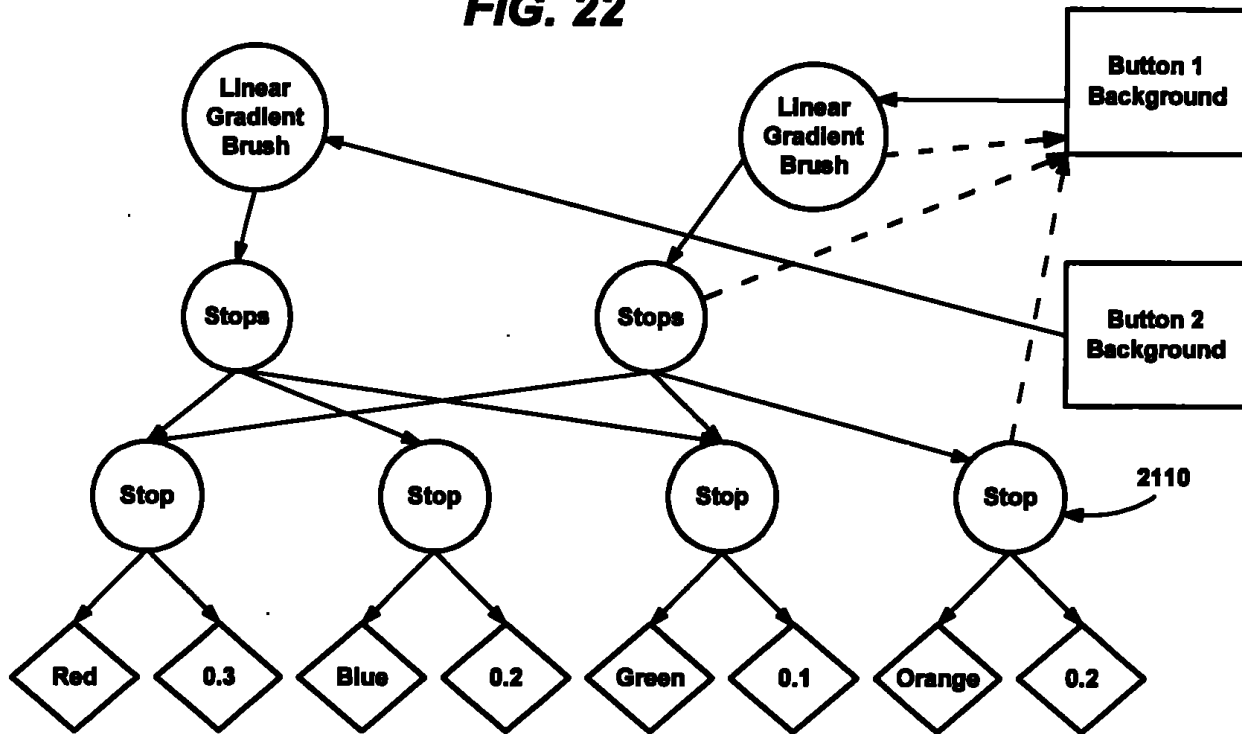
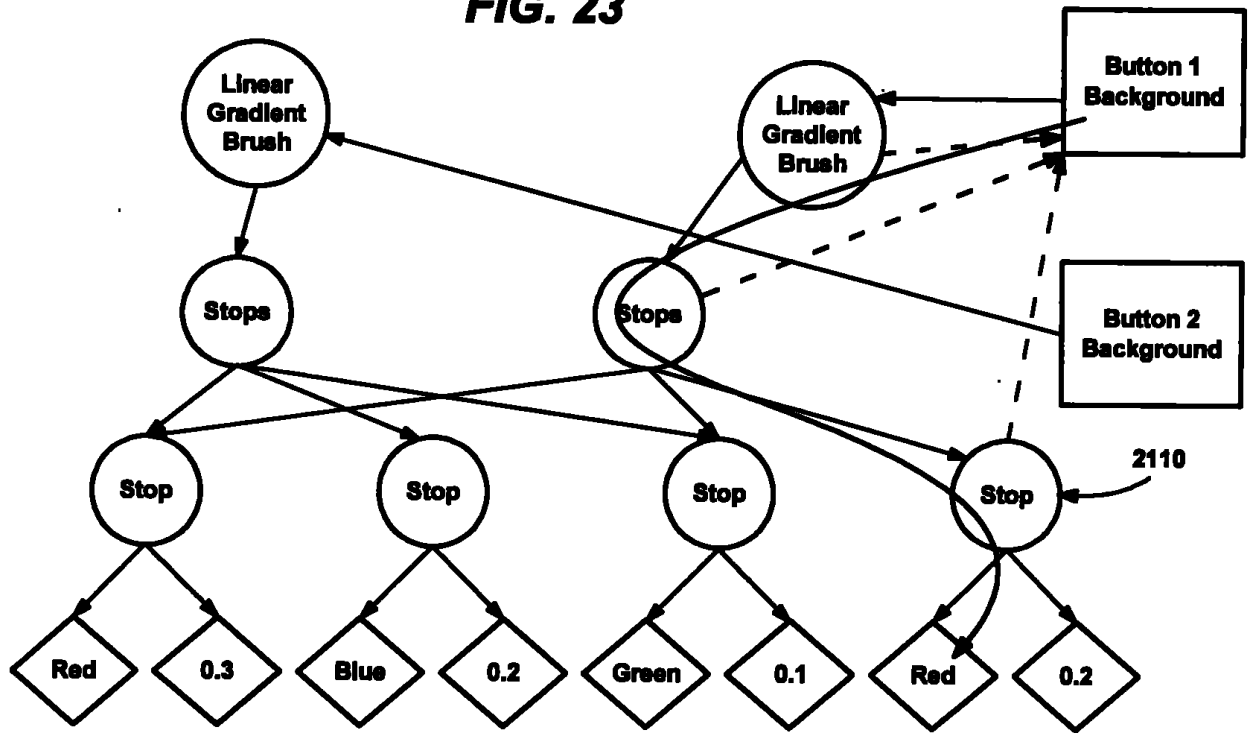
FIG. 22

FIG. 23



371

892

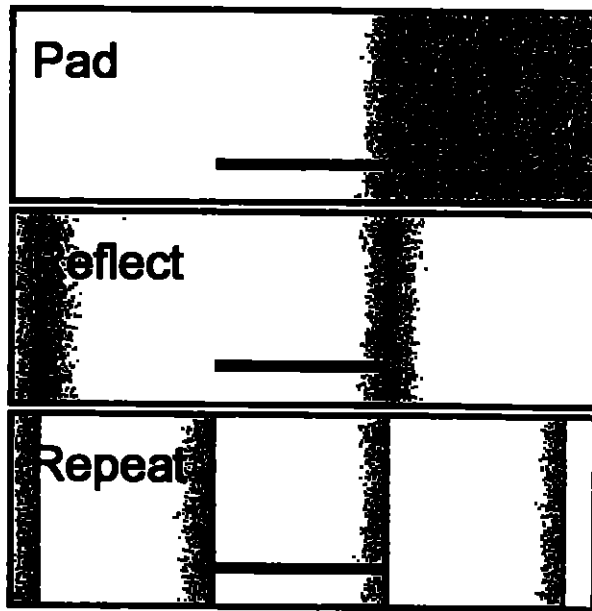


FIG. 24

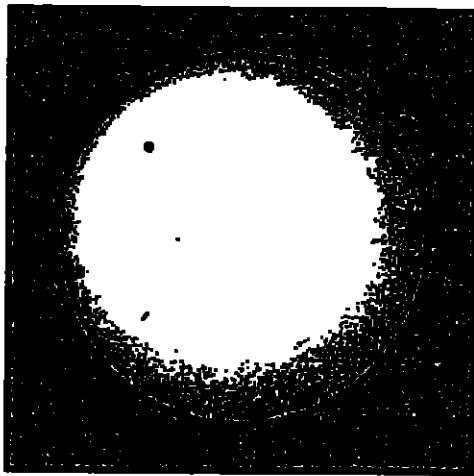


FIG. 25

372

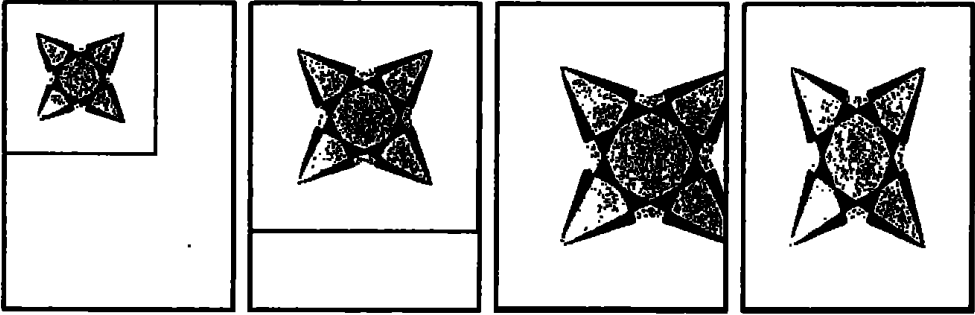


FIG. 26

373

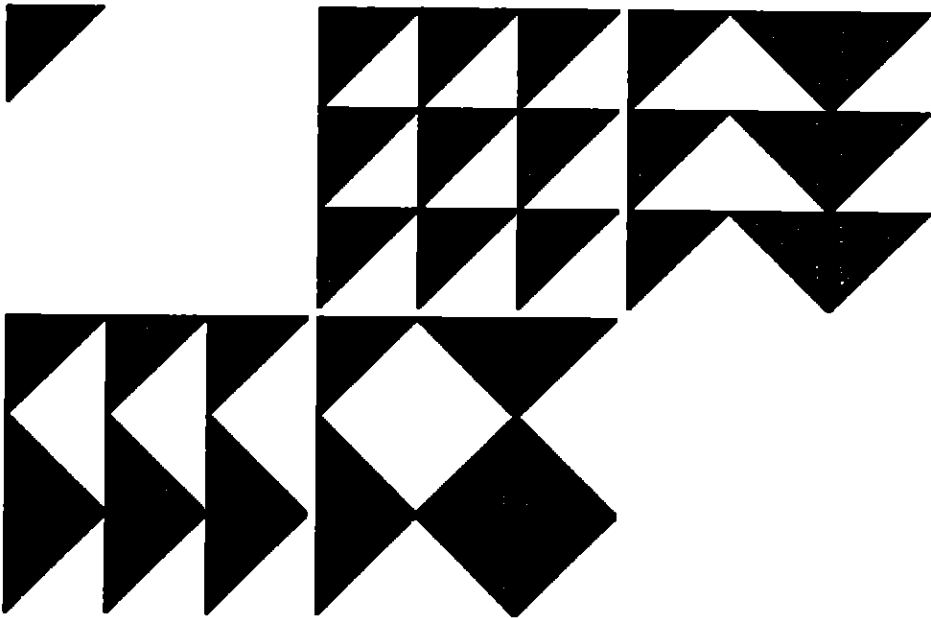


FIG. 27

805

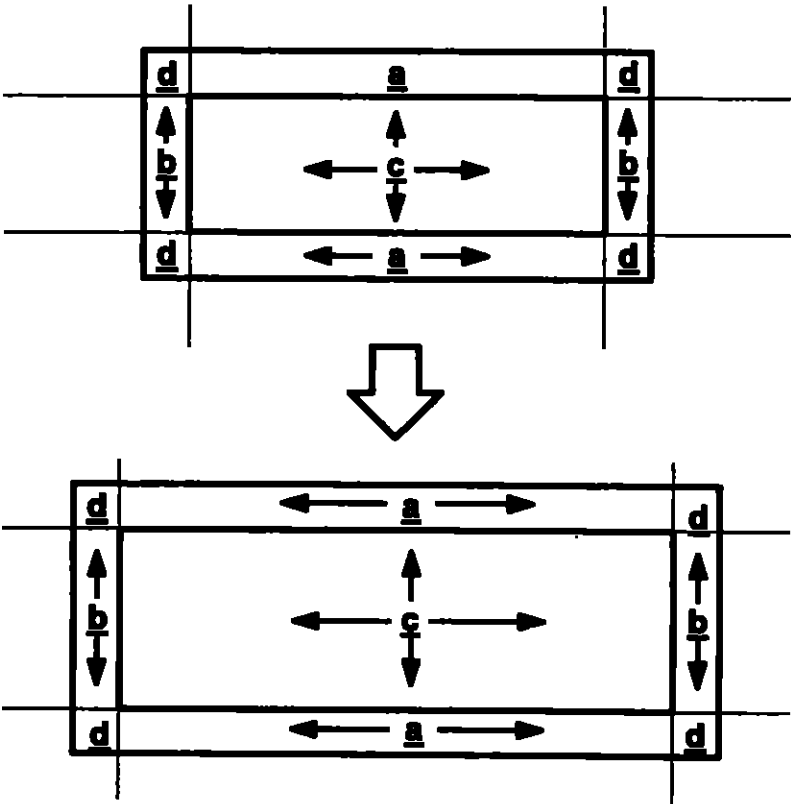


FIG. 28

375

89611

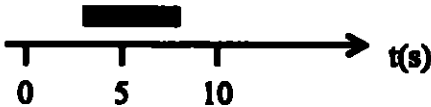


FIG. 29

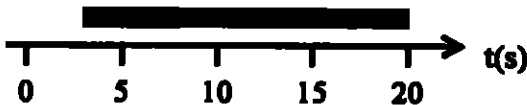


FIG. 30

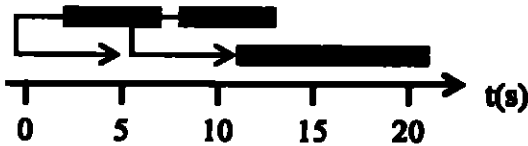


FIG. 31

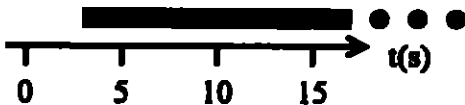


FIG. 32

326

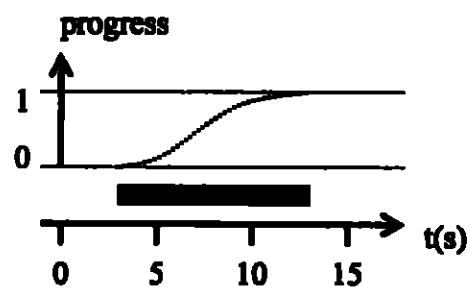


FIG. 33

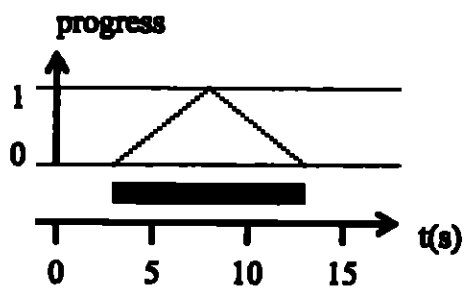


FIG. 34

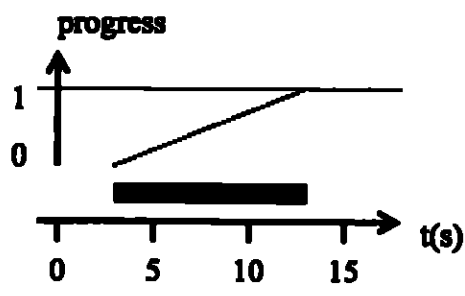


FIG. 35

898

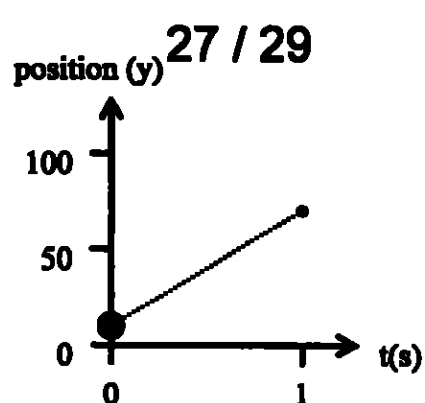


FIG. 36

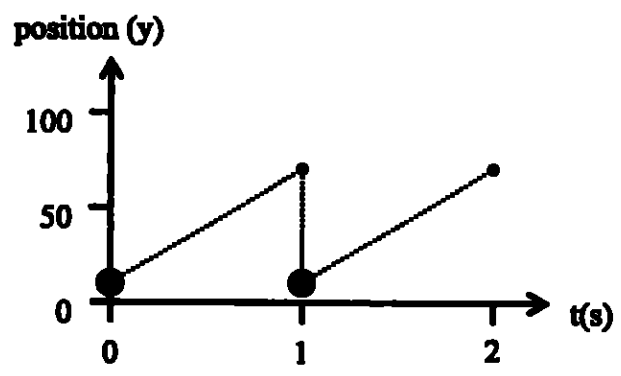


FIG. 37

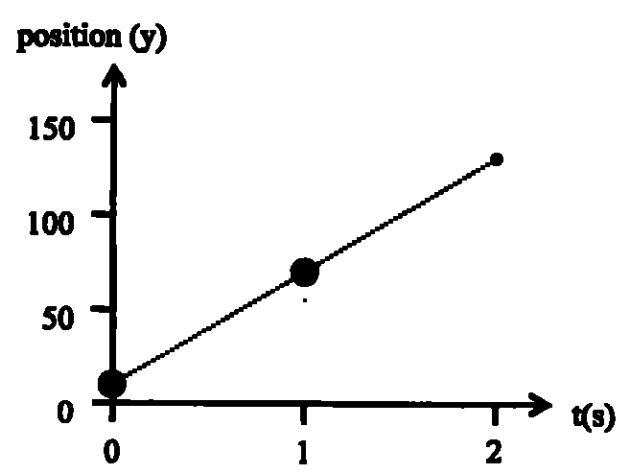


FIG. 38

378

8491

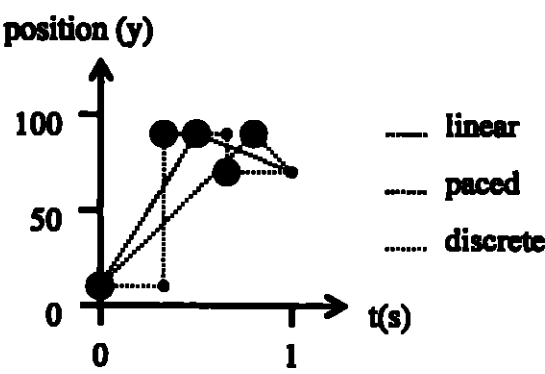


FIG. 39

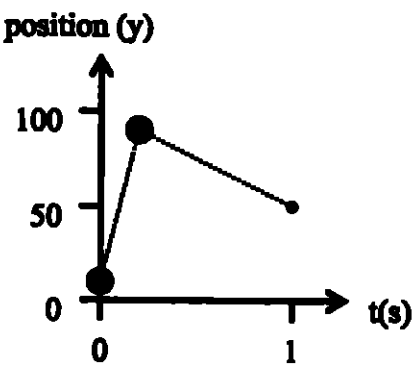


FIG. 40

379

700

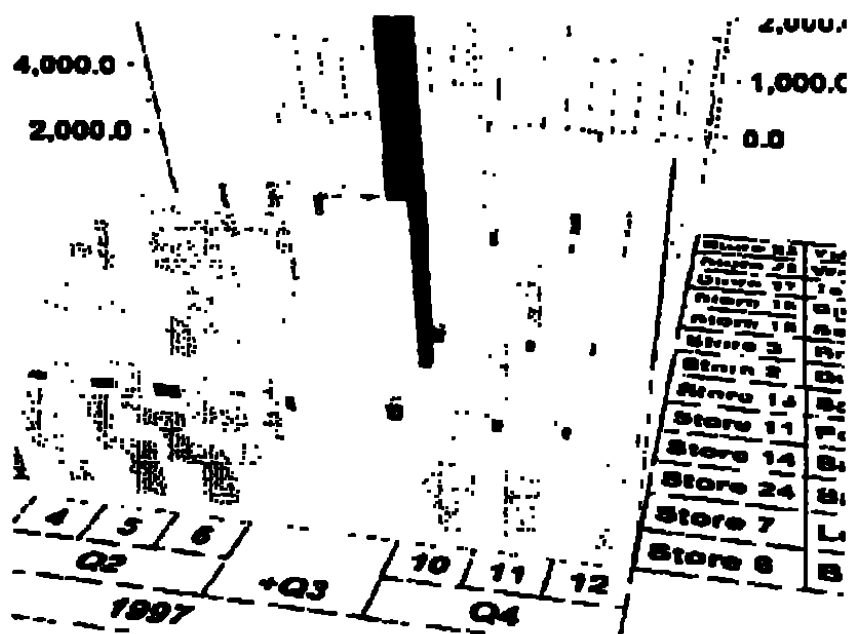


FIG. 41

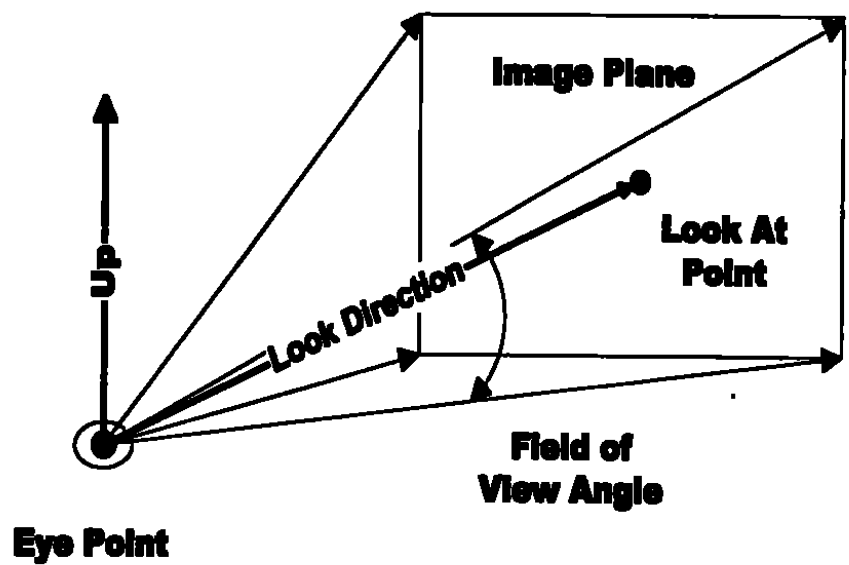


FIG. 42

06

adi

SOLICITUD**PCT**

Los abajo firmantes solicitan que la presente solicitud internacional sea procesada de acuerdo con el Tratado de Cooperación de Patentes.

PARA USO EXCLUSIVO DE LA OFICINA RECEPTORA

Solicitud Internacional No.

Fecha de Presentación Internacional

Nombre de la Oficina Receptora y "Solicitud Internacional PCT"

Archivo de Referencia del Agente o Solicitante

(si se desea) (12 caracteres máximo)

4121

Casilla No. I TÍTULO DE LA INVENCIÓN**CAPA DE INTEGRACIÓN DE MEDIOS****Casilla No. II SOLICITANTE**☐ Esta persona también
es inventor

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de

381

la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).

MICROSOFT CORPORATION
One Microsoft Way
Redmond, Washington 98052
US

Teléfono No. **425-822-8080**

Facsímile No.

Teleprinter No.

No. de registro del Solicitante con la Oficina

Estado (que es, el país) de nacionalidad:

US

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

☒ Todos los estados designados

☐ Todos los estados designados excepto los Estados Unidos de América

☐ Solamente los Estados Unidos de América

☐ Los estados indicados en la casilla complementaria

Casilla No. III SOLICITANTE(S) Y/O INVENTOR(ES)

ADICIONAL(ES)

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).

**Subramanian, Sriram
6209 10th Avenue NE
Kirkland, Washington 98033
US**

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)

Estado (que es, el país) de nacionalidad:

US

Estado (que es, el país) de residencia:

US

904

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
 - ☐ Todos los estados designados excepto los Estados Unidos de América
 - ☐ Solamente los Estados Unidos de América
 - ☐ Los estados indicados en la casilla complementaria
- ☒ Solicitantes adicionales y/o inventores (adicionales) se indican en una hoja a continuación.

**Casilla No. IV AGENTE O REPRESENTANTE COMÚN; O DIRECCIÓN
DE CORRESPONDENCIA**

Por medio de la presente la persona identificada adelante ha sido señalada para actuar en nombre del (los) solicitante (s) ante las Autoridades Internacionales competentes como:

- ☒ Agente
- ☐ Representante común

334

405

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país).

Michalik, Albert S.
704-228th Avenue NE, Suite 193
Sammamish, WA 98074
US

Teléfono No. 425-836-3030

Facsimile No. 425-836-3003

Teleprinter No.

No. De registro del Agente con la Oficina 37,395

☐ **Dirección de correspondencia:** Marque esta casilla de verificación cuando ningún agente o representante común es/haya sido señalado y el espacio anterior se utiliza en cambio para indicar una dirección espacial a la cual la correspondencia deba ser enviado.

Formulario PCT/RO/101 (primera hoja) (Enero 2004)

Ver notas al formulario de solicitud

725

906

Hoja No. 2

**Continuación de la Casilla No. III SOLICITANTES
ADICIONALES Y/O INVENTORES (ADICIONALES)**

***Si ninguna de las siguientes subcasillas se utiliza, esta
hoja no se incluye en la solicitud***

**Nombre y dirección: (apellido seguido por nombre; para
una persona jurídica nombre legal completo. La dirección
debe incluir código postal y nombre del país. El país de
la dirección indicada en esta casilla es el Estado del
solicitante (que es, el país) de residencia si no se
indica adelante Estado de residencia).**

**Blanco, Leonardo E.
9935 225th Avenue NE
Redmond, Washington 98053
US**

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor (si esta casilla de verificación es
marcada, no llene adelante)

906

907

Estado (que es, el país) de nacionalidad:

Uruguay

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América
- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).*

**Curtis, Donald B.
13533 SE 52nd Street
Bellevue, Washington 98006
US**

367

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)

Estado (que es, el país) de nacionalidad:

US

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América
- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

Nombre y dirección: (apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del

901

solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).

Beda, Joseph S.
2437 East Lake Washington
Seattle, Washington 98112
US

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor (si esta casilla de verificación es marcada, no llene adelante)

Estado (que es, el país) de nacionalidad:

Canadá

Estado (que es, el país) de residencia:

US

Esta persona es solicitante para los propósitos de:

- ☐ Todos los estados designados
- ☐ Todos los estados designados excepto los Estados Unidos de América

3-87

- ☐ Solamente los Estados Unidos de América
- ☐ Los estados indicados en la casilla complementaria

Nombre y dirección: *(apellido seguido por nombre; para una persona jurídica nombre legal completo. La dirección debe incluir código postal y nombre del país. El país de la dirección indicada en esta casilla es el Estado del solicitante (que es, el país) de residencia si no se indica adelante Estado de residencia).*

Schnelder, Gerard A.
8 West Aloha Street, #304
Seattle, Washington 98119
US

Esta persona es:

- ☐ Solo solicitante
- ☐ Solicitante e inventor
- ☒ Solo inventor *(si esta casilla de verificación es marcada, no llene adelante)*

Estado (que es, el país) de nacionalidad:

Alemania

911

```

        Point3D rotationCenter);

    public static ScaleTransform3D CreateScale(Vector3D scaleVector);
    public static ScaleTransform3D CreateScale(Vector3D scaleVector,
5        Point3D scaleCenter);

    public static Transform3D Identity { get; }

    // Instance members
10    public bool IsAffine { get; }

    public Point3D Transform(Point3D point);
    public Vector3D Transform(Vector3D vector);
    public Point4D Transform(Point4D point);
15
    public void Transform(Point3D[] points);
    public void Transform(Vector3D[] vectors);
    public void Transform(Point4D[] points);
    }
20

```

Note that the Transform() methods that take Point3D/Vector3D will raise an exception if the transform is not affine.

Transform3DCollection will mimic TransformCollection, with the Add* methods modified in the same way that the Create* methods above are modified :

```

25    public sealed class System.Windows.Media3D.Transform3DCollection :
        Transform3D, IList
    {
        // follow the model of TransformCollection
    }

```

912

AffineTransform3D is a base class that concrete affine 3D transforms derive from (translate, skew, rotate, scale), and it exposes read access to a Matrix3D:

```

5 public abstract class System.Windows.Media3D.AffineTransform3D : Transform3D
{
    internal AffineTransform3D(); // non-extensible

    public virtual Matrix3D Value { get; }
}

```

10 TranslateTransform3D:

```

public sealed class System.Windows.Media3D.TranslateTransform3D :
    AffineTransform3D
{
    public TranslateTransform3D();
    public TranslateTransform3D(Vector3D offset);
15 public TranslateTransform3D(Vector3D offset,
        Vector3DAnimationCollection offsetAnimations);
    public new TranslateTransform3D Copy();

    [Animations("OffsetAnimations")]
    public Vector3D Offset { get; set; }
    public Vector3DAnimationCollection OffsetAnimations { get; set; }

    public override Matrix3D Value { get; }
25 }

```

ScaleTransform3D:

```

30 public sealed class System.Windows.Media3D.ScaleTransform3D :
    AffineTransform3D
{

```

298

93

```

public ScaleTransform3D();
public ScaleTransform3D(Vector3D scaleVector);
public ScaleTransform3D(Vector3D scaleVector, Point3D scaleCenter);
public ScaleTransform3D(Vector3D scaleVector,
5      Vector3DAnimationCollection scaleVectorAnimations,
      Point3D scaleCenter,
      Point3DAnimationCollection scaleCenterAnimations);
public new ScaleTransform3D Copy();

10 [Animations("ScaleVectorAnimations")]
public Vector3D ScaleVector { get; set; }
public Vector3DAnimationCollection ScaleVectorAnimations { get; set; }

[Animations("ScaleCenterAnimations")]
15 public Point3D ScaleCenter { get; set; }
public Point3DAnimationCollection ScaleCenterAnimations { get; set; }

public override Matrix3D Value { get; }
}
20

```

RotateTransform3D is more than just a simple mapping from the 2D rotate due to the introduction of the concept of an axis to rotate around (and thus the use of quaternions):

```

25 public sealed class RotateTransform3D : AffineTransform3D
{
    public RotateTransform3D();
    public RotateTransform3D(Vector3D axis, double angle);
    public RotateTransform3D(Vector3D axis, double angle, Point3D center);

30 // Quaternions supplied to RotateTransform3D methods need to be normalized,
    // otherwise an exception will be raised.
    public RotateTransform3D(Quaternion quaternion);

```

298

914.

```
public RotateTransform3D(Quaternion quaternion, Point3D center);  
public RotateTransform3D (  
    Quaternion quaternion,  
    QuaternionAnimationCollection quaternionAnimations,  
5    Point3D center,  
    Point3DAnimationCollection centerAnimations);  
  
public new RotateTransform3D Copy();  
  
10 // Angle/Axis are just a different view on the QuaternionRotation parameter. If  
    // Angle/Axis changes, QuaternionRotation will change accordingly, and vice-versa.  
    public double Angle { get; set; }  
    public Vector3D Axis { get; set; }  
  
15 [Animations("QuaternionRotationAnimations")]  
    public Quaternion QuaternionRotation { get; set; }  
    public QuaternionAnimationCollection QuaternionRotationAnimations { get; set; }  
  
    [Animations("CenterAnimations")]  
20    public Point3D Center { get; set; }  
    public Point3DAnimationCollection CenterAnimations { get; set; }  
  
    public override Matrix3D Value { get; }  
    }  
25
```

Note that only the Quaternion property here is animatable. In general, animations of axis/angles don't tend to work out well. Better to animate the quaternion, and we can extract axes and angles from the base value of the quaternion. If you do want to simply animate an angle against a fixed axis, the easy way to specify this is to build two quaternions representing those positions, and animate between them.

299

MatrixTransform3D may build a Transform3D directly from a Matrix3D:

```

public sealed class System.Windows.Media3D.MatrixTransform3D : Transform3D
{
    public MatrixTransform3D();
5    public MatrixTransform3D(Matrix3D matrix);

    public new MatrixTransform3D Copy();

    public Matrix3D Value { get; set; }
10 }

```

When a Transform3D type property is specified in markup, the property system uses the Transform type converter to convert the string representation to the appropriate Transform derived object. Animated properties are not described using this syntax, however the complex property syntax can be used for animation descriptions.

The syntax is modeled off of the 2D Transform, wherein "<" represents an optional parameter:

- matrix(m00 m01 m02 m03 m11 ... m33)
- translate(tx ty tz)
- scale(sx <sy> <sz> <cx> <cy> <cz>)
 - If <sy> or <sz> is not specified it is assumed to be a uniform scale.
 - If <cx> <cy> <cz> are specified, then they all need to be specified, and <sx> <sy> do as well. They are used for the scaling center. If they're not, center is assumed to be 0,0,0.
- rotate(ax ay az angle <cx> <cy> <cz>)
 - ax,ay,az specifies axis of rotation
 - angle is the angle through that axis
 - If cx, cy, cz is not specified it's assumed to be 0,0,0.

The following is example grammar.

	transform-list:
	wsp* transforms? wsp*
5	transforms:
	transform
	 transform comma-wsp+ transforms
	transform:
10	matrix
	 translate
	 scale
	 rotate
	 skewX
15	 skewY
	matrix:
	"matrix" wsp* "(" wsp*
	number comma-wsp
20	number comma-wsp
	... 13 more times ...
	number wsp* ")"
	translate:
25	"translate" wsp* "(" wsp* number (comma-wsp number comma-wsp number)?
	wsp* ")"
	scale:
30	"scale" wsp* "(" wsp* number (comma-wsp number comma-wsp number
	(comma-wsp number comma-wsp number comma-wsp number)?
	)? wsp* ")"

rotate:

```
"rotate" wsp* "(" wsp* number wsp* number wsp* number wsp* number
  ( comma-wsp number comma-wsp number comma-wsp number )? wsp* ")"
```

5 Retained3DVisual derives from Visual, and in so doing gets its properties, including opacity, 2D Geometric Clip, 2D Blend Mode, Hit Testing API, 2D Bounds query and Participation in the Visual tree. Note that all of the opacity, clip, blend mode, and bounds apply to the 2D projection of the 3D scene.

```
10        public class System.Windows.Media.Visuals.Retained3DVisual : Visual
    {
        public Retained3DVisual();
        public Retained3DVisual(UIContext Context);

        // Modeling-oriented semantics. Default value is an empty Model3DCollection
15        public Model3DCollection Models { get; set; }

        // Ambient properties

        // Camera – there's no default, it's an error not to provide one.
20        public Camera Camera { get; set; }

        // ViewPort establishes where the projection maps to in 2D. Default is 0,0,1,1
        [Animation("ViewPortAnimations")]
        public Rect ViewPort { get; set; }
25        public RectAnimationCollection ViewPortAnimations { get; set; }

        public Fog Fog { get; set; }
    }
}
```

30 The ViewPort box establishes where the projection determined by the Camera/Models combination maps to in 2D local coordinate space. Note that there is no Bounds3D method on Retained3DVisual. This is available as Models.Bounds.

The Drawing3DContext very much parallels the 2D DrawingContext, and is accessible from the Model3DCollection of a Retained3DVisual via RenderOpen/RenderAppend. It resembles an immediate-mode rendering context, even though it is retaining instructions internally.

```

5 public class System.Windows.Media3D.Drawing3DContext : IDisposable
{
    internal Drawing3DContext(); // can't be publicly constructed

    // Rendering
10 public void DrawMesh(Mesh3D mesh, Material material, object hitTestToken);

    // These are for drawing imported primitives like .x files
    public void DrawImportedPrimitive(ImportedPrimitive3DSource primitiveSource,
                                     object hitTestToken);
15 public void DrawImportedPrimitive(ImportedPrimitive3DSource primitiveSource,
                                     Material overridingMaterial,
                                     object hitTestToken);

    public void DrawVisual(Visual visual, Point3D centerPosition,
20                          object hitTestToken);

    public void DrawModel(Model3D model);

    public void AddLight(Light light);
25

    // Stack manipulation
    public void PushTransform(Transform3D transform);
    public void Pop();

    public void Close(); // Also invoked by Dispose();
30 }

```

919

Specific details on the semantics of these Drawing3DContext operations are described below with reference to the Modeling API, for which the Drawing3DContext is a convenience. For example, DrawImportedPrimitive (ImportedPrimitive3DSource primitiveSource, objectHitTestToken) creates an
 5 ImportedPrimitive3D, and adds it into the currently accumulating Model3D (which in turn is manipulated by Push/Pop methods on the context).

DrawModel() is another crossover point between the "context" world and the "modeling" world, allowing a Model3D to be "drawn" into a context. There is no explicit "readback" from the Drawing3DContext, because it has the
 10 Model3DCollection backing it, and that collection can be enumerated as needed.

The modeling API is the public and protected API for these classes, (not showing inherited members), wherein Model3D is the abstract model that everything builds from:

```

15 public abstract class Model3D : Animatable
    {
        public Transform3D Transform { get; set; } // defaults to Identity
        public ShadingMode ShadingMode { get; set; }
        public object HitTestToken { get; set; }

20     public Rect3D Bounds3D { get; } // Bounds for this model

        // singleton "empty" model.
        public static Model3D EmptyModel3D { get; }

25     }
  
```

Model3DCollection is where a combination of models is constructed, and treated as a unit, optionally transforming or applying other attributes to them:

```

30 public sealed class Model3DCollection : Model3D, IList
    {
        public Model3DCollection(params Model3D[] models);
        public Model3DCollection(ICollection collection);
  
```

307

910

```

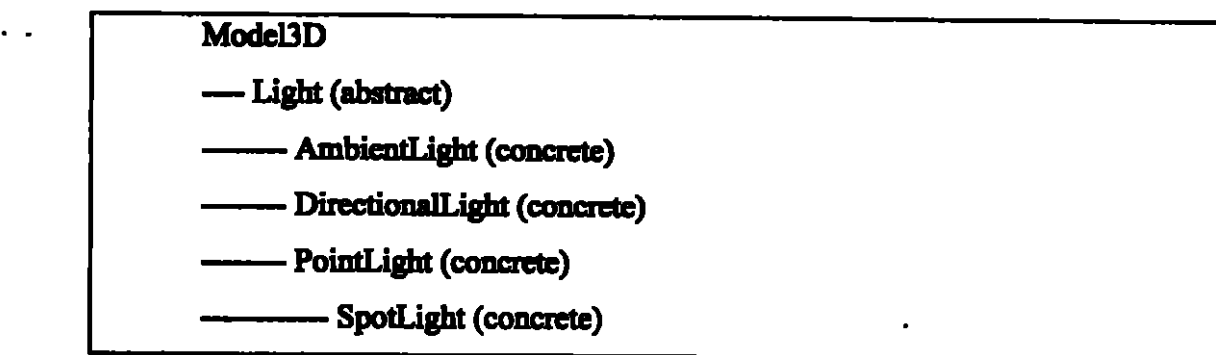
// Drawing3DContext semantics
public Drawing3DContext RenderOpen();
public Drawing3DContext RenderAppend();

5 // IList interface (bunch of stuff I'm not going into here, plus...)
public int Add(Model3D model);
public Model3D this[int index] { get; set; }
public void Insert(...);
// etc...
10 }

```

Note that Model3DCollection also has RenderOpen/Append, which returns a Drawing3DContext. Use of this context modifies the ModelCollection itself. The difference between RenderOpen() and RenderAppend() is that RenderOpen() clears out the collection first. Further note that in this implementation, only one Drawing3DContext may be open at a time on a Model3DCollection, and when it is opened, applications may not directly access (for read or write) the contents of that Model3DCollection.

Lights are Model3D's. These include Ambient, Positional, Directional and Spot lights. Lights have an additional property of being part of a modeling hierarchy, and are thus subject to coordinate space transformations. Provided are ambient, diffuse, and specular colors on lights. Note also that there are no lights that are hierarchically scoped or scoped to a particular volume. A suitable light hierarchy is set forth below:



207

921

The base light class is an abstract class:

```
public abstract class Light : Model3D
{
    internal Light(); // only allow public construction – no 3rd party lights

    [Animation("AmbientColorAnimations")]
    public Color AmbientColor { get; set; }
    public ColorAnimationCollection AmbientColorAnimations { get; set; }

    [Animation("DiffuseColorAnimations")]
    public Color DiffuseColor { get; set; }
    public ColorAnimationCollection DiffuseColorAnimations { get; set; }

    [Animation("SpecularColorAnimations")]
    public Color SpecularColor { get; set; }
    public ColorAnimationCollection SpecularColorAnimations { get; set; }
}
```

Ambient lights light objects uniformly, regardless of their shape:

```
public sealed class AmbientLight : Light
{
    public AmbientLight(Color ambientColor);
}
```

Directional lights have no position in space and project their light along a particular direction, specified by the vector that defines it:

```
public sealed class DirectionalLight : Light
{
    public DirectionalLight(Color diffuseColor, Vector3D direction); // common usage

    [Animation("DirectionAnimations")]
    public Vector3D Direction { get; set; }
}
```

922

```
public Vector3DAnimationCollection DirectionAnimations { get; set; }
}
```

The direction need not be normalized, but it needs to have non-zero magnitude.

5

Positional lights have a position in space and project their light in all directions. The falloff of the light is controlled by attenuation and range properties:

```
[ strong name inheritance demand so 3rd parties can't derive... we can't seal, since
  SpotLight derives from this ...]
10 public class PointLight : Light
    {
        public PointLight(Color diffuseColor, Point3D position); // common usage

        [Animation("PositionAnimations")]
15     public Point3D Position { get; set; }
        public Point3DAnimationCollection PositionAnimations { get; set; }

        // Range of the light, beyond which it has no effect. This is specified
        // in local coordinates.
20     [Animation("RangeAnimations")]
        public double Range { get; set; }
        public DoubleAnimationCollection RangeAnimations { get; set; }

        // constant, linear, and quadratic attenuation factors defines how the light
25     // attenuates between its position and the value of Range.
        [Animation("ConstantAttenuationAnimations")]
        public double ConstantAttenuation { get; set; }
        public DoubleAnimationCollection ConstantAttenuationAnimations { get; set; }

30     [Animation("LinearAttenuationAnimations")]
        public double LinearAttenuation { get; set; }
        public DoubleAnimationCollection LinearAttenuationAnimations { get; set; }
```

307

```

5  [Animation("QuadraticAttenuationAnimations")]
    public double QuadraticAttenuation { get; set; }
    public DoubleAnimationCollection QuadraticAttenuationAnimations { get; set; }
    }

```

Note that PointLight needs to have a strongname inheritance demand on it; to enable SpotLight to derive from it, but not third parties.

The SpotLight derives from PointLight as it has a position, range, and attenuation, but also adds in a direction and parameters to control the "cone" of the light. In order to control the "cone", outerConeAngle (beyond which nothing is illuminated), and innerConeAngle (within which everything is fully illuminated) needs to be specified. Lighting between the outside of the inner cone and the outer cone falls off linearly. (Note that there is an "angular" falloff between the edge of the inner cone and the outer cone, and a falloff in distance, relative to the position of the light, and is affected by attenuation and range.)

```

20  public sealed class SpotLight : PointLight
    {
        public SpotLight(Color color,
                          Point3D position,
                          Vector3D direction,
                          double outerConeAngle,
                          double innerConeAngle);

25  [Animation("DirectionAnimations")]
        public Vector3D Direction { get; set; }
        public Vector3DAnimationCollection DirectionAnimations { get; set; }

        [Animation("OuterConeAngleAnimations")]
30  public double OuterConeAngle { get; set; }
        public DoubleAnimationCollection OuterConeAngleAnimations { get; set; }
    }

```


924

```
[Animation("InnerConeAngleAnimations")]
public double InnerConeAngle { get; set; }
public DoubleAnimationCollection InnerConeAngleAnimations { get; set; }
}
```

5

Note that, like everywhere in the MIL API, angles are specified in degrees.

Primitive3D's are leaf nodes that result in rendering in the tree. Concrete classes bring in explicitly specified meshes, as well as imported primitives (.x files). The constructor is internal; (extensibility is available via MeshPrimitive3D).

10

```
public abstract class Primitive3D : Model3D
{
    internal Primitive3D(object hitTestToken);
}
```

15

MeshPrimitive3D is for modeling with a mesh and a material:

```
public sealed class MeshPrimitive3D : Primitive3D
{
    public MeshPrimitive3D(Mesh3D mesh, Material material, object hitTestToken);
    public Mesh3D Mesh { get; set; }

    public Material Material { get; set; }
}
```

20

25

Note that MeshPrimitive3D is a leaf geometry, and that it contains but is not itself, a Mesh. This means that a Mesh can be shared amongst multiple MeshPrimitive3D's, with different materials, subject to different hit testing, without replicating the mesh data.

30

ImportedPrimitive3D represents an externally acquired primitive (potentially with material and animation) brought in and converted into the appropriate internal form. It is treated as a rigid model. The canonical example of this is an .X File, and there is a subclass of ImportedPrimitive3DSource that explicitly imports XFiles.

309

925

```

public sealed class ImportedPrimitive3D : Primitive3D
{
    public ImportedPrimitive3D(ImportedPrimitive3DSource primitive,
        object hitTestToken);
5    public ImportedPrimitive3DSource PrimitiveSource { get; set; }

    // Allow overriding the imported material(s) if there was any. If not specified,
    // this is null, and the built in material is used.
    public Material OverridingMaterial { get; set; }
10 }

```

Since .x files are expected to be commonly included in scenes, a simple TypeConverter format for expressing this is supported, such as:

```

<ImportedPrimitive3D xfile="myFile.x" />

```

The VisualModel3D takes any Visual (2D, by definition), and places it in the scene. When rendered, it will be screen aligned, and its size will not be affected, but it will be at a particular z-plane from the camera. The Visual will remain interactive.

```

public sealed class VisualModel3D : Model3D
20 {
    public VisualModel3D(Visual visual, Point3 centerPoint, object hitTestToken);

    public Visual Visual { get; set; }
    public Point3D CenterPoint { get; set; }
25 }

```

Rendering a VisualModel3D first transforms the CenterPoint into world coordinates. It then renders the Visual into the pixel buffer in a screen aligned manner with the z of the transformed CenterPoint being where the center of the visual is placed. Under camera motion, the VisualModel3D will occupy the same amount of screen real estate, and will be forward facing, and not be affected by lights, or the like.

710

926

The fixed point during this camera motion of the visual relative to the rest of the scene will be the center of the visual, since placement happens based on that point.

The Visual provided is fully interactive, and is effectively "parented" to the Retained3DVisual enclosing it, which means that a given Visual can only be used
5 once in any VisualModel3D, just like a Visual can only have a single parent. Note that this is one of the two mechanisms for embedding interactive Visuals into 3D. The other is to use a VisualMaterial and use it on a primitive.

DrawMesh and DrawVisual both take a hitTestToken "object." This will be returned by IVisual.HitTest() when a 3D Visual gets hit. This is then used to
10 disambiguate what gets hit in 3D. Note that "pick paths" are not necessary here, since each DrawMesh, even if it gets the same mesh, can be provided with a different "hitTestToken."

In one implementation, HitTesting is implemented via the Direct3D RayIntersect utility, which provides distance to hit, u,v coordinates of hit point, allows
15 for hit testing beyond the initially hit object, and so forth. Note also that the hit testing result has sufficient information to give the u,v coordinate of the primitive that was hit, to allow translation into a hit test query on whatever texture was wrapping it. Further, for VisualModel3D, as well as the use of a VisualMaterial, the hit test will proceed into the 2D visual at the correct mapping point. This results by allowing the
20 hit test context to come in from the outer 2D environment, continue through the 3D environment, and pick up again in a nested 2D environment, potentially forever. An end result of this is that when providing this pass-through, user-interaction into live, texture mapped Avalon 2D controls operates properly.

The present invention supports a general notion of importing 3D primitives
25 from external sources. One implementation of this in v1 will be from a .x file format.

```

public abstract sealed class ImportedPrimitive3DSource
{
    // Don't allow external derivation
    internal ImportedPrimitive3DSource();
30 }

```

3.11

927

5 .X Files come into the system as an XFile3DSource, and are used to construct a ImportedPrimitive3D. It is separated out into a separate data type so that it can be used and shared in multiple ImportedPrimitive3Ds. The appropriate conversions and processing of the .x file may happen on construction of this, or may be delayed. Note that .x files are imported as a unit, and nothing inside of it is separately addressable.

```

10 public sealed class XFile3DSource : ImportedPrimitive3DSource
    {
        // .X file data is specified as a stream, allowing it to come from
        // arbitrary sources (files, resources, memory, URLs).
        public XFile3DSource(IStream xFileStream);
    }

```

15 The Mesh3D primitive is a straightforward triangle primitive (allowing both indexed and non-indexed specification) that can be constructed programmatically. Note that this supports what is expected to be the most common usage of the primitive (namely position, normal, color, and texture information, with the last three being optional). The mesh also allows selection of whether it is to be displayed as triangles, lines, or points. It also supports the three topologies for interpreting indices, namely triangle list, triangle strip, and triangle fan.

20 For vertex formats and other primitive construction that are not supported directly by Mesh3D, an .x file can be constructed and imported.

```

25 public sealed class System.Windows.Media3D.Mesh3D : Animatable
    {
        public Mesh3D();

        // Vertex data. Normals, Colors, and TextureCoordinates are all optional.
        public Point3DCollection Positions { get; set; }
        public Vector3DCollection Normals { get; set; } // assumed to be normalized
        public ColorCollection Colors { get; set; }
        public ColorCollection SpecularColors { get; set; }
        public PointCollection TextureCoordinates { get; set; }
    }

```

392

928.

```

// Topology data. If null, treat as non-indexed primitive
public IntegerCollection TriangleIndices { get; set; }

// Primitive type - default = TriangleList
5 public MeshPrimitiveType MeshPrimitiveType { get; set; }
}

```

MeshPrimitiveType is defined as:

```

10 public enum System.Windows.Media3D.MeshPrimitiveType
{
    TriangleList,
    TriangleStrip,
    TriangleFan,
15 LineList,
    LineStrip,
    PointList
}

```

20 The per-vertex data in Mesh3D is divided up into Positions, Normals, Colors, and TextureCoordinates. Of these, only Positions is required. If any of the other are provided, they need to have the exact same length as the Positions collection, otherwise an exception will be raised.

The Normals, if provided, are assumed to be normalized. Note that the system
25 will not attempt to calculate normals based on topology/proximity; instead, when normals are desired, they need to be supplied.

The TriangleIndices collection has members that index into the vertex data to determine per-vertex information for the triangles that compose the mesh. This collection is interpreted based upon the setting of MeshPrimitiveType. For
30 TriangleList, every three elements in the TriangleIndices collection defines a new triangle. For TriangleFan, indices 0,1,2 determine the first triangle, then each subsequent index, *i*, determines a new triangle given by vertices 0,*i*,*i*-1. For

313

929

TriangleStrip, indices 0,1,2 determine the first triangle, and each subsequent index *i* determines a new triangle given by vertices *i-2*, *i-1*, and *i*. LineList, LineStrip, and PointList have similar interpretations, but the rendering is in terms of lines and points, rather than triangles.

5 If TriangleIndices is null, then the Mesh is implemented as a non-indexed primitive, which is equivalent to TriangleIndices holding values 0,1,...,n-2,n-1 for a Positions collection of length n.

 Upon construction of the Mesh, the implementation creates the optimal D3D structure that represents this mesh. At this point, the actual Collection data structures
10 can be thrown away by the Mesh implementation to avoid duplication of data. Subsequent readback of the mesh if accessed in through some other mechanism (traversing the Retained3DVisuals model hierarchy for instance) will likely reconstruct data from the D3D information that is being held onto, rather than retaining the original data.

15 The mesh derives from Changeable (through Animatable), and thus can be modified. The implementation will need to trap sets to the vertex and index data, and propagate those changes to the D3D data structures. There is no explicit support for declarative animations of the vertex or index data in a mesh. In other words, Point3DAnimationCollections are not seen here, for instance. This is consistent with
20 the 2D polyline paths, for example.

 Like other types, the XAML complex property syntax can be used to specify the collections that define Mesh3D. However, that can be unwieldy and verbose, and thus TypeConverters are provided to make the specification more succinct.

 Each collection defined in mesh can take a single string of numbers to be
25 parsed and used to create the collection. For instance, a Mesh representing an indexed triangle strip with only positions and colors could be specified as:

```

30       <Mesh3D
          meshPrimitiveType="TriangleStrip"
          positions="1,2,3, 4,5,6, 7,8,9, 10,11,12, 13,14,15, 16,17,18"
          colors="red blue green cyan magenta yellow"
          triangleIndices="1,3,4,1,2,3,4,5,6,1,2,4,2"
         />

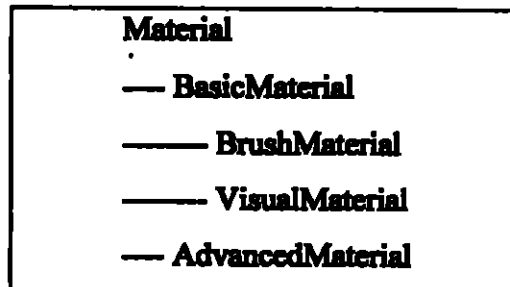
```

319

930

Of course, any of these could be represented much more verbosely in the complex property syntax.

The methods that construct Primitive3D's take a Material to define their appearance. Material is an abstract base class with three concrete subclasses: BrushMaterial, VisualMaterial, and AdvancedMaterial. BrushMaterial and VisualMaterial are both subclasses of another abstract class called BasicMaterial. Thus:



The BrushMaterial method takes a single Brush and can be used for a wide range of effects, including achieving transparency (either per-pixel or scalar), having a texture transform (even an animate one), using video textures, implicit auto-generated mipmaps, and so forth. Specifically, for texturing solid colors, images, gradients, or even another Visual, a SolidColorBrush, ImageBrush, GradientBrush, or VisualBrush would be used to create the BrushMaterial.

The VisualMaterial method is designed to construct a material out of a Visual. This material will be interactive in the sense that input will pass into the Visual from the 3D world in which it is embedded. One difference between this and a BrushMaterial with a VisualBrush is that the BrushMaterial is non-interactive.

The AdvancedMaterial class, whiles considerably more complex than simply using a BrushMaterial or VisualMaterial, provides for further flexibility.

```
public abstract class Material : Animatable
{
    internal Material(); // don't allow external subclassing

    public new Material Copy(); // shadows Changeable.Copy()
```

318

```

    public static Material Empty { get; } // singleton material
}

public abstract class BasicMaterial : Material
5 {
    internal BasicMaterial(); // don't allow external subclassing

    public new BasicMaterial Copy(); // shadows Changeable.Copy()
}
10

```

Materials gain flexibility and "economy of concept" by being based on a Brush. For example, there need not be a separate Texture hierarchy reflecting things like video textures, gradient textures, or the like, since those are all specifiable as Brushes. Brushes already encapsulate both alpha-mask and scalar opacity values, so those both become available to texturing. Brushes already have a 2D Transform associated with them which, in the case of texturing, will be interpreted as a texture transform for transforming uv coordinates in the mesh to map into the texture. Brushes are the right place to hang stock procedural shaders such as a wood grain shader. This would then be usable in 2D as a fill or pen, and in 3D as a texture. No specific API support need be given in the 3D space for procedural shaders.

Note that pixel or vertex shaders may be provided as "stock" shaders, many of which will be parameterized. The way that these will be accessible in the API is that for shaders that make sense in the 2D world, they will be exposed as concrete subclasses of Brush, with their parameterization expressed either through the constructors on the class, or as properties on the class. They can then be applied to 2D objects. Shaders that only make sense in 3D will be exposed as concrete subclasses of Material (perhaps as subclasses of BasicMaterial), where they can also be parameterized through their constructor. This exposure will then allow the shaders to be applied to 3D (and 2D where appropriate) meshes.

As described above, BrushMaterial encapsulates a Brush. A BrushMaterial applied to a Primitive3D is treated as a texture. Textures will be mapped directly - that is, the 2D u,v coordinates on the primitive being mapped will index directly into

932

the correspond x,y coordinates on the Texture, modified by the texture transform. Note that, like other 2D, the texture's coordinate system runs from (0,0) at the top left with positive y pointing down.

5 A VisualBrush used for the Brush will not accept input, but it will update according to any animations on it, or any structural changes that happen to it. To use a Visual as a Material and still receive input, the VisualMaterial is used, as described herein.

```

10 public sealed class BrushMaterial : BasicMaterial
    {
        public BrushMaterial(Brush brush);

        public new BrushMaterial Copy(); // shadows Material.Copy()

        public Brush Brush { get; set; }

15 // Additional texturing specific knobs.
    }

```

20 As described above, VisualMaterial encapsulates an interactive Visual. This differs from BrushMaterial used with a Visual in that the Visual remains live in its textured form. Note that since the mapping is based on the u,v grid associated with a primitive, there are situations where user-input will not have full fidelity with its 2D counterpart. For example, consider a mouse-capture scenario where a 2D control captures the mouse (for implementing scroll bar behavior, for instance). In these

25 scenarios, the mouse is allowed to stray from the scroll bar, but the system is able to figure out what, for instance, y position the mouse is at with respect to the scroll bar and operate properly. In a situation where the control is textured onto a primitive with a u,v grid, and the mouse strays off of the primitive, then in general, there is no way to make this determination. There are certain things that can be done while authoring to

30 counteract this, but in general, this is going to be a limitation of mapping onto a 2D space. Note that the Visual is then, in effect, parented in some fashion to the root Retained3DVisual. In one implementation, it is illegal to use a single UiElement in

318

933

more than one Material, or to use a VisualMaterial in more than one place, due to the fact that the framework is based on single parenting of controls.

```

public sealed class VisualMaterial : BasicMaterial
{
    public VisualMaterial(Visual visual);

    public new VisualMaterial Copy(); // shadows Changeable.Copy()

    public Visual Visual { get; set; }

    --(need to add viewport/viewbox stuff for positioning...)

    // Additional texturing specific knobs.
}

```

BrushMaterials/VisualMaterials and BumpMaps are used to define AdvancedMaterials:

```

public class AdvancedMaterial : Material
{
    public AdvancedMaterial();

    // Add common constructors.

    public new AdvancedMaterial Copy(); // shadows Changeable.Copy()

    public BasicMaterial DiffuseTexture { get; set; }
    public BasicMaterial SpecularTexture { get; set; }
    public BasicMaterial AmbientTexture { get; set; }
    public BasicMaterial EmissiveTexture { get; set; }

    [ Animations("SpecularPowerAnimations") ]
    public double SpecularPower { get; set; }
}

```

318

```

public DoubleAnimationCollection SpecularPowerAnimations { get; set; }

public BumpMap DiffuseBumpMap { get; set; }
public BumpMap ReflectionBumpMap { get; set; }
5 public BumpMap RefractionBumpMap { get; set; }

public BrushMaterial ReflectionEnvironmentMap { get; set; }
public BrushMaterial RefractionEnvironmentMap { get; set; }
}
10

```

Note that the EnvironmentMaps are textures that are expected to be in a particular format to enable cube-mapping. Specifically, the six faces of the cube map will need to be represented in the sections of the Brush associated with the Texture (likely something like a 3x2 grid on the Brush). Note that the Ambient, Diffuse, and Specular properties take a BasicMaterial, and not a general Material since they're not allowed to be specified as AdvancedMaterials themselves. Further, note that the environment maps are BrushMaterials exclusively, since the interactivity that VisualMaterial provides does not make sense for environment maps.

Bump maps are grids that, like textures, get mapped onto 3D primitives via texture coordinates on the primitives. However, the interpolated data is interpreted as perturbations to the normals of the surface, resulting in a "bumpy" appearance of the primitive. To achieve this, bump maps carry information such as normal perturbation, and potentially other information. They do not carry color or transparency information. Because of this, it is inappropriate to use a Brush as a bump map. Therefore, a new BumpMap class may be provided, as an ImageSource of a particular pixel format.

Material provides a simple TypeConverter that allows the string specification of a Brush to automatically be promoted into a BrushMaterial:

```

Material:
30 ... delegate to Brush type converter ...

```

This allows specifications such as:

935

```
<MeshPrimitive3D ... Material="yellow" />
```

```
<MeshPrimitive3D ... Material="LinearGradient blue green" />
```

```
5 <MeshPrimitive3D ... Material="HorizontalGradient orange purple" />
```

```
<MeshPrimitive3D ... Material="*Resource(myImageResource)" />
```

Also compound property notation,

10

```
<MeshPrimitive3D>
```

```
<MeshPrimitive3D.Material>
```

```
<LinearGradientBrush>
```

```
.....
```

15

Ambient parameters of the model, which are not embeddable at arbitrary levels in the geometric hierarchy, may be provided. For example, fog can be added to the scene by setting a Fog property on the Retained3DVisual, e.g., the Fog available is "pixel fog." Fog is represented as an abstract class, and hierarchy as shown below:

20

```
public abstract class Fog : Animatable
```

```
{
```

```
    // only constructable internally
```

```
    internal Fog(Color color);
```

```
    public new Fog Copy(); // hides Changeable.Copy()
```

25

```
    [Animation("ColorAnimations")]
```

```
    public Color Color { get; set; }
```

```
    public ColorAnimationCollection ColorAnimations { get; set; }
```

30

```
    // singleton representation of "no fog"
```

```
    public static Fog NoFog { get; }
```

```
}
```

320

936.

```

public sealed class LinearFog : Fog
{
    public LinearFog(Color color, double fogStart, double fogEnd);

5    [Animation("FogStartAnimations")]
    public double FogStart { get; set; }
    public DoubleAnimationCollection FogStartAnimations { get; set; }

    [Animation("FogEndAnimations")]
10    public double FogEnd { get; set; }
    public DoubleAnimationCollection FogEndAnimations { get; set; }
}

public sealed class ExponentialFog : Fog
{
15    public ExponentialFog(Color color, double fogDensity, bool squaredExponent);

    [Animation("FogDensityAnimations")]
    public double FogDensity { get; set; }
    public DoubleAnimationCollection FogDensityAnimations { get; set; }

20    public bool SquaredExponent { get; set; }
}

```

25 The fogDensity ranges from 0-1, and is a normalized representation of the density of the fog. The fogStart and fogEnd comprise z-depths specified in device space [0,1] and represent where the fog begins and ends.

The Camera is a mechanism by which a 3D model is projected onto a 2D visual. The Camera itself is an abstract type, with two concrete subtypes – ProjectionCamera and MatrixCamera. ProjectionCamera takes well-understood parameters such as Position, LookAtPoint, and FieldOfView to construct the Camera. MatrixCamera takes a Matrix3D used to define the World-To-Device transformation.

```

public abstract class Camera : Animatable

```

937

```

{
    // Only allow to be built internally.
    internal Camera();

5    public new Camera Copy(); // hides Changeable.Copy()
}

```

In a Retained3DVisual, a Camera is used to provide a view onto a Model3D, and the resultant projection is mapped into the 2D ViewPort established on the Retained3DVisual. Note that the 2D bounding box of the Retained3DVisual will simply be the projected 3D bbox of the 3D model, wrapped with its convex, axis-aligned hull, clipped to the clip established on the visual.

The ProjectionCamera is the means by which a camera is constructed from parameters such as Position, LookAtPoint, and FieldOfView. It encapsulates both perspective projections and orthographic projections. FIG. 42 provides a good indication of the relevant aspects of a ProjectionCamera showing viewing and position, (where FieldOfView should be in the horizontal direction).

```

public class ProjectionCamera : Camera
{
20    // Common constructors
    public ProjectionCamera();
    public ProjectionCamera(Point3D position,
                           Vector3D lookDirection,
                           Vector3D Up,
25    double fieldOfView);

    public new ProjectionCamera Copy(); // hides Changeable.Copy()

    // Camera data
30    public bool IsOrthographic { get; set; } // default == false: Perspective

    [Animations("NearPlaneDistanceAnimations")]
}

```

322

```

public double NearPlaneDistance { get; set; } // default = 0
public DoubleAnimationCollection NearPlaneDistanceAnimations { get; set; }

[Animations("FarPlaneDistanceAnimations")]
5 public double FarPlaneDistance { get; set; } // default = infinity
public DoubleAnimationCollection FarPlaneDistanceAnimations { get; set; }

[Animations("FieldOfViewAnimations")]
public double FieldOfView { get; set; }
10 public DoubleAnimationCollection FieldOfViewAnimations { get; set; }

[Animations("PositionAnimations")]
public Point3D Position { get; set; }
public Point3DAnimationCollection PositionAnimations { get; set; }
15

[Animations("LookDirectionAnimations")]
public Vector3D LookDirection { get; set; }
public Vector3DAnimationCollection LookDirectionAnimations { get; set; }

20 [Animations("UpAnimations")]
public Vector3D Up { get; set; }
public Vector3DAnimationCollection UpAnimations { get; set; }
}

```

25 Note that the FieldOfView represents the horizontal field of view, and is specified in degrees (like other MIL angles). The Near and Far PlaneDistances represent 3D world-coordinate distances from the camera's Position along the LookDirection vector. The NearPlaneDistance defaults to zero and the FarPlaneDistance defaults to infinity.

30 Upon actual projection, if the NearPlaneDistance and FarPlaneDistance are still zero and infinity, respectively, then the model is examined and its bounding volume is projected according to the camera projection. The resulting bounding

939

volume is then examined so that the near plane distance is set to the bounding volume's plane perpendicular to the LookDirection nearest the camera position. This is essentially the same for the far plane, but using the farthest plane. This results in optimal use of z-buffer resolution while still displaying the entire model. If
 5 IsOrthographic is true, then an orthographic projection is used, and FieldOfView is ignored.

Note that the "projection plane" defined by the parameters of the ProjectionCamera is then mapped into the ViewPort rectangle on the Retained3DVisual, and that represents the final transition from 3-space to 2-space.

10 The MatrixCamera subclass of Camera provides a means for directly specifying a Matrix as the projection transformation. This is useful for applications that have their own projection matrix calculation mechanisms.

```

public class MatrixCamera : Camera
{
15 // Common constructors
    public MatrixCamera();
    public MatrixCamera(Matrix3D matrix);

    public new MatrixCamera Copy(); // hides Changeable.Copy()
20
    // Camera data
    public Matrix3D Matrix { get; set; } // default = identity
}
  
```

25 Note that the MinimumZ and MaximumZ values directly control the z-buffer range. They are considered after the application of the Matrix as the projection matrix.

The resultant projection is mapped into the ViewPort rectangle on the Retained3DVisual, and that represents the final transition from 3-space to 2-space.

The following are more complete markups showing specification of an entire
 30 Model3D hierarchy in XAML. Note that some of the syntax may change.

324

440

The following example simply creates a Model with two imported .x files and a rotation transform (about the z-axis by 45 degrees) one on of them, and a single white point light sitting up above at 0,1,0.

```
5  <Model3DCollection>

    <!-- Model children go as children here -->
    <PointLight position="0,1,0" diffuseColor="white" />

    <ImportedPrimitive3D xfile="myFile.x" />

10  <Model3DCollection transform="rotate(0, 0, 1, 45), scale(2)" >
    <ImportedPrimitive3D xfile="mySecondeFile.x" />
    </Model3DCollection>

15 </Model3DCollection>
```

This markup may be in a file, a stream, a resource or any suitable entity. A client program will invoke loading of that XAML, and that will in turn construct a complete Model3DCollection, to be used by the application as desired.

20 The following example provides an explicitly declared MeshPrimitive3D, through the use of the complex-property XAML syntax. The mesh will be textured with a LinearGradient from yellow to red. There is also a light in the scene.

```
25 <Model3DCollection>

    <!-- Model children go as children here -->
    <PointLight position="0,1,0" diffuseColor="white" />

    <MeshPrimitive3D material="LinearGradient yellow red">
    <MeshPrimitive3D.Mesh>
30  <Mesh3D
    meshPrimitiveType="TriangleStrip"
    positions="1,2,3, 4,5,6, 7,8,9, 10,11,12, 13,14,15, 16,17,18"
```

941

5

```

        normals="... sensible normal vectors ..."
        textureCoordinates=".5,.5, 1,1, 0,0, .25,.25, .3,.4, .7,.8"
        triangleIndices="1,3,4,1,2,3,4,5,6,1,2,4,2" />
    </MeshPrimitive3D.Mesh>
</MeshPrimitive3D>

</Model3DCollection>

```

The example takes the first .x file and adds in a XAML-specified animation.
 10 This particular example adds a uniform scale that scales the .x file from 1x to 2.5x over 5 seconds, reverses, and repeats indefinitely. It also uses acceleration/deceleration to slow-in/slow-out of its scale:

15

20

25

30

```

<Model3DCollection>

    <!-- Model children go as children here -->
    <PointLight position="0,1,0" diffuseColor="white" />

    <ImportedPrimitive3D xfile="myFile.x">
        <ImportedPrimitive3D.Transform>
            <ScaleTransform3D>
                <ScaleTransform3D.ScaleVector>
                    <VectorAnimation
                        from="1,1,1"
                        to="2.5,2.5,2.5"
                        begin="immediately"
                        duration="5"
                        autoReverse="true"
                        repeatDuration="indefinite"
                        acceleration="0.1"
                        deceleration="0.1" />
                </ScaleTransform3D.ScaleVector>
            </ScaleTransform3D>
        </ImportedPrimitive3D.Transform>
    </ImportedPrimitive3D>

```

328

942

```

</ImportedPrimitive3D.Transform>
</ImportedPrimitive3D>

</Model3DCollection>

```

5

This example imports a .x file and applies a live UI as its material:

```

<Model3DCollection>

  <!-- Model children go as children here -->
  <PointLight position="0,1,0" diffuseColor="white" />

  <ImportedPrimitive3D xfile="myFile.x" >
    <ImportedPrimitive3D.OverridingMaterial>
      <VisualMaterial>
        <Button Text="Press Me" OnClick="button_OnClick" />
      </VisualMaterial>
    </ImportedPrimitive3D.OverridingMaterial>
  </ImportedPrimitive3D>

</Model3DCollection>

```

10

15

20

XAML supports syntax for referencing resources. "Resources" in this context is a broad term, and it implies both a "resource" section of a XAML file, as well as loading from the managed resource "fork" of a managed assembly. Both mechanisms are supported, and resources can be referenced from multiple places in order to share them. There is currently no mechanism for directly loading native Win32 resources, however, implementation of an IResourceLoader interface allows for that.

25

Although this is not directly related to XAML, it is related to how applications will be constructed that share things like meshes and materials. Note that the 3D "modeling" objects (Lights, Model3DCollections, MeshPrimitive3D's, Materials, Mesh3D, Brush, and the like) are of type Changeable. As described above, changeable values provide a Copy() method that conceptually make a deep copy, but

30

327

943
in fact do on demand shallow-cloning. Use of this Copy() method results in optimal sharing of objects while still allowing subparts of those objects to change.

Two-dimensional "shapes", like Rectangle, Polygon, Ellipse, and the like, are introduced into the Windows Presentation Framework via the Shape class, which is itself a UiElement. Shapes like Rectangle can be placed on a Canvas, which itself is a UiElement intended for use as a container of shapes. The following is one example:

```
10 <Canvas Background="black" Top="100" Left="100" Height="600" Width="800">
    <Rectangle Top="600" Left="100" Width="100" Height="50"
        Fill="red" Stroke="blue" StrokeWidth="10"/>
    <Line x1="100" y1="300" x2="300" y2="100" Stroke="green" StrokeWidth="5" />
</Canvas>
```

In this case, the Canvas is a UiElement, and the Rectangle and Line are each UiElements. Everything that is contained in the Canvas, in this case, is a UiElement.

15 However, consider combining Paths in markup, as this example shows:

```
20 <Canvas ...>
    <Path Fill="Red" Stroke="Blue">
        <Path.Data>
            <CircleGeometry ... />
            <RectangleGeometry ... />
            <PathGeometry ... />
        </Path.Data>
    </Path>
</Canvas>
```

25 In this case, the CircleGeometry, RectangleGeometry, (i.e., what is inside Path.Data), themselves are not UiElements, but the Path container is.

An element level "3D Shape" poses a problem in that the 3D characteristics of a Retained3DVisual does not go beyond the Visual itself. That is, the contents of two Visual's will never interpenetrate in z (e.g., one cannot have an arrow visual piercing an apple visual). Since Controls and Shapes derive from Visual, this also means that

there is no support for having interpenetrating Controls or Shapes. They are always 2D projections of their content.

To have 3D "objects" interpenetrate at the Element level, and also to express that in markup, a Viewport3D element is provided that can contain a complex 3D model with the needed interpenetration. However, it will not interact in z with any other Viewport3Ds (and is called a Viewport3D because it is a 2D viewport into 3D space).

Viewport3D derives from the same class that Shape derives from (either FrameworkElement or Control). Note that Viewport3D should not derive from Shape, since Shape has properties like Fill, Stroke, and so on which do not make sense for 3D. Thus, Viewport3D, Shape, and Image are all siblings. Viewport3D will have a number of properties that can be set, including some of the properties that exist on Shape (Top, Left, Width, Height, Transform and Clip). These properties, while they not relevant in 3D, do make sense for the 2D projection of 3D, which is what Viewport3D, being a UiElement, is.

Viewport3D will also have a Camera associated with it (and a reasonable default will be provided) as well as 3D scene-wide properties like fog (i.e., properties that are also on Retained3DVisual). Viewport3D also has a Model3DCollection-valued property (which is itself a Model3D) that represents a collection of 3D objects to be presented through the shape. This model can also include lights. Whatever an application wants to allow to interact in z will be placed into this single Model3DCollection for a single Viewport3D.

If the camera is not specified, a default one will be used that provides a projection such that the Viewport3D bounds determined by Top, Left, Width, Height are filled with the model, the near/far clip planes are set to just in front of and just behind the model, the camera is looking down the z axis, with y being up. If no lights are specified, a white directional light pointing down the y-axis will be implicitly used.

The markup for Viewport3D is similar that for Path shown above, since that is an example where non-UiElements are specified inside of the Path markup. The following is an example of Viewport3D markup:

```
<Canvas ...>
```

945

```

<!-- 2D elements go here, and Viewport3D is a 2D element. -->
<Viewport3D optionalCamera="..." >
  <XFilePrimitive path="..." />

  5  <Mesh ...>
      <BrushMaterial>
        <SolidColorBrush color="yellow">
      </BrushMaterial>
    </Mesh>

  10

  <Mesh ...>
    <VisualMaterial>
      <Button ... />
    </VisualMaterial>
  15 </Mesh>

  <SpotLight ... />

  20

  <Model3DCollection Transform="3DTransform">
    <Mesh ... />
    <More models, meshes, lights, groups, etc./>
  </Model3DCollection>
  25 </Viewport3D>

  <!-- More 2D elements -->
  <Rectangle ... />
  </Canvas>
  30

```

Inside the <Viewport3D> tags, the Model3DCollection-valued member is being populated from individual, markup-specified Model3D's. Note that there is a

one-to-one mapping between the tags here and the subclasses of Model3D . Further, note that the second Mesh in this example has a Visual material comprising a markup specified control (a Button in this case). This control will be fully interactive, and can specify an OnClick and the like.

5 The direct child nodes of Viewport3D are Model3D's, because the "default" collection is the Models collection. The syntax of this may change generally to require specification of <Viewport3D.Model> in complex-property syntax. This markup will follow the same path for resources that regular 2D markup follows, e.g., to define a Material and use it in multiple places. That functionality is provided by the
10 general resource mechanism.

 Generally, hit testing occurs at the UiElement or Control level. However, hit testing would be limited on Viewport3D as a whole, or otherwise to hit test would require splitting models up into multiple Viewport3D's and hit test on them. However, as soon as they are in multiple Viewport3D's, they no longer are part of the
15 same 3D environment. This is resolved this by allowing an event handler to be specified on individual members of a Model3D in the markup or code, as well as an "id". When hit testing resolves, the appropriate method will be invoked (the sender will still be the Viewport3D as a whole), and the identifier will come down as part of the EventArgs. Applications should then be able to isolate their logic either in
20 different event handlers, or by switching off of the "id".

 The use of a VisualMaterial is an indication to the system that hit testing should proceed into that embedded element. The system ensures that this does, in fact, happen. Thus the hit test operation transitions from 2D into 3D and back into 2D.

 The following is an API specification for Viewport3D:

```
25   public class Viewport3D : UiElement // (or Control / FrameworkElement)
    {
        // Stock 2D properties
        public BoxUnit Top { get; set; }
        public BoxUnit Left { get; set; }
30   public BoxUnit Width { get; set; }
        public BoxUnit Height { get; set; }
        public Transform Transform { get; set; }
```

```

public Geometry Clip { get; set; }

// 3D scene-level properties
public Fog Fog { get; set; }
5 public Camera Camera { get; set; } // have good default

// The 3D Model itself
public Model3D Model { get; set; }
10 }

```

To do transparency correctly in 3D, primitives (and, in the limit, individual triangles) are sorted to render back to front, whereby the z-buffer does not participate. This is necessary to insure correct blending.

15 An **EmbeddingHostVisual** is also provided.. This is a subclass of **Visual** that cannot be constructed by external parties, and can only be built internally. **EmbeddingHostVisual** contains a **Visual** that represents that the specified **Visual** is being used as an "embedding host" for some **Visual-to-Visual** relationship other than parent/child. 2D **Visuals** that are embedded in a 3D scene are another example of this.

20 Effects may be another example of one non-parent/child relationship between **Visuals**. A definition is set forth below:

```

public class EmbeddingHostVisual : Visual
{
    internal EmbeddingHostVisual(Visual hostingVisual); // can't be externally built
25 public Visual HostingVisual { get; }

    // All IVisual read services will result in exceptions raised.
}

```

30 Because the relationship to an **EmbeddingHostVisual** "parent" is defined to not be the typical parent/child relationship, that means that read services like extracting a **Transform** between **Visuals** do not make sense because there is not a uniform

448

mapping. These operations are handled in the 3D subsystem itself, but note that the mechanism is unique to Retained3DVisual.

MEDIA

- 5 **MediaData** can be used to play any audio/video content via the following object, also described in the attached Appendix:

```
public System.Windows.Media.MediaData : Changeable, IDisposable,
ITimingControl
{
    public MediaData (string source);

    //
    // MediaData properties
    //
    public MediaState State { get; }
    public Time MediaDuration { get; }
    public bool HasAudio { get; }
    public bool HasVideo { get; }
    public long Width { get; }
    public long Height { get; }
    public double Volume { get; set; }
    public bool Mute { get; set; }

    //
    // MediaData Methods
    //
    public void Play();

    //
    // Changeable
    //
    public bool IsChangeable { get; } // defaults to true
```

333

949

```

public Changeable Copy();
public void MakeUnchangeable();
public UseStatus StatusOfNextUse { get; set; } // defaults to false
public event EventHandler Changed { add; remove; }
public UIContext UIContext { get; }

//
// IDisposable interface
//
public void Dispose();

//
// ITimingControl interface
//
double Acceleration { get; set; }
bool AutoReverse { get; set; }
TimeSyncValue Begin { get; set; }
double Deceleration { get; set; }
Time Duration { get; set; }
TimeSyncValue End { get; set; }
TimeEndSync EndSync { get; set; }
TimeFill Fill { get; set; }
TimeFill FillDefault { get; set; }
Timeline ParentTimeline { get; set; }
double RepeatCount { get; set; }
Time RepeatDuration { get; set; }
TimeRestart Restart { get; set; }
TimeRestart RestartDefault { get; set; }
double Speed { get; set; }
int CurrentRepeat { get; }
Time CurrentTime { get; }
bool IsForwardProgressing { get; }

```

A Rect contains the points inside it, as well as all points lying on its edges. Thus, the edges of a Rect are inclusive. As a consequence, a Rect of 0 Width or 0 Height is not empty, as it contains the set of points found on the 1-dimensional line segment represented by the Rect. If both Width and Height are 0, then the Rect contains the one Point at Location. This means that the truly empty Rect is a special value, and can be accessed via the static EmptyRect property. The Empty Rect's Width and Height properties will return negative infinity (the only case where either can be negative), the X and Y properties will return positive infinity, and none of these properties will be modifiable. This is to ensure that there is not an "empty" width but normal height, or vice versa.

```

public struct System.Windows.Rect
{
    public Rect (); // Sets all values to 0
    public Rect (Point location, Size size);
    public Rect (double x, double y, double width, double height);

    // Both of these will normalize the rectangle
    public Rect (Point point1, Point point2);
    public Rect (Point point, Vector vector);

    public static Rect FromLTRB(double left, double top, double right, double
bottom);

    public static Rect Empty { get; } // returns the Empty Rect
    public bool IsEmpty { get; }

    public static Rect Infinite { get; }

    public Point Location { get; set; }
    public Size Size { get; set; }

```

951

```

public double X { get; set; }
public double Y { get; set; }
public double Width { get; set; }
public double Height { get; set; }

public double Left { get; }    // Alias to X
public double Top { get; }     // Alias to Y
public double Right { get; }   // X + Width
public double Bottom { get; }  // Y + Height

public Point TopLeft { get; }
public Point TopRight { get; }
public Point BottomLeft { get; }
public Point BottomRight { get; }
public Point Center { get; }

public bool Contains(Point point);
public bool Contains(Rect rect);
public bool Contains(double x, double y);

public bool IntersectsWith(Rect rect);

public void Intersect(Rect rect);
public static Rect Intersect(Rect rect1, Rect rect2);

public void Union(Rect rect);
public static Rect Union(Rect rect1, Rect rect2);
public void Union(Point point);
public static Rect Union(Rect rect, Point point);

...

public void Offset(Vector offset);
public void Offset(double dx, double dy);

```

229

952

```

public static Rect Offset(Rect rect, Vector offset);
public static Rect Offset(Rect rect, double dx, double dy);

public void Inflate(Size size);

// If -width > Width*2 or -height > Height*2, the Rect becomes Empty
public void Inflate(double width, double height);

public static Rect Inflate(Rect rect, double width, double height);
public static Rect Inflate(Rect rect, Size size);

// Transforms 4 corners and sets new axis aligned rect.
public void Transform(Matrix matrix);
public static Rect Transform(Rect rect, Matrix matrix);

public static Boolean Equals(Rect rectangle1, Rect rectangle2)
public virtual Boolean Equals(Object o)
public virtual Int32 GetHashCode()

public static Boolean operator==(Rect rectangle1, Rect rectangle2)
public static Boolean operator!=(Rect rectangle1, Rect rectangle2)
public virtual String ToString()
}

```

As shown, the rect provides a number of methods. IsEmpty returns true if the instance is equal to the Empty Rect. The FromLTRB method essentially returns a rectangle initialized with (left, top, right-left, bottom-top). Then Contains method

5 returns true if The Rect is not Empty, $p.x \geq r.x$, $p.x \leq r.x + r.width$, $p.y \geq r.y$ and $p.y \leq r.y + r.height$. This means that this method will return false if the rectangle has a negative width or height.

IntersectsWith returns true if neither Rect is Empty, $r1.x \leq r2.x + r2.width$, $r2.x \leq r1.x + r1.width$, $r1.y \leq r2.y + r2.height$, and $r2.y \leq r1.y + r1.height$. The

953

intersection of two rectangles is computed by taking the max of the left and top dimensions and the min of the right and bottom dimensions. If either Rect is Empty, the Empty Rect is returned.

5 The union of two rectangles is computed by taking the min of the left and top dimensions and the max of the right and bottom dimensions. If either Rect is Empty, the other Rect is returned.

For the Offset method, the offset is simply added to the location of the rectangle. This method has no effect if the Rect is Empty.

10 For Inflate, the inflation amount is simply applied equally to all sides. Adjustments are made, comprising, $r.x = r.x - s.width$; $r.y = r.y - s.height$; $r.width = r.width + 2 * s.width$ and $r.height = r.height + 2 * s.height$. This method has no effect if the Rect is Empty, because an Empty Rect has no location.

15 The following markup syntax specifies the x, y, width and height. It should be noted that this is equivalent to what the type converter for System.Drawing.Rectangle does.

```
rectangle:
    ( coordinate comma-wsp ){3} coordinate | "Empty"
```

20 Matrices for 2D computations are represented as a 3x3 matrix. The MIL uses row-vector syntax. The mil only allows affine transforms, and thus only requires only six values, rather than a full 3x3 matrix. These are named and defined as follows.

$$\begin{bmatrix} m11 & m12 & 0 \\ m21 & m22 & 0 \\ offsetX & offsetY & 1 \end{bmatrix}$$

25 When a matrix is multiplied with a point, it transforms that point from the new coordinate system to the previous coordinate system:

$$\begin{bmatrix} X_{newCoordSys} & Y_{newCoordSys} & 1 \end{bmatrix} \cdot \begin{bmatrix} m11 & m12 & 0 \\ m21 & m22 & 0 \\ offsetX & offsetY & 1 \end{bmatrix} = \begin{bmatrix} X_{oldCoordSys} & Y_{oldCoordSys} & 1 \end{bmatrix}$$

276

Transforms can be nested to any level. Whenever a new transform is applied it is the same as pre-multiplying it onto the current transform matrix:

$$\begin{bmatrix} x_{newCoordSys} & y_{newCoordSys} & 1 \end{bmatrix} \cdot \begin{bmatrix} m11_1 & m12_1 & 0 \\ m21_1 & m22_1 & 0 \\ offsetX_1 & offsetY_1 & 1 \end{bmatrix} = \begin{bmatrix} m11_1 & m12_1 & 0 \\ m21_1 & m22_1 & 0 \\ offsetX_1 & offsetY_1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{oldCoordSys} & y_{oldCoordSys} & 1 \end{bmatrix}$$

- 5 Most places in the API do not take a Matrix directly. Instead, the Transform class supports animation in a deep way.

```
public struct System.Windows.Media.Matrix
{
    // Construction and setting
    public Matrix(); // defaults to identity

    public Matrix(
        double m11, double m12,
        double m21, double m22,
        double offsetX, double offsetY);

    // Identity
    public static Matrix Identity { get; }
    public void SetIdentity();
    public bool IsIdentity { get; }

    // Unset support
    public static Matrix Unset { get; } // all values are NaN
    public bool IsSet { get; } // false if any NaN

    // Math operations
    public void Prepend(Matrix matrix); // "this" becomes: matrix * this
    public void Append(Matrix matrix); // "this" becomes: this * matrix
}
```

955

```
// Matrix operations
```

```
public void Rotate(double angle); // Appends a rotate
```

```
public void RotatePrepend(double angle); // Prepends a rotate
```

```
public void RotateAt(double angle, Point center); // Appends a rotate
```

```
public void RotateAtPrepend(double angle, Point center); // Prepends a rotate
```

```
public void Scale(Double scaleX, Double scaleY); // Appends a scale
```

```
public void ScalePrepend(Double scaleX, Double scaleY); // Prepends a scale
```

```
public void ScaleAt(Double scaleX, Double scaleY, Point point); // Appends a  
scale
```

```
public void ScaleAtPrepend(Double scaleX, Double scaleY, Point point);
```

```
// Prepends a
```

```
scale
```

```
public void Skew(Double skewX, Double skewY); // Appends a skew, in  
degrees
```

```
public void SkewPrepend(Double skewX, Double skewY); // Prepends a skew
```

```
public void Translate(Double offsetX, Double offsetY); // Appends a translation
```

```
public void TranslatePrepend(Double offsetX, Double offsetY); // Prepends a  
translation
```

```
public static operator * (Matrix matrix1, Matrix matrix2);
```

```
// Transformation services
```

```
public Point Transform(Point point);
```

```
public void Transform(Point[] points);
```

```
// Since this is a vector ignores the offset parts of the matrix
```

```
public Vector Transform(Vector vector);
```


956

```

public void TransformVectors(Vector[] vectors);

// Inversion
public double Determinant { get; }
public bool HasInverse { get; }
public Matrix Inverse { get; } // Throws ArgumentException
    InvalidOperationException if !HasInverse

// Individual members
public double M11 { get; set; }
public double M12 { get; set; }
public double M21 { get; set; }
public double M22 { get; set; }
public double OffsetX { get; set; }
public double OffsetY { get; set; }
};

```

For the markup syntax, the order is M11, M12, M21, M22, OffsetX, OffsetY:

```

matrix:
    ( coordinate comma-wsp ){5} coordinate | "Identity"

```

5

THREE-DIMENSIONAL VISUALS AND ELEMENTS

This section is directed towards three-dimensional effects, primarily for providing straightforward three-dimensional effects via the media integration layer for applications to use. Such effects are integrated with remoting, printing, desktop composition, and participate cleanly in the MIL architecture and element model, and the composability of media across the MIL. For example, business visualization has some requirements for three-dimensional presentation and interactivity. This may require hit testing resolution into thousands or tens of thousands of objects, and will result in interactive visualizations that might look something like the image shown in

10

15 FIG. 41.

227

952

The features described herein are generally directed towards runtime rendering and interaction, rather than functionality that is provided via modeling tools. The modeling functionality that is provided is more targeted for situations where animation or runtime construction will be needed to be exposed to the runtime. For instance, 3D text extrusion may be provided, even though it is really a modeling feature, since applications will want to databind text in, thus requiring this to be a runtime feature.

As described above, Visuals in the MFL system represent the 2D compositing tree of what ultimately gets rendered. Note that `UiElements` and `Controls` are `Visuals`. Additionally, `Visuals` have a collection of child `Visuals` that are rendered in front (in painter's algorithm terms) of the `Visual`'s own content. As also described above, there are a variety of concrete `Visual` subclasses, including `RetainedVisual`, `SurfaceVisual`, `HwndVisual`, and so forth. A `RetainedVisual` represents a set of 2D drawing instructions that the user either creates directly onto the `Visual`, or does so as the result of receiving an `OnRender()` callback from the layout system.

As will be understood, 3D fits naturally into the 2D visual world. To this end, a `Retained3DVisual` is provided, that, like its 2D counterpart, represents a list of drawing instructions that is imperatively filled in by opening a drawing context, and rendering into that drawing context. This will effectively result in building a scene graph, or a metafile, or an instruction list into the `Visual` itself.

A `Retained3DVisual` is essentially a set of three-dimensional (rendering instructions / scene graph / metafile) data including lights, a camera to define the 2D projection of that scene, a rectangular two-dimensional viewport in local coordinate space for mapping the projection to, and other ambient parameters, such as antialiasing switches, fog switches, and the like.

Note that an on-demand `OnRender()` capability is not provided for the `Retained3DVisual`. This functionality exists for 2D, to help manage the amount of data drawn in the 2D world. Like 2D, rendering happens via a `DrawingContext` where "immediate-mode feeling" calls get made. For instance, in 2D, the following may be in a drawing context:

```
DrawingContext ctx = ...;  
ctx.DrawRectangle(...);  
ctx.PushTransform(...);
```

280

5

```
ctx.DrawGeometry(...);
ctx.PushTransform(...);
ctx.DrawEllipse(...);
ctx.Pop();
ctx.Pop();
```

For consistency with 2D, a similar model is provided in 3D that looks like:

10

```
DrawingContext3 ctx = ...;
ctx.DrawMesh(mesh, material);
ctx.PushTransform(transform3);
ctx.DrawMesh(...);
ctx.PushTransform(...);
ctx.DrawMesh(...);
ctx.Pop();
ctx.Pop();
```

15

Note that this model of rendering works well for both a Retained Mode 3D visual (where the “instructions” are simply saved), and an Immediate Mode 3D visual (where the rendering happens directly, and a camera needs to be established up front). In fact, in the retained mode case, what happens internally is a 3D modeling hierarchy is getting built up and retained. Alternatively, in the immediate mode case, no such thing is happening, instructions are being issued directly, and a context stack (for transforms, for example) is being maintained.

20

The following is an example showing programming with the 3D Visual API.

25

This example creates a Retained3DVisual, obtains a drawing context to render into, renders primitives and lights into it, sets a camera, and adds the visual to the visual children of a control:

30

```
// Create a 3D visual
Retained3DVisual visual3 = new Retained3DVisual();
--
// Render into it
using (Drawing3DContext ctx = visual3.Models.RenderOpen())
```

```

{
    // Render meshes and lights into the geometry
    ctx.DrawMesh(mesh, material);
    ctx.PushTransform(transform3);
5   ctx.DrawMesh(...);
    ctx.PushTransform(secondTransform3);
    ctx.AddLight(...);
    ctx.DrawMesh(...);
    ctx.Pop();
10   ctx.Pop();
}

// Establish ambient properties on the visual
visual3.Camera = new ProjectionCamera(...);
15

// Add it to the compositing children of some control called myControl
((IVisual)myControl).AddChild(visual3);

```

20 The integration of 2D into 3D is a significant feature, and enables the very interesting scenarios of having 2D UI interact in a 3D world, be mapped onto 3D surfaces, and so forth. 2D integrates into 3D in separate ways. One way is as a texture onto a 3D primitive. Any Avalon 2D Brush (as well as more complex material specifications) can be used to texture 3D. A special case of this is the VisualBrush which can host an arbitrary Visual (and thus Controls, or entire Application UIs).

25 Mechanisms are provided to resolve hit testing and further hit test into that visual for applications that desire to do so and, in certain cases, the hit-testing resolution will be done such that those Visuals can be live. Note that since the Visual itself is not a bitmap, Visual-texturing can be implemented through more sophisticated techniques, such as taking the 2D geometry in the Visual and bringing it into 3D, thus having 3D

30 vector graphics rather than 2D bitmaps used as textures.

Another way that 2D integrates into 3D is as a rendering instruction on a 3D context. It is also desirable to be able to simply render a 2D visual, screen aligned,

960

and unscaled without any perspective effects, at a particular z depth, e.g., to stack a sphere, a 2D visual, and a cube while still allowing them all to share the same z-buffer, thus allowing, for instance, a sphere to pass through a piece of 2D UI. This will be exposed by a "DrawVisual" instruction on the Drawing3DContext.

5 As described herein, Animation should operate intuitively, and 2D animation technology is directly transferable to 3D. An exception is rationalizing the timelines.

The above shows an imperative rendering style of usage where drawing instructions are issued to the context. This is not a declarative usage, and thus this imperative approach is not appropriate for declarative markup.

10 Because with the above mechanism there is no 3D Model being explicitly constructed externally, (even though internally it is), a programmer cannot address a 3D model or enumerate its contents. As a result, there is no place with which to write "effects" that take a model as input and generate a new model.

15 To provide a declarative way of building up and using 3D "resources" as in 2D with Brushes, Pens, Geometry, Paths, and so forth, a number of types are provided that allow users to construct what goes into the 3D instruction stream, and the constructed object can be set into a Retained3DVisual instead of using the context.

For example, the above Drawing3DContext-based sample code could be rewritten as:

```
20   // Create a 3D visual
    Retained3DVisual visual3 = new Retained3DVisual();

    visual3.Models.Add(new MeshPrimitive3D(mesh, material));

25   Model3DCollection innerGroup1 = new Model3DCollection();
    innerGroup1.Transform = transform3;
    innerGroup1.Add(new MeshPrimitive3D(mesh, material));

    Model3DCollection innerGroup2 = new Model3DCollection();
30   innerGroup2.Transform = secondTransform3;
    innerGroup2.Add(new Light(...));
    innerGroup2.Add(new MeshPrimitive3D(...));
```

283

```

innerGroup1.Add(innerGroup2);

visual3.Models.Add(innerGroup1);
5
// Everything else is the same as before...

// Establish ambient properties on the visual
visual3.Camera = new ProjectionCamera(...);
10
// Add it to the compositing children of some control called myControl
((IVisual)myControl).AddChild(visual3);

```

Here, a model is being built, and then assigned into the Retained3DVisual.

15 PushTransform/Pop pairs are replaced by construction of a Model3DCollection which itself has a transform and Models beneath it. Providing both the modeling approach and the imperative context-based approach provides a solution for element-level declarative markup, visual enumeration, scene graph effects and modifiability of visual contents, and is done in an architecturally viable way.

20 The root of the modeling class tree is Model3D, which represents a three-dimensional model that can be attached to a Retained3DVisual. Ultimately, lights, meshes, .X file streams (so it can come from a file, a resource, memory, and so forth), groups of models, and 3D-positioned 2D visuals are all models. Thus, there is the following hierarchy:

- Model3D
 - Model3DCollection – is a collection of Model3D
 - Primitive3D
 - MeshPrimitive3D(mesh, material, hitTestID)
 - ImportedPrimitive3D(stream, hitTestID) (for .x files)
 - Light --
 - AmbientLight
 - SpecularLight

462

- DirectionalLight
- PointLight
 - SpotLight
- VisualModel3D – has a Visual and a Point3 and a hitTestID

The Model3D class itself supports a number of operation, including Get 3D bounding box., Get and set the Transform of the Model3D, Get and set other “node” level properties, like shading mode and Get and set the hitTestObject. Note that
 5 explicit 3D-initiated hit testing is not being exposed within a 3D scene. In other words, there is not an API to project an arbitrarily directed ray into a 3D scene to see what is hit. The hit testing functionality will be accessed by performing a hit test against the 2D projection in the Retained3DVisual.

10 Incorporating 3D poses a challenge when it comes to rationalizing the 2D coordinate system with the 3D coordinate system. In 2D, the coordinate system has the origin in the upper left, +x to the right, and +y down.

The 3D coordinate system may be a right-handed one, where +z is coming out of the screen, +x is to the right (which is the same as 2D), and +y is up (which is different from 2D). This is because a left handed coordinate system in a 3D graphics
 15 context tends to create large numbers of unnecessary programming errors. Note that modern hardware has the ability to download per-vertex shaders, thus making it possible to use whatever coordinate system convention is desired.

Another aspect of coordinate system rationalization is the u,v space that textures live in. There is no particular default convention for u,v space, but models
 20 with “+v going down” will allow a more direct mapping of 2D UI through texturing than models with “+v going up” (for those the texture will likely appear upside down). In the event where the model being textured does not have the properly oriented u,v grid, the Transform property on the Material’s Brushes can be used to counteract.

A number of experiments have been performed to look into the most viable
 25 way to get text to look attractive in 3D; (note that this doesn’t necessarily imply “3D Text” with width, height, and depth). To this end, an application may construct 2D Visuals holding the text (and other arbitrary 2D elements) to be used in 3D, ensure that that Visual gets oversampled (e.g., by 4x-8x), and then use this Visual as a Material

265

onto a plane or any arbitrary 3D object. If the resultant texturing is anisotropic, the result is attractive text.

Note that declarative types specified in this section (e.g., the vectors, points, lights, meshes, materials, primitives, transforms, and the like) are all readily describable through XAML-based markup using the standard XAML CLR class type description mechanisms. Types may have TypeConverters, but if not, they are specifiable in markup through the standard simple and complex property description mechanisms that XAML offers. Note that TypeConverter specifications are for interpreting pure string-based representations of types as they appear in XAML. Thus, for example, for a Vector3D that's used in a ScaleTransform3D, the XAML markup would be:

```
<ScaleTransform3D scaleVector="1,2,8" />
```

Note that the "ScaleTransform3D" and the "scaleVector" are parsed and understood by the generic XAML parser, but the "1,2,8" is fed to the Vector3D's TypeConverter and expected to generate a Vector3D (since that's the type of the "ScaleVector" property of ScaleTransform3D.) Further, Note that while not explicitly listed for each type, these types have the following methods (shown here for Vector3D, but applicable to others):

```
public static bool operator == (Vector3D vector1, Vector3D vector2)
public static bool Equals(Vector3D vector1, Vector3D vector2)
public static bool operator != (Vector3D vector1, Vector3D vector2)
public override bool Equals(object o)
public override int GetHashCode()
public override string ToString()
```

Also, any type that derives from Changeable (either directly or indirectly) will need to have a "public new MyType Copy()" method on it. These primitive types simply exist in support of the other types described in this section. Wherever possible, they mirror the primitive types used in 2D, and this similarity is a design goal for these types.

964

The Point3D object is a straightforward analog to the 2D Point type
System.Windows.Point:

```
5 public struct System.Windows.Media3D.Point3D
{
    public Point3D(); // initializes to 0,0,0
    public Point3D(double x, double y, double z);

    public double X { get; set; }
10 public double Y { get; set; }
    public double Z { get; set; }

    public void Offset(double dx, double dy, double dz);

15 public static Point3D operator +(Point3D point, Vector3D vector);
    public static Point3D operator -(Point3D point, Vector3D vector);
    public static Vector3D operator -(Point3D point1, Point3D point2);
    public static Point3D operator *(Point3D point, Matrix3D matrix);
    public static Point3D operator *(Point3D point, Transform3D transform);

20 public static explicit operator Vector3D(Point3D point);

    // Explicit promotion of a 3D point to a 4D point. W coord becomes 1.
    public static explicit operator Point4D(Point3D point);
25 }
```

```
30 coordinate:
    double-number-representation
    -
    comma-wsp:
    one comma with any amount of whitespace before or after
```

287

965

```
coordinate-triple:
    (coordinate comma-wsp){2} coordinate
```

```
5 point3D:
    coordinate-triple
```

Vector3D is a straightforward analog to the 2D Vector type
System.Windows.Vector:

```
10 public struct System.Windows.Media3D.Vector3D
    {
        public Vector3D(); // initializes to 0,0,0
        public Vector3D(double x, double y, double z);

15     public double X { get; set; }
        public double Y { get; set; }
        public double Z { get; set; }

        public double Length { get; }
20     public double LengthSquared { get; }

        public void Normalize(); // make the Vector3D unit length

        public static Vector3D operator -(Vector3D vector);
25     public static Vector3D operator +(Vector3D vector1, Vector3D vector2);
        public static Vector3D operator -(Vector3D vector1, Vector3D vector2);
        public static Point3D operator +(Vector3D vector, Point3D point);
        public static Point3D operator -(Vector3D vector, Point3D point);
        public static Vector3D operator *(Vector3D vector, double scalar);
30     public static Vector3D operator *(double scalar, Vector3D vector);
        public static Vector3D operator /(Vector3D vector, double scalar);
        public static Vector3D operator *(Vector3D vector, Matrix3D matrix);
```

200

966

```

public static Vector3D operator *(Vector3D vector, Transform3D transform);

// return the dot product: vector1.X*vector2.X + vector1.Y*vector2.Y
public static double DotProduct(Vector3D vector1, Vector3D vector2);

5 // return the normal vector between the two input vectors by
// computing the cross product.
public static Vector3D CrossProduct(Vector3D vector1, Vector3D vector2);

10 // return the angle required to rotate v1 into v2, in degrees
// This will return a value between (-180, 180] degrees
public static double AngleBetween(Vector3D vector1, Vector3D vector2);

public static explicit operator Point3D(Vector3D vector);}

15 // Explicit promotion of a 3D vector to a 4D point. W coord becomes 0.
public static explicit operator Point4D(Vector3D point);

```

```

20 point3D:
    coordinate-triple

```

Point4D adds in a fourth, w, component to a 3D point, and is used for transforming through non-affine Matrix3D's. There is no Vector4D, as the 'w' component of 1 translates to a Point3D, and a 'w' component of 0 translates to a Vector3D:

```

public struct System.Windows.Media3D.Point4D
{
    public Point4D(); // initializes to 0,0,0,0
    public Point4D(double x, double y, double z, double w);

30 public double X { get; set; }

```

287

962

```

public double Y { get; set; }
public double Z { get; set; }
public double W { get; set; }

5  public static Point4D operator -(Point4D point1, Point4D point2);
    public static Point4D operator +(Point4D point1, Point4D point2);
    public static Point4D operator *(double scalar, Point4D point);
    public static Point4D operator *(Point4D point, double scalar);
    public static Point4D operator *(Point4D point, Matrix3D matrix);
10  public static Point4D operator *(Point4D point, Transform3D transform);
    }

```

```

point4D:
    coordinate-quad

```

15

Quaternions are distinctly 3D entities that represent rotation in three dimensions. Their power comes in being able to interpolate (and thus animate) between quaternions to achieve a smooth, reliable interpolation. The particular interpolation mechanism is known as Spherical Linear Interpolation.

20

Quaternions can either be constructed from direct specification of their components (x,y,z,w), or as an axis/angle representation. The first representation may result in unnormalized quaternions, for which certain operations do not make sense (for instance, extracting an axis and an angle).

The components of a Quaternion cannot be set once the Quaternion is constructed, since there is potential ambiguity in doing so, e.g., setting the Angle on a non-normalized Quaternion has no meaning.

```

public struct System.Windows.Media3D.Quaternion
{
    public Quaternion(); // initializes to 0,0,0,0
    ...

    // Non-normalized quaternions are allowed
    public Quaternion(double x, double y, double z, double w);

```

30

22.6

968

```

// allow construction through axis and angle
public Quaternion(Vector3D axisOfRotation, double angleInDegrees);

5 // fundamental Quaternion components
public double X { get; }
public double Y { get; }
public double Z { get; }
public double W { get; }

10 // axis/angle access. Will raise an exception if the quaternion
// is not normalized.
public Vector3D Axis { get; }
public double Angle { get; } // in degrees, just like everything else

15 // Magnitude of 1? Only normalized quaternions can be used in
// RotateTransform3D's.
public bool IsNormalized { get; }

20 public Quaternion Conjugate(); // return conjugate of the quaternion
public Quaternion Inverse(); // return the inverse of the quaternion
public Quaternion Normalize(); // return a normalized quaternion

public static Quaternion operator +(Quaternion left, Quaternion right);
25 public static Quaternion operator -(Quaternion left, Quaternion right);
public static Quaternion operator *(Quaternion left, Quaternion right);

// smoothly interpolate between two quaternions
public static Quaternion Slerp(Quaternion left, Quaternion right, double t);
30 }

```

quaternion:

2.11

coordinate-quad 	// x,y,z,w representation
"(" coordinate-triple ")" coordinate	// axis,angle representation

5 **Matrix3D** is a 3D analog to **System.Windows.Matrix**. Like **Matrix**, most APIs don't take **Matrix3D**, but rather **Transform3D**, which supports animation in a deep way. Matrices for 3D computations are represented as a 4x4 matrix. The MIL uses row-vector syntax:

m11	m12	m13	m14
m21	m22	m23	m24
m31	m32	m33	m34
offsetX	offsetY	offsetZ	m44

10 When a matrix is multiplied with a point, it transforms that point from the new coordinate system to the previous coordinate system.

Transforms can be nested to any level. Whenever a new transform is applied it is the same as pre-multiplying it onto the current transform matrix:

```

public struct System.Windows.Media3D.Matrix3D
{
    // Construction and setting
    public Matrix(); // defaults to identity

    public Matrix(
        double m11, double m12, double m13, double m14,
        double m21, double m22, double m23, double m24,
        double m31, double m32, double m33, double m34,
        double offsetX, double offsetY, double offsetZ, double m44);

    // Identity
    public static Matrix3D Identity { get; }
    public void SetIdentity();
    public bool IsIdentity { get; }
}

```

// Math operations

public void Prepend(Matrix3D matrix); // "this" becomes: matrix * this
public void Append(Matrix3D matrix); // "this" becomes: this * matrix

**// Rotations – Quaternion versions. If you want axis/angle rotation,
 // build the quaternion out of axis/angle.**

public void Rotate(Quaternion quaternion);
public void RotatePrepend(Quaternion quaternion);
public void RotateAt(Quaternion quaternion, Point3D center);
public void RotateAtPrepend(Quaternion quaternion, Point3D center);

public void Scale(Vector3D scalingVector);
public void ScalePrepend(Vector3D scalingVector);
public void ScaleAt(Vector3D scalingVector, Point3D point);
public void ScaleAtPrepend(Vector3D scalingVector, Point3D point);

public void Skew(Vector3D skewVector); // Appends a skew, in degrees
public void SkewPrepend(Vector3D skewVector);
public void SkewAt(Vector3D skewVector, Point3D point);
public void SkewAtPrepend(Vector3D skewVector, Point3D point);

public void Translate(Vector3D offset); // Appends a translation
public void TranslatePrepend(Vector3D offset); // Prepends a translation

public static Matrix3D operator * (Matrix3D matrix1, Matrix3D matrix2);

**// Transformation services. Those that operate on Vector3D and Point3D
 // raise an exception if IsAffine == false.**

public Point3D Transform(Point3D point);
public void Transform(Point3D[] points);
public Point4D Transform(Point4D point);

971

```

public void Transform(Point4D[] points);

// Since this is a vector ignores the offset parts of the matrix
public Vector3D Transform(Vector3D vector);
public void Transform(Vector3D[] vectors);

// Characteristics of the matrix
public bool IsAffine { get; } // true if  $m_{1,2,3,4} = 0$ ,  $m_{44} = 1$ .
public double Determinant { get; }
public bool HasInverse { get; }
public Matrix3D Inverse { get; } // Throws InvalidOperationException if
    !HasInverse

// Individual members
public double M11 { get; set; }
public double M12 { get; set; }
public double M13 { get; set; }
public double M14 { get; set; }
public double M21 { get; set; }
public double M22 { get; set; }
public double M23 { get; set; }
public double M24 { get; set; }
public double M31 { get; set; }
public double M32 { get; set; }
public double M33 { get; set; }
public double M34 { get; set; }
public double OffsetX { get; set; }
public double OffsetY { get; set; }
public double OffsetZ { get; set; }
public double M44 { get; set; }
};

```

1

matrix3D:

(coordinate comma-wsp){15} coordinate | "Identity"

Transform3D, like the 2D Transform, is an abstract base class with concrete subclasses representing specific types of 3D transformation. Specific subclasses of Transform3D are also where animation comes in.

In one implementation, the overall hierarchy of Transform3D is set forth below:

Transform3D

— Transform3DCollection

— AffineTransform3D

—— TranslateTransform3D

—— ScaleTransform3D

—— RotateTransform3D

— MatrixTransform3D

Root Transform3D has static methods for constructing specific classes of Transform; (note that it does not expose a Matrix3D representation, as this Transform may be broader):

```
public abstract class System.Windows.Media.Media3D.Transform3D : Animatable
```

```
{
```

```
15     internal Transform3D0;
```

```
    public new Transform3D Copy();
```

```
    // static helpers for creating common transforms
```

```
    public static MatrixTransform3D CreateMatrixTransform(Matrix3D matrix);
```

```
20    public static TranslateTransform3D CreateTranslation(Vector3D translation);
```

```
    public static RotateTransform3D CreateRotation(Vector3D axis, double angle);
```

```
    public static RotateTransform3D CreateRotation(Vector3D axis, double angle,
```

```
        Point3D rotationCenter);
```

```
    public static RotateTransform3D CreateRotation(Quaternion quaternion);
```

```
25    public static RotateTransform3D CreateRotation(Quaternion quaternion,
```