NISTIR 88-4004

Product Data Exchange Specification
First Working Draft

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
(Formerly National Bureau of Standards)
National Engineering Laboratory
Center for Manufacturing Engineering
Factory Automation Systems Division
Gaithersburg, MD 20899

December 1988

Chair, IGES/PDES Organization
Bradford Smith

Coordinator, IGES/PDES Organization
Gaylen Rinaudot

NISTIR 88-4004

Product Data Exchange Specification
First Working Draft

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
(Formerly National Bureau of Standards)
National Engineering Laboratory
Center for Manufacturing Engineering
Factory Automation Systems Division
Gaithersburg, MD 20899

December 1988

National Bureau of Standards became the National Institute of Standards and Technology on August 23, 1988, when the Omnibus Trade and Competitiveness Act was signed. NIST retains all NBS functions. Its new programs will encourage improved use of technology by U.S. industry.

Chair, IGES/PDES Organization
Bradford Smith

Coordinator, IGES/PDES Organization
Gaylen Rinaudot

U.S. DEPARTMENT OF COMMERCE
C. William Verity, Secretary

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Ernest Ambler, Director

Preface

Publications of the National Institute of Standards and Technology undergo rigorous technical, policy, and editorial reviews to assure their quality.[1] This interim report does not meet the normal requirements of the NIST technical and editorial reviews. Its content has been developed by hundreds of technical experts working in the United States and abroad to develop a standard for the exchange of product data in digital form. Collectively, the contributions of these experts are intended to form the working draft of the standard. The content of this interim report has been accepted by the IGES/PDES Organization as being sufficiently complete and sufficiently correct to warrant broad technical review by the national and international standards making organizations. Publication of this interim report freezes the content and makes it available in a stable, referenceable form suitable for comprehensive review.

A final report will be published when the review process is complete.

---

[1]The technical review includes critical evaluation of the technical content and methodology. The policy review includes examination of consistancy with NIST statutory authority and operating policy, and appropriateness of selected medium of publication, and other matters as appropriate. The editorial review includes a check on nomenclature, formatting, titling, references, indexing, citations, footnotes, and acceptable standards of writing quality.

TABLE OF CONTENTS

NISTIR 88-4004

Officers of the IGES/PDES Organization*

Chairman	Bradford Smith
Coordinator	Gaylen R. Rinaudot
IGES Project Manager	Constance Panzica, Dennette Harrod, Jr.
PDES Project Manager	Kalman Brauner, Thurber Moffett, Anthony Day
ANSI Project Manager	Earl Weaver
ISO Project Manager	Jerry Weiss
Testing Project Manager	Marc Pearson
Change Control	Curt Parks, James Johnson
Acceptance Testing	James Flemming
Application Validation	Mark Palmer
Architecture, Engineering, and Construction	Kent Reed, Pat Rourke, Barbara Warthen
Curves and Surfaces	Edward Clapp
Drafting	Bill Turcotte, Robert Parks
Edit	Philip Kennicott
Electrical Applications	Larry O'Connell
Finite Element Modeling	Harvey Gray
Form Features	Mark Dunn
Implementations	Bill Loye
Integration	Yuhwei Yang
Logical Layer	Doug Schenck
Materials Properties	Richard Brooks
Manufacturing	William Burkett, Greg Paul
Mechanical Product Definition	Tom Voegeli, Peter Everitt
Methodology Testing	Dave Remington
Physical File Format	Jeff Altemueller
Product Logistics	Rick Bsharah
Recommended Practices	Dennette Harrod, Jr., Linda Martino
Software Support	Raymond Barker
Solids Geometry	Noel Christensen
Technical Publications	Jeff Altemueller, Kelly Chi, Marc Durnin
Test Case Development	Julia Terry, Larry Mankins
Tolerance	William Burkett
User Information	Tom Wright
Verification Testing	Stan Briggs, Dave Remington

*Current and past officers who have served during the preparation of this document.

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Aebischer	Joseph	Martin Marietta Data Systems
Agrawal	Ajay	ALCOA
Aish	Robert	GMW Computers Limited
Alexander	Danny	Optigraphics Corp.
Alexander	Edward	Technical Computing Consultants
Altemueller	Jeff	McDonnell Douglas Corporation
Altmann	Charles	IBM Corp.
Altmann	Christian	IPK Berlin
Anderl	Reiner	University of Karlsruthe
Andersen	Kevin	Duke Power Company
Anderson	Bill D.	General Dynamics Corp.
Anderson	Dennis E.	Puget Sound Naval Shipyard
Anderson	Gretchen	HQ AFSC/PLXC
Anderson	Robert E.	NAVAIR Engineering Support Office
Ang	Ignatius G.	Proctor and Gamble Company
Anthony	Alfred	Arizona State Univ.
Armour	Max C.	CAM-I
Arndt	Werner	System Consult
Arnold	Larry	Hughes Aircraft
Arnold Jr.	Eugene M.	McDonnell Douglas
Arnsdorf	David R.	Industrial Technology Institute
Ashley	Colin	PAFEC Ltd
Atkins	Tom	Boeing Commercial Airplane Co.-CMO
Auger	Kevin	Evans & Sutherland
Avery	Carole	Boeing Computer Services
Ayers	Randy E.	NAVSEA
Bacela	Debra J.	Lockheed/EMSCO
Bahlo	Lenna	DIGITAL
Baidoo	Kofi	Boeing
Bailey	Bruce C.	Automation Technology Products
Baker	Dewey	Martin Marietta Data Systems
Baker	George	International TechneGroup Inc.
Baker	John L.	Aluminum Co of America
Baker	Len	Wright Patterson AFB
Bakkush	Janet	McDonnell Aircraft Company
Banocy	Janice M.	McDonnell Douglas MIS Co
Bares	Peter	IVF Co.
Barker	Raymond E.	Caterpillar, Inc.
Barrett	Myles	Computervision
Bastian	Matthew	ICAM Technologies Corporation
Battersby	Lois	US Army, PM-TPS
Beazley	William G.	W. G. Beazley & Assoc.
Beck	Douglas	Martin Marietta
Belcher	Ronald	McDonnell Douglas Automation Co.
Belnap	Boyd	Hughes Aircraft
Benacka	Robert	Magna International
Benjamin	Peter	CADAM Inc.
Bentrup	Bill	Gerber Systems Tech. Inc.
Beradino	Al	DOE/Sandia National Labs
Berenyi	Tibor A.	Deere and Company
Bernstein	Joe	Boeing Computer Services
Bettwy	Dave	National Bureau of Standards

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Betz	Gerry	Robotic Vision Systems Inc.
Bhagat	Anuradha	Electronic Data System
Billingsley	Dan	Naval Sea Systems Command
Bird	Chris	CALMA
Blaha	James R.	IIT Research Institute
Blaise	Max	Eastman Kodak Co.
Bloom	Howard	NIST
Bodnyk	Bruce W.	AMP Inc.
Boissiere	Katherine	DOE Sandia National Labs
Boldt	Thomas A.	Optigraphics Corp.
Boudreau	David	D. Appleton and Company
Boyle	Carol S.	Valisys Corp.
Bracken	Connie	EDS
Bradford	James R.	Allied-Signal Aerospace Co.
Brainin	Jack	Navy David Taylor Research Center
Brandt	Ray	BDM Corporation
Brauner	Kalman	Boeing Commercial Airplane Co.
Braunstein	Jeff	Tandem Computer
Bray	E. Dean	Allied-Signal Aerospace
Brazil	Terry	Boeing Commercial Airplane, CO.
Brei	M. L.	Institute for Defense Analysis
Bremer Jr.	Robert C.	Society of Automotive Engineers
Brengs	Raymond A.	DTRC
Brian	Jen	AUTOMATIX
Bridges	Andy	Northrop Aircraft
Bridges	Thomas F.	John J. Mullen Assoc.
Briggs	David	Boeing Commercial Airplane Co.
Briggs	Stan	PSD
Bronder	Clare	Adra Systems, Inc.
Brooks	Jeff	General CAD/CAM, Inc.
Brooks	Richard	McDonnell Douglas Corp.
Brown	David	Electronic Data Systems
Brown	David M.	Douglas Aircraft
Brown	Don	Naval Ordanance Station
Bryant	Walter A.	Boeing Computer Services
Bsharah	Frederick	Rockwell International NAAO
Burke	Andrew	Tektronix Inc.
Burkett	William C.	Lockheed Aeronautical Systems Co.
Burns III	Bernard J.	Naval Surface Weapons Center
Cain	William D.	DOE/Martin Marietta
Calkins	Bruce	Naval Sea Systems Command
Call	Bradley K.	PDA Engineering
Camarillo	Rudy	General Dynamics Corp.
Carberry	James	NAVFAC
Carpenter	S. Dean	DOE/Mason & Hanger
Carpenter	Vickie	Electronic Data Systems
Carr	William D.	Digital Equipment Corp.
Carringer	Robert	Int'l Techne Group Inc.
Carson	Philip T.	Aluminum Co. of America
Carter	Dr. Aaron L.	Hughes Aircraft Co.
Casey	Eva W.	Schlumberger
Chaffee	Steve	Auto-trol Technology
Chamberlain	Mark	Hughes Aircraft

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Davidson	Keith	International TechneGroup Inc,.
Davis	Neal	US Army Corps of Engineers
Davis	Paul H.	Pratt & Whitney
Davis-Oates	Carolyn	Battelle
Dawson	Laurel C.	IBM
Day	Anthony James	Sikorsky Aircraft UTC
Dearing	John	Fluor Engineers, Inc.
DeCarlo	Elise	Robotic Vision Systems Inc.
Deeds	Lisa	Navy David Taylor Research Center
DeJean	Jean Pierre	General Electric
Delaunay	J-Y	CN/INF
DeLiso	Robert	Obrien Kreitzbery
Delk	James	US Army Materiel Command
Dellinger	David Leigh	Boeing Military Airplane Company
DeMers	Diane	Manufacturing and Consulting Services
Dent	Deborah F.	USAE Waterways Exp. Station
DePauw	Spencer	Caterpillar, Inc.
Derfmueller	Mark	GE - Aircraft Engineers
Dernbach	Robert	Calma Company
Dickerson	Donna	Pratt and Whitney
DiFronzo	Richard N.	Metagraphics Inc.
DiGeronimo	Rose	Department of the Navy
Disa	Ralph	Pratt & Whitney
Dolson	Jim	Harvard Industries
Donaldson	Brian	Harris Computer Systems
Dorfmueller	Mark	General Electric Co.
Douglas	Gary	US Army Corps of Engineers
Downer	Ron	Hughes - MSG
Dragoo	Alan E.	McDonnell Douglas - ISG
Drybus III	George Newell	Newport News Shipbuilding
Duivenvoorden	Jacques	LOGOS Industrial Automation
Duncan	William	US Army
Dunn	Mark	United Technologies
Durnin	Marc W.	Lockheed Aeronautical Systems Co.
Dvorak	Andrew	Bath Iron Works
Dzung	C.	McDonnell Douglas Corp.
Earl	Colin R.	Automation Technology Products
Easley	Preston W.	Northrop Corporation
Eason	Lee W.	Department of Transportation
Eccles	T. E.	Tenneco, Inc.
Eliasen	Ned	Honeywell
Ellermeier	Richard	Mentor Graphics
Emmerson	Diane	Pratt & Whitney Aircraft
Endo	Richard	Versacad Corporation
Engeli	Dr. Max	FIDES TREUHANDGESELLSCHAFT ELECTRICAL
Erb	Sue	Janus Systems, Inc.
Erman	Ken	Cadkey, Inc.
Estervog	Thomas C.	Boeing Computer Services
Evans	E. G.	Mason & Hanger-Silas Mason Co., Inc.
Evans	Mike	LTV Aircraft Products Group
Even-Ari	Hillel	253 ben-Gurion Road
Everitt	Peter R.	Grumman Data Systems
Fahey	Mary Lou	NIST

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Fallon	Kristine	Computer Technology Management Inc.
Farrell	Jill	LLNL
Faulkner	John C.	SDRC
Fears	Ayron L.	IBM Corp.
Felt	Jim	Boeing Computer Services
Fergus	Peter	I.E.E
Fincher	Judith A.	Honeywell Federal Systems, Inc.
Fisher	Jennifer	DTNSRDC
Fitler	Bill	Sony
Fix	Randy	Fluor Daniel
Fkiaras	Nick	Tektronix Inc.
Flaherty	John	Boeing Computer Services
Flanagan	John	CASE Consulting
Fleming	Jim	Cummins Engine Co. Inc.
Floyd	Bill	General Dynamics
Fong	Henry H.	MARC Analysis Research Corp
Fosselman	Peter	Douglas Aircraft Corp.
Fox	Mike	GKN Technology Ltd.
Fox-McIntyre	Marlene	Naval Sea Systems Command
Francis	Ray M. (Mike)	NWC
Frayseth	Leland H.	Bechtel Petroleum Inc.
Frazier	Charlie	NASSCO
Fremgen	David	GE- Aircraft Engines
Freund	Kevin	General Dynamics - DSD Fort Worth
Friedman	Jeff	UNISYS CAD/CAM Inc.
Frimer	Morris	Boeing Electronics Company
Fuhr	Richard	Electronic Data Systems
Furlani	Cita	NIST
Fusco	Donald E.	Boeing Computer Services
Gale	Roger	DACOM
Galichon	Chris	
Gallo	Steve	United Technology Research Ctr
Gamble	Robert	Electronic Data Systems
Ganus	Floyd	LTV
Garcia	Chris	Valisys Corporation
Garner	Bruce L.	DOE/LLNL
Garner	William	Martin Marietta
Gauntlett	J. Clifford	Autodesk
Gauthier	Ronald	Prime Computer, Inc.
Geisinger	Walter F.	Construction Specifications Institute
Gemma	Jaqueline L.	General Dynamics Corporation
Gengenbach	Ulrich	Kernforschungszentrum Karlsruhe
Gerard	Douglas W.	Naval Weapons Center
Gerardi	Michael	Bath Iron Works
Gerrein	William	General Electric Co.
Gersht	Elena	Prime Computer Inc.
Gerus	Michael	AIAG
Gibbons	Albert J.	Westinghouse Electric Corp.
Gielingh	W. F.	IBBC - IND
Giguere	Marshall	Computervision Inc.
Gilbert	Chip	Martin Marietta
Gilbert	Mitchell	Grumman Aerospace Corp.
Gillard	Alfred B.	Chrysler Corp.

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Gillett	Robert	UNISYS
Gilman	Charles R.	USAF
Gischner	Burton	General Dynamics - Electric Boat
Gjovaag	Inghard	Tektronix
Glatz	Rainer	VDMA Dept. of Informatic
Gleason	Elwin B.	Int'l Fuel Cells
Glenwright	Tristan	McDonnell Douglas Astro.
Godfrey	Donald R.	Westinghouse
Goff	Harold R.	Giffels Associates
Goldsmith	George	McDonnell Douglas Astronautics
Goldstein	Barbara	McDonnell Douglas
Golish	Mike	CECER-FS
Gonia	Patrick	Honeywell CSDD
Goosen	Ted	General Dynamics
Gordon	Scott A.	NASA Goddard Space Flight Center
Goult	R.J.	Cranfield Institute of Technology
Grant	Richard	Pixelworks, Inc.
Gray	William H.	DOE/Martin Marietta
Green	Ed	UNISYS CAD/CAM Inc.
Green	Ronald	Boeing Military Airplane Co
Grubb	Mike	Nval Weapon Support Center
Gruttke	William B.	McDonnell Douglas (MDAIS)
Guilinger	Willis	Westinghouse Electric Corporation
Gunther	Eric J.	U S Air Force
Gurga	Eugene F.	Computervision Corp
Gygi	Michael	Douglas Aircraft
Hadfield	Brad	Harley-Davidson Inc.
Hagerty	Jeffrey	LLNL
Hagopian	Valerie	MCS
Hailstone	Simon	Cranfield Inst. of Tech.
Haines	Mark	ITI
Halford	Joseph D.	E.I. DuPont De Nemours & Co
Hamilton III	C. Hayden	PDA Engineering
Hammon	William	Tandem Computer
Hang	Bruce	Sperry Defense Products Group
Haridim	Yosef	Northrop Corp.
Harris	Alex	UNISYS
Harrison	Don E.	LTV Aircraft-Products Group
Harrison	Jessie A.	Rohr Industries, Inc.
Harrison	Randy J.	DOE/ Sandia National Labs
Harrod Jr	Dennette A.	Computervision Corporation
Harrow	Pat	Harrow Associates Ltd.
Hart	John	Boeing Advanced Systems Co.
Hartman	Brett	Boeing Computer Services
Harvey	Marie T.	Naval Electronic Systems Center
Hashimoto	Kenichi	Nippon Computer Graphics Assoc.
Hass	Barry	Raytheon Company
Haurie	Ken	FAI
Heatley	Lynn	Boeing Commercial Airplanes
Hebert	Charles	Boeing - GA
Heide	Scott	ICAD, Inc.
Heier	Trond	CADCOM A/S
Hemmelgarn	Don	International TechneGroup Inc

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Henghold	William M.	Universal Technology Corp.
Henninger	W. C.	ECRA
Herrin	Mike	Northrop
Hill	Patrick	Touche Ross
Hinson	Don	Boeing Computer Services
Ho	Dr. C. Y.	University of Missouri-Rolla
Hocking	Maria	Vitro Corp.
Hoffman	Mark M.	US Air Force
Hoffman	Peter	Northrop Corp
Hollingshead	Mark A.	McDonnell Douglas M&E
Hollingshead	Paul	McDonnell Aircraft
Hooper	R. John	Eastman Kodak Company
Hoover	Lloyd	Honeywell
Hopp	Theodore H.	NBS
Hori	Kazuo	McDonnell Douglas Astronautics
Horton	Mitchel D.	SDRC
Howard	H. Craig	Stanford University
Howell	Fran	Interleaf
Hu	Audrey	RCFC
Huang	Bill	EDS
Huaovc	Maria	FMC
Hughes	Rendell	Lockheed Aeronautical Systems Co.
Hullford	Tom	IBM
Humpal	Julie	Hewlett Packard
Humphrey	Darrell F.	Martin Marietta Data Systems
Hunten	Keith	General Dynamics Corporation
Hurd	Jeffrey	
Hussong	William A.	Honeywell
Iacovelli	Diane	Northrop ASD
Iannola	Joseph	EATON Corporation
Igoshi	Masanori	Japan Soc. for Promo. of Machine Indust.
Ippolito	Greg	General Dynamics
Isenberg	Jerrold	Northrop Corp - Aircraft Division
Isenberg	Madeleine R.	Northrop Corporation
Ishikawa	Yoshiaki	Ishikawajima-Harima (IHI)
Isler	Richard E.	DOE/Sandia National Labs
Ivey	Robert	Westinghouse Electric Corp.
Jacobson	Carl	Computervision/Prime
Jaeger	Dwight L.	DOE/Los Alamos
Jain	Sunil	InterCAD Corporation
Jakopac	David	Vista Technologies, Inc.
Jansen	David	Rosetta Technology
Jaynes	Steve	Finklestein Associates Inc.
JeJean	Jean Pierre	General Electric
Jenkins	George	Vitro Corp.
Jennings	Richard	CADKEY, Inc.
Jensen	David	EDS-BOC Hqts.
Jewison	Michael	Magna Int'l Suite 202
Jeziorski	George G	McDonnell Douglas
Joe	Julie	CALMA
Johnson	Doug	LTV Aerospace
Johnson	Jim L.	Genral Dynamics
Johnson	Robert B.	FAI

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Johnson	Robert H.	CIMDATA Inc.
Johnson	Stanley	NASA Johnson Space Center
Jones	Doug	Accugraph Corp.
Judd	Jon L.	Int'l TechneGroup Inc.
Jurrens	Kevin	Allied-Signal Aerospace
Kaczmarek	Edward	H.A. Simons LTD
Kaletta	Gary	McDonnell Douglas
Kallel	Maher	Massachusetts Institute of Technology
Kaminski	Diane	Northrop Corp.
Kamvar	Esfandiar	A T & T Bell Laboratories
Kapell	Chars	Eastman Kodak
Karlsten	Keith	MacNeal-Schwendler Corp.
Kassel	Ben	DTNSRDC
Kassel	Denis	ALCATEL
Kato	Toru	Toyota Motors
Keane	Peter J.	Western Forge Corp
Keith	Gregory	Automation Technology Products
Kellett	Leo C.	Eastman Kodak
Kelley	Dewayne	Menasco
Kelley	Michael	Auto-Trol Technology
Kelly	J.C.	DOE Sandia National Labs
Kemmerer	Sharon J.	National Bureau of Standards
Kengott	Debbie	AutoTrol Technology Corp.
Kennedy	Mary Kathryn	McDonnell Douglas Astronautics Co.
Kennicott	Philip	General Electric Co.
Kento	Kunihiko	Komatsu Ltd.
Kieffer	Paul	Textron Lycoming
Kiggans	Robert	SCRA
Kimura	Fumihiko	University of Tokyo
Kimura	Keith M.	Rockwell International - NAA
King	Edward	McDonnell Douglas
King	Elbert	General Motors-Truck & Bus Group
Kirksharian	Aron	Naval Supply Systems Command
Kishinami	Takeshi	Hokkaido Univ. - JAPAN
Klages	Ed	DOE/Martin Marietta
Klein	Stanley	S. Klein Newsletter
Klement	Kornel	Tech. Univ. Darmstadt
Klick	Anthony	Deleitte Haskins & Sells
Klimow	Dr. V.	Moscow Power Engineering Institute
Klingler	Paul	Eastman Kodak Company
Klyachman	Paul	Segnetron
Kmetz	Mike	University of Wyoming
Knapp	Lewis	Sun Microsystems
Knight	Frank	The Aerospace Corp.
Kniley	Michael	Applicon
Kockler	Ronald	Northrop Aircraft
Koetter	Frank	Eastman Kodak Company
Kohler	Rick	Dana Corp.
Kojima	Toshio	Mechanical Engineering Laboratory
Kontry	Karen L.	Electronic Data Systems
Korah	John K.	Black & Decker Corporation
Korchmarev	Nelli	Xerox Special Information Systems
Koretsky	Robert	University of Portland

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Korff	Ralf	BMW AG
Koski	Donald C.	Pratt & Whitney
Kovacich	Gregory	Holguin Associates
Krishnaswami	Ravi	Electronic Data Systems
Kriz	Kenneth J.	GTE
Krocinski	Vincent	IBM
Krol	Paul G.	Daisy Systems
Kshirsagar	Sudhir	Proctor & Gamble
Kuan	L. P.	Computer Systems Tech.
LaColla	M. K.	Northrop Advanced Systems
Ladd	Harry	DuPont
Langenbach	Cliff	GE Aircraft Engines
Langhorst	Fred	Mentor Graphics
LaPay	Deborah	Westinghouse Electric Corp.
LaPenta	John	Scan Graphics Inc.
Laughlin	Greg	Rockwell International
Lauritzen	Louis	LTV
Lawler	Bruce	ICAD
Lawrence	Ralph W.	Defense Logistics Agency
Lazear	Mike	Versacad Corporation
Leach	Lanse M.	U.S Military Academy
Leal	David	CEGB - Computing Center
Lee	John	QC Data Collectors, Inc.
Lee	Kaiman	Naval Facilities Engineering Command
Lee	Tony	Aries Technology Inc.
Leifeste	Barney	CASE Consulting, Inc.
Leifeste	Kim	Department of Energy
Lepisto	Bruce	Department of Defense
Leung	Richard	A T & T
Leung	Wing	DTNSRDC
Lew	Norman	US Navy - Western Division
Liao	Swanwa	IBM
Lichten	Larry	California State University
Lichten	Olga	IBM Corp.
Lietz	Carol A.	McDonnell Douglas M&E
Liewald	Mike	Hughes Aircraft
Lin	Vincent	Automation Technology Products
Linsner	Jim	Boeing Computer Services
Little	Maureen	Naval Civil Engineering Laboratory
Livingstone	Philip	Battelle
Lock	Patricia	Northrop Coporation
Lodewijks	Bart	Nederlandse Philips Bedryven B.V.
Long	Julie	Electronic Data Systems
Long	S.J.	Rockwell
Losinski	Mark	Control Data Corporation
Lovdahl	Richard	Lorien Systems
Lownsbery	Bruce	LLNL
Loye	William	Cadnetix
Lunsford	Wayne	DOE/Los Alamos National Labs
Lynch	Philip	Schlumberger CAD/CAM
Mac Gowan	Donna	Interleaf, Inc.
MacDonald	Pete	Hasbro Inc.
Mack	Jan Fr.	CADCOM

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Mackelprang	Scott	Rocketdyne
MacLatchie	Robert	PlanPrint Company
Madison	R. Lee	Lockheed - Georgia
Madsen	Gerald	Martin Marietta
Magnus	Stephen	Management Roundtable, CAD/CAM Alert
Magoon	Gary	CADKEY Inc.
Magyari	Don	Expressway
Magyor	Joseph	Computervision Corp.
Major	Fritz	Fraunhofer Institut
Makoski	Tom	International TechneGroup Inc.
Maldague	Pierre	Applicon
Mankins	Larry	Boeing Military Airplane Company
Marechal	Guy	Philips & MBLE Associated
Marks	William D.	Hughes Aircraft Co
Marshall	Jim	IBM
Marshall	Mike	CAMAX Systems
Martin	Bryan	Lockheed Aeronautical Systems Company
Martin	Douglas J.	NASSCO
Martin	Duane	General Electric
Martin	Robert R.	Ontologic Inc.
Martino	Linda	IBM Corp.
Marz	Steven D.	Intergraph
Mason	Howard G.	British Aerospace
Mastrangelo	Steven G.	Micro Control Systems, Inc.
Mastrangelo	Steven G.	CADKEY
Matthews	Jon H.	JJH Inc. (NIDDESC)
Mayer	Ralph J.	Adra Systems, Inc.
Mays	James	Naval Supply Systems Command
Mazey	Richard	Battelle Memorial Laboratory
McBurney	Dennis	Boeing Military Airplane Co
McClure	Terry	Martin Marietta
McDevitt	Tom	Northrop ASD
McFadden	Patrick	Boeing Computer Services
McIntyre	C. Kevin	Artecon
McMillen	Stanley	Gerber Systems Technology
McMullen	Jim	Fluor Engineers Inc.
Meagher	Robert J.	Eastman Kodak
Meals	Monte	CALMA Co.
Meaney	George	FMC
Meersman	Robert	KHT-TILBURG University - INFOLAB
Meier	Christian	Zellweger Uster AG
Mellenthin	Carl A.	McDonnell Aircraft Co.
Melson	Thomas G.	McDonnell Aircraft Company
Menas	Telis K.	Aris Aegis Inc.
Merkley	Karl	The Aerospace Corporation
Merrill	Chris	Corp of Engineers
Messcher	Walter	US Department of Transportation
Messinger	Ray	Lorien Systems
Meyer	H. W. Guy	Wisconsin Dept of Transportation
Meyer	Lyle	Eastman Kodak
Meyer	Thure	
Meys	Frans	Philips B. V.
Miao	Walter	Philips Electronic Inst., Inc.

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Middaugh	Robert	Naval Air Engineering Center
Miessner	Wayne	Stone and Webster Engineering
Milander	Roberta	Harris
Miller	Darin	General Dynamics
Miller	Jay A.	CADNETIX Inc
Miller	Rex	Xerox Corporation
Miller	William	CADKEY Inc.
Milloy	Joan	Control Data Corporation
Mindel	Carolyn F.	S D R C
Mitchell	Mary	NIST
Miyamatsu	Masato	Komatsu, Ltd.
Moffett	Thurber J.	Northrop Corporation
Mollon	Roger	UNISYS CAD/CAM Inc.
Monson	Paul	General Motors Corporation
Montoya	Lloyd	Los Alamos Nat'l Labs
Moon	Randall K.	FSA, Inc.
Moore	Kevin	ATP
Moore	Richard	Eaton Corporation
Mordoch	Avraham	Avraham Y. Goldratt Ins.
Morea	Greg	General Dynamics Corporation
Morgan	Donald E.	General Electric Co.
Morrill	Charles B.	IBM Corp.
Morris	Michael	Nova Scotia CAD/CAM Centre
Morrison	Errol	UNISYS
Morse	Darla	Martin Marietta
Moshirvaziri	Sorour	IBM Corp.
Motz	Phil	Cincinnati Milacron
Moyer	Robert S.	Naval Electronic Systems Eng Cen
Mudd	Joseph A.	Bechtel Petroleum Inc.
Muller	Herbert	Thomatronik
Murphy	Jim	Tektronics, Inc.
Murphy	John	NESEA
Muth	Kenneth J.	Monroe Auto Equipment
Nagel	Roger	Lehigh University
Nagendra	I. V.	ICAM Technology
Nairn	Bill	CADDETC
Nakamura	Yasushi	NISSAN Motor Co., LTD
Navsky	Bruce	IBM
Nell	James G.	Westinghouse Electric Corp.
Nelson	Bruce R.	Control Data Corporation
Nelson	Paul A.	Hughes Aircraft Co.
Newton	Christopher	National Computing Center
Nicholas	Chris	Jupiter Systems
Nisenzon	Semyon	Apollo Computer
Nolan	Mike	Tandem Computers Inc.
Nordstrom	Karen	South Carolina Research Authority
Norling	David	Boeing Computer Services
Norton	Fred J.	DOE/LLNL
Noss	David S.	Naval Electronic Systems Center
Nowacki	Prof Horst	Technical University of Berlin
Nugteren	Brigitte	Ford Aerospace
Nurkse	Peter	FMC Corp.
O'Connell	Larry	DOE Sandia National Labs

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
O'Dowd	William L.	Black, O'Dowd and Associates, Inc.
Oakes	W. Rob	DOE/Los Alamos National Labs
Oates	Carolyn D.	Battelle
Obermeier	Wesley W.	LTV Aerospace and Defense Company
Ohlsson	Alf	Volvo Car Corporation
Ohtaka	Akihiko	Nihon UNISYS Ltd.
Olgilvie	Linda M.	Intergraph Corp.
Oliva	Lawrence	Prime Computer, Inc.
Olmsted	Peter	Olmsted Engineering
Olzer	J. H.	Lockheed Aero. Sys. Co.
Opene	Emeka	Douglas Aircraft Co.
Oravitz	Jacqueline L.	James M. Montgomery, Inc.
Ordiway	Dennis	Northrop Corporation
Otis	Allen	Servio Logic
Overbeek	Mike	Auto-Trol Technology Corp
Owen	Jon	CADDETC, University of Leeds
Paine	Louis C.	EDS
Palatine	James R.	AVCO Lycoming Textron
Pallas	Jack	A T & T
Palmer	Bill	Strageties & Systems Inc.
Palmer	Mark	NIST
Pandit	Lalit	Hewlett Packard
Panzica	Connie	General Motors BOC Hqts
Parker	Julie	NBS
Parker	Lawrence O.	Hughes Aircraft Co.
Parks	Curtis H.	NIST
Parks	Jeff L.	Boeing Computer Services
Parks	Robert E.	DOE Sandia National Labs
Paschelke	Robert	Point Control Co.
Pate	James S.	Boeing Computer Services
Paul	Greg A.	General Dynamics
Payne	John H.	Honeywell, Inc.
Pearson	Mark	CADDETC University of Leeds
Peck	Lawrence J.	Cimlinc Inc.
Peltzman	Alan	Peltzman & Associates
Penney	Jason	Servio Logic
Perlotto	Kim	Pratt & Whitney
Perry	Kathy	Dept. of Army
Peskin	Adele	UNISYS CAD/CAM
Peters	Tom	Computervision Corporation
Pewitt	Brent	Syerdup Technology, Inc.
Phan	Hy	Manufacturing & Consulting Serv Inc
Phelps	Freda	Sun Micro Systems
Philips	Charles	Department of Transportation
Philips	Gerald G.	Johnson Controls Inc.
Philips	Linda	Pratt & Whitney
Phillipson	Brian	British Aerospace
Piatok	Milton	Boeing Electronics Co.
Pilkenton	Steve	General Dynamics
Pilla	Paul F.	McDonnell Douglas Company
Pistey	Robert	IBM
Plotkin	Martin	ICAD, Inc.
Plunkett	Patrick T.	Naval Air System Command

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Poland	David A.	Northrop Aircraft
Polini	Michael A.	The Jonathan Corporation
Pollak	Gary W.	Society of Automotive Engineers
Poretti	Robert	Honeywell
Powers	Michael	General Electric Corp.
Prassinos	Pete	LLNL
Preschoux	Jean-Luc	Avions Marcel Dassault-Breguet Aviation
Price	Richard A.	Boeing Military Airplane Company
Primm	Richard	McDonnell Douglas
Prohaska	Tim	OPTI-COPY
Purdon	James C.	Schlumberger CAD/CAM
Purves	Lloyd	NASA Goddard Space Flight Center
Raikar	Ajay	ALCOA
Rajan	Roy K.	General Dynamics-FW Div.
Randall	C. David	Electronic Data Systems
Ranke	Guus	Nederlandse Philips Bedryven
Ratiner	Natalie	Cognition Inc.
Ray	Dr. Brian	Robert Gordon's Inst. of Tech.
Ray	Steve	NBS
Reddy	Ken	Finkelstein Assoc., Inc.
Reed	Kent	National Bureau of Standards
Reid	E. A.	Caterpillar Inc.
Reid	Richard	Hawker de Havilland Ltd.
Remington	David O.	Naval Surface Warefare Cntr.
Renkel	Harold E.	NASA, Lewis Research Center
Ressler	Sanford	NIST
Reyer	Jim	Cadnetix
Rhodes	Keith	LLNL
Richter	Karen	Inst. for Defense Analyses
Riddel	Charles	UNISYS CAD/CAM Inc.
Riffert	William	Prime Computer
Rinaudot	Gaylen R.	NIST
Rios	Phil	CALMA Co.
Robertson	Jon	Graphic Information ExchangJUL87vice Inc
Roche	Martin	National Bureau of Standards
Rodenberger	C. Mark	General Dynamics
Rohde	Leslie	Tektronix
Rolfe	Dr. Robert M.	Harris Corporation
Rosen	M. C.	Rosen Schneck Research Consultants
Rosenfeld	Larry	ICAD, Inc.
Rosol	Phillip	Martin Marietta Data Systems
Ross	Kathleen	Arthur D. Little
Rottler	Norman	US Navy
Rourke	Patrick W.	Newport News Shipbuilding
Rovani	William	The Kendall Co.
Ruedlinger	Joseph	IBM
Rumpf	Jim	Society of Automotive Engineers
Rutherford	Sherm	Data General
Ryan	Herbert F.	McDonnell Aircraft Company
Rygiel	Walter J.	Ford Motor Company
Salley	George	Boeing
Sanderson	Bob	Grumman Aerospace
Sandor	Douglas	Omni Construction Co, Inc.

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Sanford	David T.	Boeing
Sargent	James	PRC/GIS
Sarris	Anthony	Ontek Corporation
Savage	Charles	D. Appleton Co.
Scalf	Bill	Caterpillar, Inc.
Schachtner	Steven R.	Martin Marietta
Schaffran	Robert W.	Department of the Navy
Schall	Nancy	International TechneGroup Inc.
Scheller	Bob	McDonnell Aircraft Company
Schenck	Douglas	McDonnell Douglas
Schilli	Bruno	RPK
Schlechtendah	E. G.	Kernforschungszentrum Karlsruhe
Schmid	Randy	CADAM, Inc.
Schmidt	John	Middle Tennessee State University
Schneck	R. J.	311 Place des Boudeaux
Schneider	Charles W.	Richard Roeder Associates
Schneider	David	Chrysler Motors
Schoemaker	Jan	Alcater NV
Schroeter	Dirk	Martin Marietta Corporation
Schuldt	Ronald	Martin Marietta
Schumacher	Art	Ford Motor Company
Schwartz	A. M.	Fluor Engineers, Inc.
Scott	Ronald	McDonnell Douglas
Sedwick	Michael T.	AT&T
Seibert	Gary M.	US Army Corps of Engineers
Selvig	E. R.	Lockheed-Aeronautical Systems Co.
Seymour	Henry A.	Martin Marietta
Shak	Arnold	Grumman Data Systems
Shannon	Dan	Martin Marietta Data Systems
Shaver	Jerry	Gerber Systems Technology
Shaw	Nigel K.	Leeds University
Shematek	Joseph R.	Northrop Corporation
Sheridan	Jim	General Dynamics
Sherin	Dave	South Carolina Research Authority
Shields III	Richard V.	Ingalls Shipbuilding
Shih	Chia Hui	SDRC
Shoji	Toshiaki	Mistubishi Heavy Industry Ltd.
Showalton	Steve	
Shrivastava	Arvind	Chisholm Institute of Technology
Shuford	Annette	DTNSRDC
Siemon	Mark	U.S. Navy
Sieveking	Tristan	McDonnell Aircraft Co.
Sikorsky	Jon	Florida Power & Light Co.
Simha	Hln.	Sundaram-Clayton Ltd.
Simon	John C.	McDonnell Douglas MIS Co
Simpson	Dave	CADAM Inc.
Simpson	Frank	CADKEY Inc.
Sindt	James	Honeywell
Singh	J.	Computervision Corporation
Sison	Robert	CADAM, Inc.
Sivertsen	Ole Ivar	Sintef Machine Design Div.
Skeels	Jack A.	Northrop Aircraft
Skolnick	Richard	System Development Corp.

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Skutch	Maria	Westinghouse Electric Corp.
Slack	Francine	Electronic Data Systems
Slovensky	Leonard W.	McDonnell Douglas AIS Company
Smedley	Joseph	Valid Logic Systems
Smith	Bradford	NIST
Smith	Dennis P.	Genral Dynamics
Smith	Keith M.	Summa Technologies Inc.
Smith	Leon	LLNL
Smith	Myron	Eastman Kodak
Smith	Robert F.	CALMA Co.
Snodgrass	Neil	D. Appleton and Company
Snodgrass	William H.	Douglas Aircraft Corporation
Snyder	Jim	DOE/Martin Marietta Energy Systems
Sobczak	John	General Motors Corp.
Spencer	William	Nova Scotia Research
Stasny	Nita	LTV Aerospace
Steckel	Bruce	Western Forge Corp.
Stephans	Oren	Department of Navy, David Taylor Naval
Stewart	Hugh	Evans & Sutherland Ltd.
Stil	Guy	Aerospatiale
Stoddard	Charles L.	Pratt & Whitney
Stonecash	Jon	Control Data Corporation
Stoner	Joseph	DOE/LLNL
Straw	Andy	Eastman Kodak
Strong	Bob	Ontalogic
Stuart	J.W.	Arthur D. Little, Inc.
Stumps	Roger	Boeing
Sutton	John T.	IBM Corp.
Swain	Seldon	USAF
Swanger	Michael H.	Georgia Tech
Swanson	Kristofer	DOE/LLNL
Swearengen	John M.	Lockheed Aeronautical Systems Co.
Sweeney-Easte	Errol	PACCAR
Swistock	Edward	Schlosser Forge
Szmrecsanyi	Emery	Chrysler Motors
Tahmasebi	Farhad	STX
Tamura	Randall	IBM
Tang	Ed	Digital Equipment Corp.
Tarnoff	Nicholas	NBS
Tate	James "Steve"	Boeing Military
Taylor	Herb	Auto-Trol Technology
Taylor	Mark	Cincinnati Milacron
Taylor	Susan M.	LLNL
Tenarvitz	Henry	The CAD/CAM System for Moldmakers
Tenenbaum	Boris	WANG Laboratories
Tepe	Richard	International TechneGroup Inc
Terrill	Sherwin	NCR Corporation
Terry	Julia	DOE/Martin Marietta Energy Systems
Theilen	David	DOE/Allied Bendix Aerospace
Thomas	Debbie	Rutherford Appleton Laboratory
Thompson	Gerald A.	Hughes Aircraft
Thompson	J. Garth	Kansas State University
Thompson	Paul	Control Data Corporation

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Thompson	Vince E.	Az-Tech Software
Thompson, Jr.	C.B. "Bill"	Southern Company Services Inc
Thompson, Jr.	Clifton B.	Southern Company Services
Thomsen	Joseph	Northrop Corporation
Timmer	Henry G.	Northrop
Titterington	Dick	Boeing Computer Services
Tittle	Fremont	Control Data Corporation
Tolbert	Robert	Newport News Shipbuilding
Torossian	Julie A.	EDS
Tovar	Luis	General Dynamics
Traczek	Christine M.	Douglas Aircraft Company
Traff	Rod	Martin Marietta Data Systems
Trippner	Dietmar	BMW AG
Tully	Jim	Racal-Redac Systems Ltd.
Tur	Larry	Naval Ordnance Station
Turcotte	Bill	IGES Data Analysis Corp
Turner	James A.	University of Michigan
Turner	Monica	CADAM Inc.
Tyler	Joan	NIST
Uchida	Kohtaro	RICOH Co., Inc.
Valters	Indy	Interface Technologies Inc.
Van Hee	Barbara	Monsanto Enviro-Chemical Systems Inc.
Van Maanen	Jan	Rutherford Appleton Laboratory
Van Roessell	Jan W.	EROS DATA CENTER
Vander Schaaf	James R.	Bath Iron Works
Vassau	Ronald	The Aerospace Corp.
Veasey	Byron	E-Systems
Vergeest	Joris	Delft Univ. of Technology
Vermilyea	Gail	Vertek Associates
Vickers	Donald L.	DOE/LLNL
Voegeli	Tom	Caterpiller Inc.
Wajih	Reza	SoftCAD, Inc.
Walkinshaw	Steven	US Dept. of Transportation
Walters	Craig	McDonnell Douglas
Wandmacher	Richard R.	General Motors
Wang	Douglas	Enertronics Research Inc.
Wang	Kuei	Eastman Kodak Co.
Wapman	Derek	DOE/LLNL
Warthen	Barbara	GE CALMA
Wassmuth	Calvin L.	Pratt & Whitney
Watson	George	Auto-trol Technology
Watts	Steven L.	Int'l TechneGroup Inc.
Weaver	Earl P.	Ballistics Research Laboratory
Wechsler	Donald	MITRE Corporation
Weebe	Thomas K.	Northrop Advanced Systems
Weick	Werner	PROCAD GmbH
Weidemann	Christian	IVF Co.
Weiss	Jerry A.	McDonnell Aircraft Co.
Weissflog	Uwe	IBM Corp.
Weller	George	Steelcase Inc.
Wells	Bob	NC State Transportation
Wells	Michael	Rosetta Technologies, Inc.
Wenzel	Bernd	GEMAP

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Wester	Ronald O.	General Electric Co.
Whelan	Anna	McDonnell Douglas
Whelan	Tracy	CADAM Inc.
Whilhelmson	Jack L.	SCRA
Whitaker	Gary	Ford Motor Company
White	Cynthia V.	NAVSEA
White	Richard M.	Naval Electronic Systems JUL87
White	Robert	Computervision Corp.
Whitehurst	George	NASA Langley Research Center
Whiteman	Mike	Deere & Co.
Whitt	Gary	Intergraph Corp.
Whittaker	Harry	DTNSRDC
Wickens	Lawrence	CAD/CAM Data Exchnge
Wilhelmson	Jack	South Carolina Research Authority
Wilk	Richard	Eaton Corporation
Willcox	Gary	Int'l TechneGroup Inc.
Williams	Anne	McDonnell Aircraft Co.
Willis	Michael	General Dynamics
Willis	Robert	National Computer Graphics Assoc.
Wilson	Peter	General Electric Co
Wilson	Robert M.	DOE/Martin Marietta
Wimple	Carolyn	DOE/LLNL
Winfrey	Richard C.	Digital Equipment Corp.
Winn	Alan R.	Air Force Materials Lab
Wittenberg	Edward R.	Ford Motor Company
Wix	Jeff	Wix McLelland Ltd.
Wolf	Robert	Xerox Corp.
Wolfe	Bruce	Boeing Computer Services
Wolfe	L. Stephen	Computer Aided Design Report
Wong	Vivian	Douglas Aircraft
Woodall	Alison	CADDETC - Leeds University
Wooley	Dan	Newport News Shipbuilding
Wright	Debbie	Sikorsky Aircraft
Wright	Thomas	NIST
Wright	Tom	Stone and Webster Engineering
Wright	Wayne "Les"	General Dynamics Corp - Data Sys Div
Wu	Hui-Chung	Digital Equipment Corp.
Wu	Ting-Yuan	IBM
Wuerth	Peter	Boeing Computer Services
Yajima	Akio	Hitachi
Yang	Sheree	Ford Aerospace & Communications Co
Yang	William J.	William J. Yang & Associates
Yang	Yuhwei	D. Appleton Co.
Yatchak	John	Martin Marietta
Yaworsky	George M.	
Yee	David	Scribe Systems
Yee	Susan K.	General Dynamics
Yinger	Michael A.	Yinger & Associates
Young	W.H.	RCFC
Zapomueel	Herbert	Siemens AG ZTZFAAUT4
Zaretski	Philip	EDS
Zastera	Eugene	US Army
Zech	Deborah Lynne	Ford Motor Company

Members of the IGES/PDES Organization

LNAME	FNAME	COMPANY
Zimmerman	John	DOE/Allied-Signal Inc.
Ziolko	Glen A.	LTV Aerospace & Defense
Zobrist	Dr. George	University of Missouri-Rolla

Introduction

This interim report is a compilation of documents associated with the effort to create the Product Data Exchange Specification (PDES) standard. Collectively, these documents form the first working draft of the standard. The documents contained in this report have been accepted by the committee chairmen of the IGES/PDES Organization as being sufficiently complete and sufficiently correct to warrant broad technical review by the national and international participants in the standardization process. Publication of the documents in this interim report freezes the content and makes it available in a stable, referenceable form suitable for comprehensive review.

The Product Data Exchange Specification is being developed by the IGES/PDES Organization as a proposed standard. Throughout this document there are references to another proposed standard called the Standard for The Exchange of Product data. This second proposed standard which we will refer to as the ISO Product Data Standard is a work item in ISO. The IGES/PDES organization is submitting the specification to ISO to be put through its formal standardization procedures. The responsible ISO committee is TC184/SC4/WG1. At this point in time, PDES and the ISO Product Data Standard are the same standard and it is the intention of all parties to keep them the same.

The various documents compiled into this NISTIR are what are known as "N" documents. These "N" documents each have a separate number assigned by the ISO working group. Eventually the "N" numbers will disappear and be replaced by more coherent clause numbers (analogous to sections or chapters.) The table of contents shows the current correspondence between "N" numbers and future clause and annex portions of the proposed standard.

Please note that publication of this document is primarily an effort to create a stable source of material which can be used for further development. It is clearly premature to be used for any type of procurement action and is subject to change.

Title: Introduction, Scope and Definitions

Owner: Jon Owen

Date: 31 October 1988

Corresponding ISO Document Number: N283

FOREWORD, INTRODUCTION, SCOPE, REFERENCES and DEFINITIONS

DOCUMENT NUMBER: Clauses 0-3 **VERSION:** 1.0

TITLE: Foreword, Introduction, Scope, References and Definitions

ABSTRACT:

This document gives a definition of the minimum capability which must be implemented in software translators which claim to read in or write out the physical file format of this standard.

KEY WORDS:

Definitions, Foreword, Introduction, References, Scope

DATE: 31 October 1988

OWNER: Jon Owen

PHONE NUMBER: +44 532 334 455

FAX NUMBER: +44 532 445 270

ISO REPRESENTATIVE: Jon Owen

STATUS: Working

INDUSTRIAL AUTOMATION SYSTEMS - EXCHANGE OF PRODUCT MODEL DATA - REPRESENTATION AND FORMAT DESCRIPTION

First Working Draft
2 November 1988

PREPARED FOR TC184/SC4/WG1 TOKYO MEETING

28 November - 2 December 1988

Foreword

This document has been prepared by Working Group One of Sub-committee Four ("Exchange of Product Model Data") of ISO Technical Committee 184 ("Industrial Automation Systems"); ISO TC184/SC4/WG1.

The preparation of this document has benefitted from the contribution of Computer-Aided Design Interfaces project (CAD*I), funded by the European Community (Esprit project 322), and the Initial Graphics Exchange Specification/Product Data Exchange Specification organization, in the United States of America.

This is the first edition of the standard.

This document incorporates the experience gained from the development of the following national standards: Initial Graphics Exchange Specification, *IGES* (ANSI Y14.26-M-1987, "Digital Representation for Communication of Product Definition Data", 1987); Standard d'Échange et de Transfert, *SET* (Afnor Z68-300, "External Representation of Product Definition Data: Data Exchange and Transfer Standard Specification", version 85-08, August 1981); and Verband der Automobilindustrie-Flächen-Schnittstelle, *VDA-FS* (DIN 66 301, "Industrielle Automation - Rechnergestützter Konstruieren: Format zum Austauch geometrischer informationen", July 1986).

The following annexes are normative:

- Information Modelling Language: Express
- Physical File Structure
- Mapping from Logical Layer to Physical Layer

The following annexes are informative:

- Development Support
- User Support
- Bibliography

Introduction

The Standard for the Exchange of Product Model Data is a neutral exchange mechanism capable of completely representing product definition data throughout the life cycle of a product.

There is an undeniable need to transfer product definition data in computer-readable form from one site to another. These sites may have one of a number of relationships between them (contractor and subcontractor, customer and supplier); the information invariably needs to iterate between the sites, retaining both data completeness and functionality, until it is ultimately archived. The most cost-effective manner to encapsulate such information is in a neutral format, independent of any Computer-Aided Design (CAD) software system.

The experience to date in using digital product data has been reasonably successful only when there has been careful planning and testing of the data paths used. The effort has been slow because of the need for personnel involved to understand the idiosyncrasies of the CAD systems and to develop competence in using the translation software: success can be attributed to the collaboration of sender and receiver. This is a reflection of present practice world-wide and underlines that the present use of digital product data is for exchange.

The term "exchange" denotes an activity characterized by the following conditions:

NOTES

1 the data transfer occurs at a single point in time

2 complete information is available on the computing systems available

3 sender and receiver can communicate to solve any problems that are encountered

The methodology underlying the development of this international standard includes the use of reference models, formal definition languages and a three layer architecture (application, logical, physical). To provide a rigorous specification, a number of formal concepts are used to describe both the "logical" content of the information and a well-defined syntax (grammar) at the physical layer. This provides the facility not only for neutral file exchange, but also as a basis for implementing and sharing product databases.

It should be emphasized that the goal is to create a complete representation of a product as stored in a database and not merely a graphical or visual representation, existing standards for which are already established (for example: Graphical Kernel System, ISO 7942).

At the application level, models specific to the activities of one application are developed. These individual application reference models are then mapped onto the Logical Layer, which is, in turn, mapped onto the Physical Layer. In practice, it is recognised that some topics are common to more than one application; specific "Topical models" are developed for these areas.

The logical layer is formulated in an information description language (called Express) which includes facilities for declarations similar to record definitions in the computer programming languages Pascal and Ada. The information content of an entity (i.e. all relevant facts about an entity) are defined in the logical layer, independent of the many possible ways in which the data might be implemented.

The physical layer (concerned with the representation on a sequential file) has its formal syntax expressed in Wirth Syntax Notation (WSN). This is an unambiguous, context-free formal grammar, thereby easing the problem of processing such a file by computer.

The standard itself comprises Clause Four and the Normative Annexes (A, B, and C). In addition, there is a series of reference models for individual applications (described in the associated Technical Report). These are integrated into a single application reference model before mapping on to the logical layer, and thence the physical layer.

TABLE OF CONTENTS

For The First Working Draft of STEP 1.0

Clause

NORMATIVE ANNEXES

Annex

A. Information Modeling Language
B. Physical File Structure
C. Mapping to Physical

INFORMATIVE ANNEXES

Annex

D. Information Models
E. Development Support

INDUSTRIAL AUTOMATION SYSTEMS - EXCHANGE OF PRODUCT MODEL DATA - REPRESENTATION AND FORMAT DESCRIPTION

1 Scope

This international standard specifies a representation and format description for the exchange of product model data.

Application reference models cover finite element (FE) modelling, presentation, drafting, product structure configuration management (PSCM), form features, electrical and electronic applications, and architecture, engineering and construction (AEC).

Finite element modelling is limited to models for linear elastic and thermal FE analyses. The intent is to capture just the finite element model itself and not any of the pre- or post-processing information. Inclusion of any processing information, as well as non-linear material or geometric properties, is left to a later version of the standard.

Presentation shall provide the entities which support the visualisation of data contained in the neutral format file, augmented by instructions of the sender governing the appearance of product model data as well as supplementary annotations. The presentation entities shall form a common basis for the needs of any application.

Drafting defines entities that are needed to express, in a computer-intelligent form, the data which enables generation of a human- interpretable representation of Product Data. Initially, only mechanical products are included; however, future expansion for other applications will take place as their requirements become available.

Product Structure Configuration Management is limited primarily to engineering data. It is assumed, however, that the product described will be manufactured (and probably supported by an organization throughout its life).

The viewpoint represented in this data model is primarily that of configuration management of product structures. Engineering design and management are also strongly represented. The particular terms may vary depending on the nature of the enterprise, but the function is that of managing the way in which deliverable products are designed and configured.

The scope of this model is restricted by:

1. product structure is restricted to the relationship between physical components and assemblies.
2. configuration management applies only to product structure; i.e. manages the way assemblies and components are configured. This is initially limited to managing the way products are "intended" to be built, not how they are actually built. It is further restricted to address the configuration of specific units of product (serial or lot numbers), and not time ranges of products.
3. product structure will support more than one bill-of-materials view.
4. product structure and configuration management will support material information only with respect to stock items and make-from issues. This means that the current model will not include material property data, which is addressed elsewhere.

5. the primary goal is to capture the "as designed" information. This is initially limited to mechanical product issues, and no attempt at integration with other model areas (AEC, Electrical) has been attempted.

Form Features covers nominal shape of form features and intrinsic (size) tolerances of that shape. Other data about form features, such as non-shape properties, tolerances on the relation of the feature to other elements of shape, and processes are not covered. A form feature is regarded as a portion of the "skin" of a shape. Explicit feature representation is addressed in a general manner. The model deals with implicit representations in greater detail.

Electrical and electronic applications are limited, at present, to two specific models:

layered electrical product is intended to define the as-designed data about electrical products which may be expressed in terms of topological layers. Included are voids, boundaries and information joining or contained on layers. The model addresses printed wiring boards and includes entities which provide for their manufacture on a panel together with test coupons. The model has untested potential for describing printed wiring circuits, microwave striplines, flex harnesses, hybrid microelectronics, integrated circuits, and composite products.

functional (connectivity) deals with logical connectivity and the functional design hierarchy. Specifically, for combinatorial digital logic circuits, it accommodates data describing output changes in response to input stimuli, and propogation delays.

General AEC Reference Model provides a general framework for the organization of product data that are exchanged by various disciplines within architecture, engineering, and construction. These include: process plants, buildings, ships, sites, etc. The model will cover product data for the full product life cycle, from requirements through to demolition.

AEC Ship Structural Model is an application model for the structural aspects of the design and manufacture phases in the shipbuilding industry. The model encompasses the data that make up a ship's primary structure system.

The geometric topical model is distinguished from application-specific models and satisfies the requirements of several application reference models. The various geometric models define/represent the theoretically exact shape of an object and its position in space:

wireframe representation this comprises the points, curves and the possibly topological relationships necessary to define directed curves completely and unambiguously.

surface representation this comprises the points, curves, surfaces and possibly topological relationships necessary to define directed surfaces completely and unambiguously.

boundary representation this comprises the geometric and topological relationships necessary to define the nominal shape of the "bounded volume" of the product completely and unambiguously. Examples of geometry include those in the surface and wireframe models; examples of topological constructs include shell, face, edge, vertex.

constructive solid geometry this comprises the set of primitive volumes, each of which contains the dimensions (which define the shape of the primitive) and axes (which define its local co-ordinate system), the operations and the sequence of operations that yield a complete and unambiguous product shape.

tolerances whereas dimensions on an engineering drawing define the theoretically exact shape of a three-dimensional object, tolerances on those dimensions define the allowable deviation of the dimension from the nominal value as measured on the manufactured object. The reference model captures the semantic content of the tolerances as specified in ANSI Y14.5M (1982), ISO 1660 and ISO 1101, which give the standard representation for dimensions and tolerances on engineering drawings. (These specifications are considered to be functionally identical for the purposes of this effort.)

Computing tolerances, process tolerances, pictorial representation of tolerances, dimensioning practices and non-mechanical part tolerances (for example: electrical component values) are excluded from the scope of this model.

The product model to which tolerances are applied is assumed to be complete, unambiguous and usually three-dimensional definitions of the shape of the desired object, represented by a boundary representation (Brep) solid model, an equivalent bounded, surfaced wireframe model, or other functionally equivalent geometric model.

Logical Layer The entities in the logical layer are defined using Express (Annex A), which is designed to satisfy:

1. modelling the things of interest
2. defining the constraints that are to be imposed upon them
3. defining the operations which establish how they are to be used
4. modelling in a computer-sensible manner (i.e. processable by computer)

2 Normative References

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO 6937 Coded Character Set for Text Communication

3 Definitions

For the purposes of this International Standard, the following definitions apply.

abstract schema those definitions which describe the content of the data and the relationships between the various elements or components of the data; a product of the application layer activity

application area a set of applications doing the same or closely related functions that use data which is logically the same

conceptual level all aspects which deal with the interpretation (meaning) and manipulation of information describing an entity world in an information system

concrete schema a derivation of the abstract schema which enables an automatic mapping to be effected onto the physical file

product data all data elements necessary to define the geometry, the function, and the behaviour of a piece part or an assembly of parts over its entire lifespan; the term includes all product definition data elements as well as additional elements for reliability and maintenance

product definition data the totality of data elements required to completely define a product; this includes geometry, topology, relationship, tolerances, attributes and features necessary to completel define a component part or an assembly of parts for the purpose of design, analysis, manufacture, test, and inspection

reference model a standard representation created for use in constructing something; often shorthand for "application reference model"

4 Requirements

The standard is the set of functions described in Clause Four and Annexes A, B, and C. An implementation of this international standard shall support all or a subset of these functions, as described in Clause Four.

Title: ISO STEP Baseline Requirements Document (IPIM)

Owners: P.R. Wilson and P.R. Kennicott

Date: 27 October 1988

Corresponding ISO Document Number: N284

ISO TC184/SC4/WG1 Document October 27, 1988

(Working/Recommended)

EXTERNAL REPRESENTATION OF PRODUCT DEFINITION DATA

DOCUMENT NUMBER 4 **Version Tokyo**

TITLE: Integrated Product Information Model

ABSTRACT: This document contains the definition of the STEP standard logical schema for Product Definition Data and comprises the Product Definition Data logical schema for Resource, Life Cycle and Application Information Models.
The schema is defined using the EXPRESS language as specified in document N268.

KEY WORDS: Integrated Model, EXPRESS, Schema Overview, Miscellaneous Resources, Units, Geometry, Topology, Shape Representation, Form Features, Shape Interface, Tolerances, Materials, Presentation, Life Cycle, Applications.

DATE: October 27, 1988

OWNER: SG6,
Editors P. R. Wilson & P. R. Kennicott
General Electric CR&D
Building K1, Room KW-D238
River Road
Schenectady, NY 12309
U.S.A

TELEPHONE: (518) 387-5241

TELEX :

ISO REPRESENTATIVE: Peter Wilson

STATUS: Working/Recommended

Contents

List of Tables

List of Figures

4 Requirements: Integrated Product Information Model

4.1 Introduction

This, and following, Sections define the *Integrated Product Information Model*, also referred to as the *Logical Schema*, for Version 1 of this standard.

This Section defines what is referred to as the "Core Model"; that is, the basic information that is required to define the shape of a product, plus other information that is likely to be common across many applications.

Another Section defines the information pertinent to particular "applications" that are necessary to the production of a product.

The technical content of the Integrated Product Information Model (IPIM) has been developed by many technical committees working in parallel, each producing a *Topical Information Model* (TIM) of their particular technology area; these have been documented elsewhere. Within the IPIM, these individually developed TIMs have been reorganised and integrated (collated) into a coherent whole. The objective of the integration process was to produce a single, minimally redundant, unambiguous, and complete information model for the standard. The IPIM has been designed to be independent of any particular implementation technology for data exchange. During the course of this process changes have had to be made to portions of some of the Topical Models to meet the objectives; the information content exhibited by the TIMS, though, has been retained within the IPIM.

The technical content of the IPIM is described both formally and informally. The EXPRESS language has been used to define formally the Integrated Product Information Model. Additional textual descriptions are supplied to aid the reader in interpreting the formal definition.

4.1.1 Schema Overview

This section provides an overview of the set of sub-schemas forming the current Integrated Product Information Model and their interrelationships. The language used to describe this in an informal manner is in the EXPRESS style, but is not EXPRESS.

The overall schema is broken down into a set of sub-schemas which may, in turn, be further partitioned. The schemas have been deliberately kept "open". All schemas, except the outermost EXPORT everything. Each enclosing schema ASSUMEs all its directly enclosed schemas. The effect of this is that the overall model is independent of the partitioning into schemas.

Interrelations between the schemas can be determined by examining the ASSUME clauses.

```
SCHEMA ipim_schema
  ASSUME : ipim_resources_schema
  ASSUME : ipim_geometry_schema
  ASSUME : ipim_topology_schema
  ASSUME : ipim_shape_schema
  ASSUME : ipim_design_shape_schema
  ASSUME : ipim_nominal_shape_schema
  ASSUME : ipim_solids_schema
  ASSUME : ipim_shape_interface_schema
  ASSUME : ipim_features_schema
  ASSUME : ipim_tolerances_schema
  ASSUME : ipim_material_schema
  ASSUME : ipim_presentation_schema
```

```
ASSUME : ipim_life_cycle_schema
ASSUME : ipim_applications_schema
ASSUME : ipim_product_manifestation_schema
ASSUME : ipim_drafting_schema
ASSUME : ipim_mechanical_product_schema
ASSUME : ipim_pscm_schema
ASSUME : ipim_aec_schema
ASSUME : ipim_aec_core_schema
ASSUME : ipim_ship_structure_schema
ASSUME : ipim_electrical_schema
ASSUME : ipim_electrical_functional_schema
ASSUME : ipim_electrical_schematic_schema
ASSUME : ipim_lep_schema
ASSUME : ipim_analysis_schema
ASSUME : ipim_fem_schema
ASSUME : ipim_data_transfer_schema

SCHEMA ipim_resources_schema
  ASSUME : ipim_geometry_schema
  ASSUME : ipim_topology_schema
  ASSUME : ipim_shape_schema
  ASSUME : ipim_tolerances_schema
  ASSUME : ipim_material_schema
  ASSUME : ipim_presentation_schema

  SCHEMA ipim_geometry_schema
  END_SCHEMA : SCHEMA ipim_geometry_schema

  SCHEMA ipim_topology_schema
    ASSUME : ipim_geometry_schema
  END_SCHEMA : SCHEMA ipim_topology_schema

  SCHEMA ipim_shape_schema
    ASSUME : ipim_design_shape_schema
    ASSUME : ipim_nominal_shape_schema
    ASSUME : ipim_features_schema
    ASSUME : ipim_shape_interface_schema

    SCHEMA ipim_design_shape_schema
      ASSUME : ipim_nominal_shape_schema
      ASSUME : ipim_features_schema
      ASSUME : ipim_tolerances_schema
      ASSUME : ipim_lep_schema
      ASSUME : ipim_geometry_schema
    END_SCHEMA : ipim_design_shape_schema

    SCHEMA ipim_nominal_shape_schema
      ASSUME : ipim_geometry_schema
      ASSUME : ipim_topology_schema
```

```
      ASSUME : ipim_solids_schema

      SCHEMA ipim_solids_schema
        ASSUME : ipim_geometry_schema
        ASSUME : ipim_topology_schema
      END_SCHEMA : ipim_solids_schema

    END_SCHEMA : ipim_nominal_shape_schema

    SCHEMA ipim_features_schema
      ASSUME : ipim_shape_interface_schema
      ASSUME : ipim_nominal_shape_schema
      ASSUME : ipim_geometry_schema
      ASSUME : ipim_tolerances_schema
    END_SCHEMA : ipim_features_schema

    SCHEMA ipim_shape_interface_schema
      ASSUME : ipim_design_shape_schema
      ASSUME : ipim_features_schema
      ASSUME : ipim_geometry_schema
      ASSUME : ipim_nominal_shape_schema
      ASSUME : ipim_solids_schema
      ASSUME : ipim_topology_schema
    END_SCHEMA : ipim_shape_interface_schema

  END_SCHEMA : ipim_shape_schema

  SCHEMA ipim_tolerances_schema
    ASSUME : ipim_shape_interface_schema
    ASSUME : ipim_features_schema
    ASSUME : ipim_geometry_schema
  END_SCHEMA : ipim_tolerances_schema

  SCHEMA ipim_material_schema
  END_SCHEMA : ipim_material_schema

  SCHEMA ipim_presentation_schema
    ASSUME : ipim_geometry_schema
    ASSUME : ipim_drafting_schema
  END_SCHEMA : ipim_presentation_schema

END_SCHEMA : ipim_resources_schema

SCHEMA ipim_life_cycle_schema
  ASSUME : ipim_resources_schema
  ASSUME : ipim_applications_schema
END_SCHEMA : ipim_life_cycle_schema

SCHEMA ipim_applications_schema
```

```
ASSUME : ipim_product_manifestation_schema
ASSUME : ipim_mechanical_product_schema
ASSUME : ipim_aec_schema
ASSUME : ipim_electrical_schema
ASSUME : ipim_analysis_schema
ASSUME : ipim_data_transfer_schema

SCHEMA ipim_product_manifestation_schema
  ASSUME : ipim_drafting_schema

  SCHEMA ipim_drafting_schema
    ASSUME : ipim_geometry_schema
    ASSUME : ipim_presentation_schema
  END_SCHEMA : ipim_drafting_schema

END_SCHEMA : ipim_product_manifestation_schema

SCHEMA ipim_mechanical_product_schema
  ASSUME : ipim_pscm_schema

  SCHEMA ipim_pscm_schema
    ASSUME : ipim_shape_interface_schema
    ASSUME : ipim_material_schema
    ASSUME : ipim_geometry_schema
  END_SCHEMA : ipim_pscm_schema

END_SCHEMA : ipim_mechanical_product_schema

SCHEMA ipim_aec_schema
  ASSUME : ipim_aec_core_schema
  ASSUME : ipim_ship_structure_schema

  SCHEMA ipim_aec_core_schema
    ASSUME : ipim_geometry_schema
    ASSUME : ipim_shape_interface_schema
    ASSUME : ipim_resources_schema
    ASSUME : ipim_material_schema
    ASSUME : ipim_drafting_alt1_schema
    ASSUME : ipim_fem_schema
  END_SCHEMA : ipim_aec_core_schema

  SCHEMA ipim_ship_structure_schema
    ASSUME : ipim_resources_schema
    ASSUME : ipim_geometry_schema
    ASSUME : ipim_material_schema
  END_SCHEMA : SCHEMA ipim_ship_structure_schema

END_SCHEMA : ipim_aec_schema
```

```
    SCHEMA ipim_electrical_schema
      ASSUME : ipim_electrical_functional_schema
      ASSUME : ipim_electical_schematic_schema
      ASSUME : ipim_lep_schema

      SCHEMA ipim_electrical_functional_schema
        ASSUME : ipim_shape_interface_schema
      END_SCHEMA : ipim_electrical_functional_schema

      SCHEMA ipim_electrical_schematic_schema
        ASSUME : ipim_geometry_schema
        ASSUME : ipim_drafting_schema
      END_SCHEMA : ipim_electrical_schematic_schema

      SCHEMA ipim_lep_schema
        ASSUME : ipim_geometry_schema
        ASSUME : ipim_topology_schema
        ASSUME : ipim_material_schema
        ASSUME : ipim_design_shape_schema
      END_SCHEMA : ipim_lep_schema

    END_SCHEMA : ipim_electrical_schema

    SCHEMA ipim_analysis_schema
      ASSUME : ipim_fem_schema

      SCHEMA ipim_fem_schema
        ASSUME : ipim_shape_interface_schema
        ASSUME : ipim_data_transfer_schema
        ASSUME : ipim_material_schema
        ASSUME : ipim_pscm_schema
      END_SCHEMA : ipim_fem_schema

    END_SCHEMA : ipim_analysis_schema

    SCHEMA ipim_data_transfer_schema
      ASSUME : everything
    END_SCHEMA : ipim_data_transfer_schema

  END_SCHEMA : ipim_applications_schema

END_SCHEMA : ipim_schema
```

4.2 Resource Schema

This is Part 1 of the Integrated Product Information Model and contains the general TYPE and FUNCTION definitions used throughout the IPIM and the general Resource entities.

```
*)
SCHEMA ipim_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_resources_schema,
         ipim_geometry_schema,
         ipim_topology_schema,
         ipim_shape_schema,
         ipim_design_shape_schema,
         ipim_nominal_shape_schema,
         ipim_solids_schema,
         ipim_shape_interface_schema,
         ipim_features_schema,
         ipim_tolerances_schema,
         ipim_material_schema,
         ipim_presentation_schema,
         ipim_life_cycle_schema,
         ipim_applications_schema,
         ipim_product_manifestation_schema,
         ipim_drafting_schema,
         ipim_mechanical_product_schema,
         ipim_pscm_schema,
         ipim_aec_schema,
         ipim_aec_core_schema,
         ipim_ship_structure_schema,
         ipim_electrical_schema,
         ipim_electrical_functional_schema,
         ipim_electrical_schematic_schema,
         ipim_lep_schema,
         ipim_analysis_schema,
         ipim_fem_schema,
         ipim_data_transfer_schema);
(*
```

4.3 Types and Functions

4.3.1 Type Definitions

This Section contains the TYPE definitions which are common to more than one of the schemas.

4.3.1.1 AGGREGATES

Listed here are several types of "generic" aggregates. These are provided for use as argument types in subsequent FUNCTION and PROCEDURE definitions.

4.3.1.1.1 GENERIC LIST

```
*)
TYPE generic_list = LIST [0 : #] OF GENERIC;
END_TYPE;
(*
```

4.3.1.1.2 GENERIC SET

```
*)
TYPE generic_set = SET [0 : #] OF GENERIC;
END_TYPE;
(*
```

4.3.1.1.3 LIST OF INTEGER

```
*)
TYPE list_of_integer = LIST [0 : #] OF INTEGER;
END_TYPE;
(*
```

4.3.1.1.4 LIST OF REAL

```
*)
TYPE list_of_real = LIST [0 : #] OF REAL;
END_TYPE;
(*
```

4.3.1.1.5 SELECT AGGREGATE OF NUMBER

```
*)
TYPE select_aggregate_of_number = select_aggregate OF NUMBER;
END_TYPE;
(*
```

4.3.1.2 COORDINATE PAIR

```
*)
```

```
TYPE coordinate_pair = ARRAY [1:2] OF REAL;
END_TYPE;
(*
```

4.3.1.3 COORDINATE TRIPLE

```
*)
TYPE coordinate_triple = ARRAY [1:3] OF REAL;
END_TYPE;
(*
```

4.3.1.4 COORDINATE SYSTEM TYPE

```
*)
TYPE coordinate_system_type = ENUMERATION OF
      (rh_rectangular,
       rh_cylindrical,
       rh_spherical,
       lh_rectangular,
       lh_cylindrical,
       lh_spherical);
END_TYPE;
(*
```

4.3.1.5 CURVE OR SURFACE

```
*)
TYPE curve_or_surface = SELECT
      (curve,
       surface);
END_TYPE;
(*
```

4.3.1.6 EXPRESS RESERVED WORDS

These types are used as a means of referring to EXPRESS reserved words.

4.3.1.6.1 EXPRESS INTEGER

```
*)
TYPE express_integer = INTEGER;
END_TYPE;
(*
```

4.3.1.6.2 EXPRESS STRING

```
*)
TYPE express_string = STRING;
END_TYPE;
```

```
(*
```

4.3.1.6.3 EXPRESS NULL

```
*)
TYPE express_null = NULL;
END_TYPE;
(*
```

4.3.1.6.4 EXPRESS NUMBER

```
*)
TYPE express_number = NUMBER;
END_TYPE;
(*
```

4.3.1.6.5 EXPRESS REAL

```
*)
TYPE express_real = REAL;
END_TYPE;
(*
```

4.3.1.6.6 EXPRESS ARRAY

```
*)
TYPE express_array = ARRAY;
END_TYPE;
(*
```

4.3.1.6.7 EXPRESS LIST

```
*)
TYPE express_list = LIST;
END_TYPE;
(*
```

4.3.1.6.8 EXPRESS SET

```
*)
TYPE express_set = SET;
END_TYPE;
(*
```

4.3.1.6.9 EXPRESS LOGICAL

```
*)
TYPE express_logical = LOGICAL;
END_TYPE;
```

```
(*
```

4.3.1.7 INFINITY

This TYPE provides a means of expressing the mathematical concept of *infinity*.

```
*)
TYPE infinity = NUMBER;
(*
```

4.3.1.8 LEADER LINE TERMINATION ENUMERATION

```
*)
TYPE leader_line_terminator_enumeration = ENUMERATION OF
     (arrowhead,
      wavy_line,
      circle);
END_TYPE;
(*
```

4.3.1.9 ROTATIONAL DIRECTION

```
*)
TYPE rotational_direction = ENUMERATION OF
    (clockwise,
     counterclockwise);
END_TYPE;
(*
```

4.3.1.10 SECURITY CLASS LEVEL

```
*)
TYPE security_class_level = ENUMERATION OF
     (unclassified,
      confidential,
      secret,
      top_secret);
END_TYPE;
(*
```

4.3.1.11 SELECT AGGREGATE

```
*)
 TYPE select_aggregate = SELECT
      (express_array,
       express_list,
       express_set);
 END_TYPE;
(*
```

4.3.1.12 SELECT FACE OR SUBFACE

```
*)
TYPE select_face_or_subface = SELECT
     (face,
      subface);
END_TYPE;
(*
```

4.3.1.13 SIZE

```
*)
TYPE size = SELECT
    (express_real,
     toleranced_real);
END_TYPE;
(*
```

4.3.1.14 TOPOLOGY AGGREGATES

Listed here are some aggregates of topological entities.

4.3.1.14.1 LIST OF VERTEX

```
*)
TYPE list_of_vertex = LIST [0 : #] OF vertex;
END_TYPE;
(*
```

4.3.1.14.2 SET OF VERTEX

```
*)
TYPE set_of_vertex = SET OF vertex;
END_TYPE;
(*
```

4.3.1.14.3 LIST OF EDGE

```
*)
TYPE list_of_edge = LIST [0 : #] OF edge;
END_TYPE;
(*
```

4.3.1.14.4 SET OF EDGE

```
*)
TYPE set_of_edge = SET OF edge;
END_TYPE;
(*
```

4.3.1.14.5 LIST OF EDGE LOGICAL STRUCTURE

```
*)
TYPE list_of_edge_logical_structure = LIST [0 : #] OF
                                        edge_logical_structure;
END_TYPE;
(*
```

4.3.1.14.6 SET OF EDGE LOGICAL STRUCTURE

```
*)
TYPE set_of_edge_logical_structure = SET [0 : #] OF
                                        edge_logical_structure;
END_TYPE;
(*
```

4.3.1.14.7 LIST OF LOOP

```
*)
TYPE list_of_loop = LIST [0 : #] OF loop;
END_TYPE;
(*
```

4.3.1.14.8 SET OF LOOP

```
*)
TYPE set_of_loop = SET OF loop;
END_TYPE;
(*
```

4.3.1.14.9 LIST OF FACE

```
*)
TYPE list_of_face = LIST [0 : #] OF face;
END_TYPE;
(*
```

4.3.1.14.10 SET OF FACE

```
*)
TYPE set_of_face = SET OF face;
END_TYPE;
(*
```

4.3.1.15 TRUE FALSE OR UNDEFINED

A selection from LOGICAL or UNDEFINED.

```
*)
```

```
TYPE true_false_or_undefined = SELECT
      (express_logical,
       undefined);
END_TYPE;
(*
```

4.3.2 Function Definitions

The EXPRESS language has a number of built in functions. This Section describes additional functions required for the the definition and constraints on the information model schema.
The Section has been reorganised so that functions that are only applicable to a single schema are placed in that schema, leaving only functions applicable to multiple schemas here.

4.3.2.1 AGREEMENT

Two entities are in **agreement** if their directions, as evaluated by the **direction fun** function, are the same.

```
*)
FUNCTION agreement(arg_1, arg_2: GENERIC): LOGICAL;
  IF direction_fun(arg_1) <> direction_fun(arg_2) THEN
    RETURN(FALSE);
  ELSE
    RETURN(TRUE);
  END_IF;
END_FUNCTION;
(*
```

4.3.2.2 ARCWISE CONNECTED

An entity is arcwise connected if any two arbitrary points in its domain can be connected by an arc within the domain that does not cross a boundary of the domain. The function **arcwise connected** returns TRUE if its argument is arcwise connected.

```
*)
FUNCTION arcwise_connected(arg: GENERIC):LOGICAL;
END_FUNCTION;
(*
```

4.3.2.3 CLOSED

An entity of dimensionality D is topologically closed if it divides R^{D+1} into exactly two regions. The function **closed** returns TRUE if its argument is closed.

```
*)
FUNCTION closed(arg: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

4.3.2.4 IS COAXIAL

The function is coaxial returns TRUE if its arguments are coaxial.

```
*)
FUNCTION is_coaxial (arg_1,arg_2: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

4.3.2.5 IS CONCAVE

The function is concave returns TRUE if its arguments form a concave shape.

```
*)
FUNCTION is_concave(arg: generic_list): LOGICAL;
END_FUNCTION;
(*
```

4.3.2.6 IS CONICAL

The function is conical returns TRUE if its argument is conical.

```
*)
FUNCTION is_conical(arg: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

4.3.2.7 CONNECTED

Two entities are connected if any portion of their domains and/or boundaries coincide in space. The function connected returns TRUE if the input arguments are connected.

```
*)
FUNCTION connected(arg_1,arg_2: GENERIC): LOGICAL;
  LOCAL
    result : LOGICAL := FALSE;
  END_LOCAL;
  IF (distance(arg_1, arg_2) = 0) THEN
    result := TRUE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.3.2.8 IS CONVEX

The function is convex returns TRUE if its arguments form a convex shape.

```
*)
FUNCTION is_convex(arg: generic_list): LOGICAL;
END_FUNCTION;
(*
```

4.3.2.9 COORDINATED LISTS

There are many instances of entities which have two or more attributes of type LIST and the entries in the LISTs must be in one to one correspondance. The function **coordinated lists** may be used to specify this constraint.

The function takes two arguments, both of type LIST, and returns TRUE if the entries in the pair of LISTs are in correspondance with each other.

```
*)
FUNCTION coordinated_lists(arg1, arg2 : generic_list) : LOGICAL;
  LOCAL
    result : LOGICAL := TRUE;
  END_LOCAL;
  IF SIZEOF(arg1) <> SIZEOF(arg2) THEN
    result := FALSE;
  END_IF;
  (* pseudo code here for checking LIST entry correspondance. *)
  RETURN(result);
END_FUNCTION;
(*
```

4.3.2.10 COORDINATE SPACE

This function returns the dimensionality of the coordinate space of a geometric entity. The default value is 3.

```
*)
FUNCTION coordinate_space(arg: geometry): INTEGER;
  LOCAL
    dim : INTEGER;
  END_LOCAL;
  IF NOT EXISTS(arg) THEN
    dim := 3;
  ELSE
  (* code to determine dimensionality of argument type GEOMETRY.
     For point or direction coordinate space is directly deducible.
     For all other geometric entities, dimensionality is the same
     as any point or direction in the attribute set.
  *)
  END_IF;
  RETURN(dim);
END_FUNCTION;
(*
```

4.3.2.11 CO-PLANAR

This function returns TRUE if its arguments are co-planar.

```
*)
FUNCTION co_planar(arg1,arg2: GENERIC): LOGICAL;
```

```
  LOCAL
    result : LOGICAL := FALSE;
  END_LOCAL;
  IF (NOT planar(arg1) OR NOT planar(arg2)) THEN
    RETURN(result);
  END_IF;
      (* pseudo code for checking co-planarity;
         if co-planar RESULT := TRUE *)
  RETURN(result);
END_FUNCTION;
(*
```

4.3.2.12 CYLINDRICAL

The function IS_CYLINDRICAL returns TRUE if its argument is a cylindrical shape.

```
*)
FUNCTION is_cylindrical(arg: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

4.3.2.13 DIMENSIONALITY

The number of parameters required to specify the geometric form of a topological or geometric entity. Edges (curves), Faces (surfaces) and shells (volumes) have dimensionality of 1, 2 and 3 respectively. By convention a Vertex (point) has dimensionality of 0.

```
*)
FUNCTION dimensionality(arg: GENERIC): INTEGER;
END_FUNCTION;
(*
```

returns the dimensionality of its argument.

4.3.2.14 DIRECTION

Both topological and geometric entities have defined orientations. For example an edge is oriented from the first to the second vertex and a surface orientation is defined by the surface normal. The function Direction returns the "direction" of the positive orientation of an entity in whatever units are appropriate to the entity. If the entity is NULL then any value is returned. That is,

```
DIRECTION_FUN(NULL) = DIRECTION_FUN(ANY_ENTITY)
```

is always TRUE.

```
*)
FUNCTION direction_fun(arg: GENERIC): direction;
END_FUNCTION;
(*
```

4.3.2.15 DISJOINT

```
*)
FUNCTION disjoint(arg_1,arg_2: GENERIC): LOGICAL;
  LOCAL
    result : LOGICAL := FALSE;
  END_LOCAL;
  IF (distance(arg_1, arg_2) <> 0.0) THEN
    result := TRUE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

returns true if the two arguments are disjoint.

4.3.2.16 DISTANCE

The function DISTANCE returns the minimum distance between its input arguments. A negative value is returned if the concept of distance does not apply to the argument pair. A zero value is returned if one or both arguments are NULL and the concept of distance would apply if they were not NULL.

```
*)
FUNCTION distance(arg_1,arg_2: GENERIC): REAL;
END_FUNCTION;
(*
```

4.3.2.17 DOES INTERSECT

```
*)
FUNCTION does_intersect(arg1,arg2: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

returns TRUE if the arguments (geometrically) intersect.

4.3.2.18 DOMAIN

The *domain* of a shape (a topological or geometric or other description of some shape) is the mathematical point set explicitly or implicitly represented by the shape. Except for entities with zero dimensionality (where the point set consists of a single point), the point set necessarily contains an infinite number of points. The entity **mathematical point set** is not intended to be implemented in any system based on the Information Model because of its infinite nature, but is included as an essential mathematical concept.

```
*)
ENTITY mathematical_point_set;
 -- points : SET OF point;
END_ENTITY;
(*
```

This function is used to express the concept of "the space (region) over which an entity exists (is defined)". As such, it is unlikely ever to be coded algorithmically.

```
*)
FUNCTION domain(arg: GENERIC): mathematical_point_set;
END_FUNCTION;
(*
```

It may be used, for example, to say that two differently defined geometric curves represent the same point set by

domain(curve1) = domain(curve2)

or different point sets by

domain(curve1) <> domain(curve2)

4.3.2.19 EMBEDDED

This function, given two input arguments that have valid DOMAINs, returns TRUE if the domain of the first argument is "embedded" in the domain of the second argument.

```
*)
FUNCTION embedded(arg1,arg2: GENERIC): LOGICAL;
  LOCAL
    result : LOGICAL := FALSE;
  END_LOCAL;
  IF (intersect(domain(arg1), domain(arg2)) = domain(arg1)) THEN
      result := TRUE;
  END_IF;
  RETURN(result)
END_FUNCTION;
(*
```

4.3.2.20 EMPTY SET

This function returns TRUE if its argument is an empty set.

```
*)
FUNCTION empty_set(arg : generic_set) : LOGICAL;
  LOCAL
    result : LOGICAL := FALSE;
  END_LOCAL;
  IF SIZEOF(arg) <= 0 THEN
    result := TRUE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.3.2.21 EXTENT

The measure of the content of the region occupied by an entity (i.e its **domain**) measured in units appropriate to the dimensionality of the entity, thus length, area and volume for dimensionalities of 1, 2 and 3. Where necessary the symbol Ξ will be used to denote extent.

The **extent** function is a generic function for determining arc-length, surface area or volume. A negative value is returned if its argument is an entity to which the concept of extent does not apply.

```
*)
FUNCTION extent(arg: GENERIC): REAL;
END_FUNCTION;
(*
```

4.3.2.22 SUM

This function returns the sum of the aggregate of input NUMBERs.

```
*)
FUNCTION sum(arg: select_aggregate_of_number): NUMBER;
  LOCAL
    result : NUMBER := 0;
  END_LOCAL;
  REPEAT i := 1 TO SIZEOF(arg);
    result := result + arg[i];
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.3.2.23 GENUS

The function **genus** returns the genus of its argument. Roughly speaking, the genus of an object is the number of holes in the object. A negative integer is returned if the argument has no genus.

```
*)
FUNCTION genus(arg: GENERIC): INTEGER;
END_FUNCTION;
(*
```

4.3.2.24 INSIDE

```
*)
FUNCTION inside(arg1,arg2,arg3: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

returns TRUE if **arg2** is geometrically inside **arg1** with respect to **arg3** which has dimensionality one greater than **arg1** and **arg2**.

4.3.2.25 INSTANTIATION

The function instantiation may be used to define the cardinality between instances of two entity types in a STEP implementation. The function returns the allowable number of instances of its second argument for one instance of its first argument.

```
*)
FUNCTION instantiation(arg1,arg2: GENERIC): INTEGER;
END_FUNCTION;
(*
```

As an example of its use, consider the following pair of statements (which would be part of a set of WHERE clauses):

```
  instantiation(b, x) = 3;
  instantiation(x, b) >= 0;
```

The first line states that an for each instantiation of an entity of type B there must be at least three instantiations of entity type X. The second line states that for each instantiation of an entity of type X there may be zero or more instatiations of entity type B. Taken together the effect is that B depends on X while X is independent of B.

4.3.2.26 INT

```
*)
FUNCTION int(arg: REAL): INTEGER;
END_FUNCTION;
(*
```

returns the integer part of its argument.

4.3.2.27 INTERSECT

The function intersect returns the domain of the geometric intersection of its two input arguments. If there is no intersection then the result is NULL.

```
*)
FUNCTION intersect(arg1,arg2: GENERIC): mathematical_point_set;
END_FUNCTION;
(*
```

4.3.2.28 MANIFOLD

Let m and n be non-negative integers and X be a subset of R^m. Let X be given the relative topology inherited from R^m. Then X is said to be a topological $n-$manifold provided each point of X has an open neighbourhood that is homeomorphic to the open unit ball of R^n.

Essentially this means that an entity with geometric or topological connotations is manifold if it is arcwise connected and neither self-intersects or overlaps itself.

The function manifold returns TRUE if its argument is $d-$manifold where d is the dimensionality of the entity.

```
*)
FUNCTION manifold(arg: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

4.3.2.29 OCCURS

```
 *)
FUNCTION occurs(element: GENERIC; aggregate: select_aggregate)
                                                          : INTEGER;
  LOCAL
    temp : INTEGER := 0;
  END_LOCAL;
     REPEAT i := 1 TO SIZEOF(aggregate);
       IF element = aggregate[i] THEN
         temp := temp + 1;
       END_IF;
     END_REPEAT;
     RETURN (temp);
END_FUNCTION;
(*
```

returns the number of times a particular instance of an entity occurs in a aggregate of the entity type.

4.3.2.30 OPEN

An entity of dimensionality D is topologically open if it is not topologically closed.

```
*)
FUNCTION open(arg: GENERIC): LOGICAL;
      RETURN(NOT closed(arg));
END_FUNCTION;
(*
```

4.3.2.31 OVERLAP

An entity of dimensionality D overlaps if at every point of the domain of the entity an open ball of diameter ϵ, where $\epsilon \rightarrow 0$, in R^{D+1} can be drawn that is partitioned into exactly two regions, and two or more points in the domain correspond to the same point in R^3.

```
*)
FUNCTION overlap(arg: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

4.3.2.32 PARAMETRIC CURVE EVALUATOR

```
*)
FUNCTION parametric_curve_evaluator(arg: curve; param_value: REAL)
```

```
                                        : coordinate_triple;
END_FUNCTION;
(*
```

returns the (X, Y, Z) values corresponding to the point on its first argument at the parameter value equal to the second argument.

4.3.2.33 PARAMETRIC SURFACE EVALUATOR

```
*)
FUNCTION parametric_surface_evaluator(arg: surface;
            param_values: coordinate_pair): coordinate_triple;
END_FUNCTION;
(*
```

returns the (X, Y, Z) values corresponding to the point on its first argument at the parameter values equal to the second argument.

4.3.2.34 PERPENDICULAR

This function returns the vector that is perpendicular to its input argument.

```
*)
FUNCTION perpendicular(arg: GENERIC): vector;
END_FUNCTION;
(*
```

4.3.2.35 PLANAR

```
*)
FUNCTION planar(arg: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

This function returns TRUE if the argument is an entity that lies in a single geometric plane.

4.3.2.36 SELF INTERSECT

An entity self-intersects if its domain is neither manifold nor overlapping. The function returns TRUE if its argument is a self intersecting entity.

```
*)
FUNCTION self_intersect(arg: GENERIC): LOGICAL;
  RETURN (NOT (manifold(arg) OR overlap(arg));
END_FUNCTION;
(*
```

4.3.2.37 SYMMETRIC

This function returns TRUE if its argument, or the members of the argument if it is an aggregate, are symmetric about a point or a curve or a surface.

```
*)
FUNCTION is_symmetric(arg: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

4.3.2.38 SYMMETRIC GEOMETRY

This function returns the Curve or Surface that is the "center of symmetry" of its argument.

```
*)
FUNCTION symmetric_geometry(arg: GENERIC): curve_or_surface;
  LOCAL
    result : curve_or_surface;
  END_LOCAL;
  result := NULL;
  IF is_symmetric(arg) THEN
    BEGIN
    (* pseudo code for determining the symmetric Curve or Surface
       and putting this into RESULT as RESULT := CURVE_OR_SURFACE *)
    END;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.3.2.39 Topological Functions

Several functions are defined here for calculating the lower level topological entities used by higher level entities.

4.3.2.39.1 CREATE VERTEX

This function creates a new instance of a VERTEX or returns an existing VERTEX if one exists with the same parameters.

```
*)
FUNCTION create_vertex(pnt: point): vertex;
  LOCAL
    new_v   : vertex;
    another : LOGICAL := FALSE;
  END_LOCAL;
  REPEAT FOR EACH vertex IN MODEL WHILE NOT another;
    IF vertex.vertex_point = pnt THEN
      (* Already a vertex at this point *)
      another := true;
```

```
      new_v := vertex;
    END_IF;
  END_REPEAT;
  IF NOT another THEN
  (* pseudo code for creating a vertex
    NEW_V.VERTEX_POINT := PNT;
    end pseudo code *)
  END_IF;
  RETURN(new_v);
END_FUNCTION;
(*
```

4.3.2.39.2 CREATE STRAIGHT EDGE

This function creates a new instance of an EDGE between two points or returns an existing EDGE if one exists between the same points.

```
*)
FUNCTION create_straight_edge(sp, ep: point): edge;
  LOCAL
    new_e     : edge;
    vtx_list : list_of_vertex;
    vtx_set   : set_of_vertex;
    another   : LOGICAL := FALSE;
  END_LOCAL;
  IF distance(sp, ep) = 0 THEN
    RETURN(NULL);
  END_IF;
  (* create vertices for the points *)
  vtx_list[1] := create_vertex(sp);
  vtx_list[2] := create_vertex(ep);
  vtx_set := vtx_set + vtx_list;
  (* check if another edge with same geometry *)
  REPEAT FOR EACH edge IN MODEL WHILE NOT another;
    IF (vtx_set = set_edge_vertices(edge)) AND
       (extent(edge) = distance(sp, ep)) THEN
      BEGIN
        another := TRUE;
        new_e := edge;
      END;
    END_IF;
  END_REPEAT;
  IF NOT another THEN
    BEGIN
      (* psuedo code for creating a new edge
      NEW_E.EDGE_START := VTX_LIST[1];
      NEW_E.EDGE_END := VTX_LIST[2];
         end pseudo code *)
    END;
```

```
  END_IF;
  RETURN(new_e);
END_FUNCTION;
(*
```

4.3.2.39.3 LIST EDGE VERTICES

Given an EDGE the function returns the list of VERTICES used by the Edge.

```
*)
FUNCTION list_edge_vertices(e: edge): list_of_vertex;
  LOCAL
    vertices : list_of_vertex := NULL;
  END_LOCAL;
  vertices := vertices + e.edge_start + e.edge_end;
  RETURN(vertices);
END_FUNCTION;
(*
```

4.3.2.39.4 SET EDGE VERTICES

Given an EDGE the function returns the set of VERTICES used by the edge.

```
*)
FUNCTION set_edge_vertices(e: edge): set_of_vertex;
  LOCAL
    vertices : set_of_vertex := NULL;
  END_LOCAL;
  vertices := vertices + list_edge_vertices(e);
  RETURN(vertices);
END_FUNCTION;
(*
```

4.3.2.39.5 LIST LOOP EDGE LOGICALS

Given a LOOP the function returns the list of EDGE LOGICAL pairs in the Loop.

```
*)
FUNCTION list_loop_edge_logicals(l: loop)
                  : list_of_edge_logical_structure;
  LOCAL
    el : list_of_edge_logical_structure := NULL;
  END_LOCAL;
  CASE TYPEOF(l) OF
    edge_loop : BEGIN
                  REPEAT i := 1 TO SIZEOF(l.loop_edges);
                    el := el + l.loop_edges[i];
                  END_REPEAT;
                END;
```

```
    poly_loop : el := create_polyloop_edge_logical(l);
  END_CASE;
  RETURN(el);
END_FUNCTION;
(*
```

4.3.2.39.6 CREATE POLYLOOP EDGE LOGICAL

Given a POLY-LOOP the function returns the list of edge/logical pairs, if necessary creating instances of vertices, edges and edge/logical pairs that would be necessary to represent the poly-loop as an edge-loop.

```
*)
FUNCTION create_polyloop_edge_logical(l: poly_loop)
                    : list_of_edge_logical_structure;
  LOCAL
   el_list     : list_of_edge_logical_structure;
   new_el      : edge_logical_structure;
   new_e       : edge;
   ev_list     : list_of_vertex;
   it          : INTEGER;
   another     : LOGICAL;
   tflag       : LOGICAL;
  END_LOCAL;
  (* create edge/logical instances for the points of the polyloop *)
  REPEAT i := 1 TO SIZEOF(l.polygon);
    it := i+1;
    IF i = SIZEOF(l.polygon) THEN
      it := 1;
    END_IF;
    (* create an edge *)
    new_e := create_straight_edge(l.polygon[i], l.polygon[it]);
    ev_list := list_edge_vertices(new_e);
    (* set TFLAG according to required edge direction in polyloop *)
    tflag := l.polygon[1] = ev_list[1].edge_start.vertex_point;
    (* check if there is another instance of this potential new
       edge/logical *)
    another := FALSE;
    REPEAT FOR EACH edge_logical_structure IN MODEL WHILE
                                                  NOT another;
      IF edge_logical_structure.edge_element = new_e AND
            edge_logical_structure.flag = tflag THEN
        BEGIN
          (* already have this edge/logical *)
          another := TRUE;
          new_el := edge_logical_structure;
        END;
      END_IF;
    END_REPEAT;
```

```
    IF NOT another THEN
      BEGIN
        (* pseudo code for creating a new edge/logical instance
        NEW_EL.EDGE_ELEMENT := NEW_E;
        NEW_EL.FLAG := TFLAG;
           end of pseudo code *)
      END;
    END_IF;
    (* and add edge/logical structure to output list *)
    el_list := el_list + new_el;
  END_REPEAT;
  RETURN(el_list);
END_FUNCTION;
(*
```

4.3.2.39.7 LIST LOOP EDGES

Given a LOOP the function returns the list of EDGES in the Loop.

```
*)
FUNCTION list_loop_edges(l: loop): list_of_edge;
  LOCAL
    edges : list_of_edge;
  END_LOCAL;
  CASE TYPEOF(l) OF
    edge_loop : BEGIN
        REPEAT FOR EACH edge_logical_structure IN l.loop_edges;
          edges := edges + edge_logical_structure.edge_element;
        END_REPEAT;
                END;
    poly_loop : BEGIN
        REPEAT FOR EACH edge_logical_structure in
                        create_polyloop_edge_logical(l);
          edges := edges + edge_logical_structure.edge_element;
        END_REPEAT;
                END;
  END_CASE;
  RETURN(edges);
END_FUNCTION;
(*
```

4.3.2.39.8 SET LOOP EDGES

Given a LOOP the function returns the set of EDGES in the Loop.

```
*)
FUNCTION set_loop_edges(l: loop): set_of_edge;
  LOCAL
    edges : set_of_edge;
```

```
  END_LOCAL;
  edges := edges + list_loop_edges(l);
  RETURN(edges);
END_FUNCTION;
(*
```

4.3.2.39.9 SET LOOP VERTICES

Given a LOOP the function returns the set of VERTICES in the Loop.

```
*)
FUNCTION set_loop_vertices(l: loop): set_of_vertex;
  LOCAL
    vertices : set_of_vertex;
    edges    : set_of_edge := set_loop_edges(l);
  END_LOCAL;
  CASE TYPEOF(l) OF
    edge_loop   : BEGIN
        REPEAT FOR EACH edge IN edges;
          vertices := vertices + set_edge_vertices(edge);
        END_REPEAT;
                  END;
    poly_loop   : BEGIN
        REPEAT FOR EACH edge IN edges;
          vertices := vertices + set_edge_vertices(edge);
        END_REPEAT;
                  END;
    vertex_loop : vertices := vertices + l.loop_vertex;
  END_CASE;
  RETURN(vertices);
END_FUNCTION;
(*
```

4.3.2.39.10 LIST FACE LOOPS

Given a FACE or a SUBFACE the function returns the list of LOOPS in the Face or Subface.

```
*)
FUNCTION list_face_loops(f: select_face_or_subface)
                                          : list_of_loop;
  LOCAL
    loops : list_of_loop;
  END_LOCAL;
  REPEAT FOR EACH loop_logical_structure IN f.bounds;
    loops := loops + loop_logical_structure.loop_element;
  END_REPEAT;
  RETURN(loops);
END_FUNCTION;
(*
```

4.3.2.39.11 SET FACE LOOPS

Given a FACE or a SUBFACE the function returns the set of LOOPS in the Face or Subface.

```
*)
FUNCTION set_face_loops(f: select_face_or_subface): set_of_loop;
  LOCAL
    loops : set_of_loop;
  END_LOCAL;
  loops := loops + list_face_loops(f);
  RETURN(loops);
END_FUNCTION;
(*
```

4.3.2.39.12 LIST FACE EDGES

Given a FACE or SUBFACE the function returns the list of EDGES in the Face or Subface.

```
*)
FUNCTION list_face_edges(f: select_face_or_subface)
                                                : list_of_edge;
  LOCAL
    edges : list_of_edge;
  END_LOCAL;
  REPEAT FOR EACH loop IN list_face_loops(f);
    edges := edges + list_loop_edges(loop);
  END_REPEAT;
  RETURN(edges);
END_FUNCTION;
(*
```

4.3.2.39.13 SET FACE EDGES

Given a FACE or SUBFACE the function returns the set of EDGES in the Face or Subface.

```
*)
FUNCTION set_face_edges(f: select_face_or_subface): set_of_edge;
  LOCAL
    edges : set_of_edge;
  END_LOCAL;
  edges := edges + list_face_edges(f);
  RETURN(edges);
END_FUNCTION;
(*
```

4.3.2.39.14 SET FACE VERTICES

Given a FACE or SUBFACE the function returns the set of VERTICES in the Face or Subface.

```
*)
FUNCTION set_face_vertices(f: select_face_or_subface)
                                                    : set_of_vertex;
  LOCAL
    vertices : set_of_vertex;
  END_LOCAL;
  REPEAT FOR EACH edge IN set_face_edges(f);
    vertices := vertices + set_edge_vertices(edge);
  END_REPEAT;
  RETURN(vertices);
END_FUNCTION;
(*
```

4.3.2.39.15 LIST SHELL FACES

Given a SHELL the function returns the list of FACES in the Shell.

```
*)
FUNCTION list_shell_faces(s: shell): list_of_face;
  LOCAL
    faces : list_of_face;
  END_LOCAL;
  CASE TYPEOF(s) OF
    closed_shell : BEGIN
          REPEAT FOR EACH face_logical_structure IN
                                                  s.cshell_boundary;
            faces := faces + face_logical_structure.face_element;
          END_REPEAT;
                  END;
    open_shell   : BEGIN
        REPEAT FOR EACH face_logical_structure IN s.shell_boundary;
          faces := faces + face_logical_structure.face_element;
        END_REPEAT;
                  END;
  END_CASE;
  RETURN(faces);
END_FUNCTION;
(*
```

4.3.2.39.16 SET SHELL FACES

Given a SHELL the function returns the set of FACES in the Shell.

```
*)
FUNCTION set_shell_faces(s: shell): set_of_face;
  LOCAL
    faces : set_of_face;
  END_LOCAL;
  faces := faces + list_shell_faces(s);
```

```
  RETURN(faces);
END_FUNCTION;
(*
```

4.3.2.39.17 LIST SHELL LOOPS

Given a SHELL the function returns the list of LOOPS in the Shell.

```
*)
FUNCTION list_shell_loops(s: shell): list_of_loop;
  LOCAL
    loops : list_of_loop;
  END_LOCAL;
  CASE TYPEOF(s) OF
    vertex_shell : loops := loops + s.vertex_shell_boundary;
    wire_shell   : BEGIN
        REPEAT FOR EACH edge_loop IN s.wire_shell_boundary;
          loops := loops + edge_loop;
        END_REPEAT;
    open_shell   : BEGIN
        FOR EACH face IN set_shell_faces(s);
          loops := loops + list_face_loops(face);
        END_REPEAT;
                   END;
    closed_shell : BEGIN
        FOR EACH face IN set_shell_faces(s);
          loops := loops + list_face_loops(face);
        END_REPEAT;
                   END;
  END_CASE;
  RETURN(loops);
END_FUNCTION;
(*
```

4.3.2.39.18 SET SHELL LOOPS

Given a SHELL the function returns the set of LOOPS in the Shell.

```
*)
FUNCTION set_shell_loops(s: shell): set_of_loop;
  LOCAL
    loops : set_of_loop;
  END_LOCAL;
  loops := loops + list_shell_loops(s);
  RETURN(loops);
END_FUNCTION;
(*
```

4.3.2.39.19 LIST SHELL EDGES

Given a SHELL the function returns the list of EDGES in the Shell.

```
*)
FUNCTION list_shell_edges(s: shell): list_of_edge;
  LOCAL
    edges : list_of_edge;
  END_LOCAL;
  REPEAT FOR EACH loop IN list_shell_loops(s);
    edges := edges + list_loop_edges(loop);
  END_REPEAT;
  RETURN(edges);
END_FUNCTION;
(*
```

4.3.2.39.20 SET SHELL EDGES

Given a SHELL the function returns the set of EDGES in the Shell.

```
*)
FUNCTION set_shell_edges(s: shell): set_of_edge;
  LOCAL
    edges : set_of_edge;
  END_LOCAL;
  edges := edges + list_shell_edges(s);
  RETURN(edges);
END_FUNCTION;
(*
```

4.3.2.39.21 SET SHELL VERTICES

Given a SHELL the function returns the set of VERTICES in the Shell.

```
*)
FUNCTION set_shell_vertices(s: shell): set_of_vertex;
  LOCAL
    vertices : set_of_vertex;
  END_LOCAL;
  REPEAT FOR EACH edge IN set_shell_edges(s);
    vertices := vertices + set_edge_vertices(edge);
  END_REPEAT;
  RETURN(vertices);
END_FUNCTION;
(*
```

4.3.2.39.22 MIXED LOOP TYPE SET

Given a set of LOOPs, the function returns TRUE if the set includes both poly-loops and other types (edge and vertex) of loops.

```
*)
FUNCTION mixed_loop_type_set(l: set_of_loop): LOGICAL;
  LOCAL
    result : LOGICAL := FALSE;
  END_LOCAL;
  IF TYPEOF(l[1]) = poly_loop THEN
    BEGIN
      REPEAT FOR EACH loop IN l;
        IF TYPEOF(loop) <> poly_loop THEN
          result := TRUE;
        END_IF;
      END_REPEAT;
    END;
  ELSE
    BEGIN
      REPEAT FOR EACH loop IN l;
        IF TYPEOF(loop) = poly_loop THEN
          result := TRUE;
        END_IF;
      END_REPEAT;
    END;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.4 Miscellaneous Resources

4.4.1 Introduction

This Section contains a collection of miscellaneous resources that appear to have general applicability. These have been extracted from several Topical Models.

4.4.1.1 ALTERNATE REPRESENTATION

This entity provides a mechanism for identifying a set of "representations" all of which, from one viewpoint or another, are representative of a single "thing". Some examples:

1. An electrical circuit that is described as a Printed Wiring Board and a Logical or Simulation Model.
2. A Brep description of a piecepart and a finite element mesh of an idealization of the part.

Essentially, two alternates are approximately the same, within some context (see the **equivalent representation**).

```
*)
ENTITY alternate_representation;
  master_representation : GENERIC;
  alternates            : LIST [2 : #] OF GENERIC;
WHERE
  master_representation IN alternates;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

master_representation: The representation preferred by the originator among the listed alternatives.

alternates: The list of entities that are different representations of the same product data.

PROPOSITIONS:

1. There must be at least two **alternates**.
2. The **master representation** entity is included in the list of **alternates** entities.

4.4.1.2 DATE

The date (AD) in the western calender. This is the ISO definition of a date.

```
*)
ENTITY date;
  year  : INTEGER;
  month : INTEGER;
  day   : INTEGER;
WHERE
  year > 0;
```

```
  {1 <= month <= 12};
  {1 <= day <= 31};
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

year: The year.

month: The month of the year.

day: The day of the month.

PROPOSITIONS:

1. There are 12 months in a year.
2. There are no months with more than 31 days.

4.4.1.3 DATE TIME

The calender date and time of day.

```
*)
ENTITY date_time;
  d  : date;
  t  : time;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

d: The date.

t: The time.

4.4.1.4 EQUIVALENT REPRESENTATION

This entity provides a mechanism for identifying a set of "representations" all of which are representative of a single "thing". Some examples:

1. A planar curve whose every point is equidistant from a single point that is both represented as a Circle and a Composite B-Spline Curve.
2. A representation of a piece part that is described as a CSG Model, a Brep Model and a blueprint.
3. Two dates, one defined via the Western Calender and the other in the Japanese Calender, both representing the same moment in time.

Essentially, two **equivalents** are meant to be identically equal (see the **alternate representation** entity).

```
*)
ENTITY equivalent_representation;
  master_representation : GENERIC;
  equivalents           : LIST [2 : #] OF GENERIC;
WHERE
  master_representation IN equivalents;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

master_representation: The representation preferred by the originator among the listed equivalents.

equivalents: The list of entities that are different representations of the same product data.

PROPOSITIONS:

1. There must be at least two equivalents.
2. The Master entity is included in the list of Equivalent entities.

4.4.1.5 PERSON AND ORGANIZATION

The name of a person and/or an organization structure

```
*)
ENTITY person_and_organization;
  person      : OPTIONAL person_name;
  company     : OPTIONAL STRING;
  department  : OPTIONAL STRING;
  section     : OPTIONAL STRING;
  project     : OPTIONAL STRING;
WHERE
  ((person <> NULL) OR
   (company <> NULL) OR
   (department <> NULL) OR
   (section <> NULL) OR
   (project <> NULL));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

person: The name of a person.

company: The name of a Company.

department: The name of a Department.

section: The name of a Section.

project: The name of a Project.

PROPOSITIONS:

1. At least one attribute must be present.

4.4.1.6 PERSON NAME

The name of a person.

```
*)
ENTITY person_name;
  name_text : STRING;
  title     : OPTIONAL STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name_text: The name of a person.

title: Optionally, the title of the person (e.g the person's position in an organisation rather than an honorific (Mrs. or Dr.)).

PROPOSITIONS:

4.4.1.7 TIME

The time defined by a 24 hour clock with zero time being midnight. This is the ISO definition for time.

```
*)
ENTITY time;
  hour   : INTEGER;
  minute : INTEGER;
  second : INTEGER;
  msec   : OPTIONAL REAL;
WHERE
  (0 <= hour < 24);
  (0 <= minute < 60);
  (0 <= second < 60);
  (0.0 <= msec < 1000.0);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

hour: The hour of day on a 24 hour clock.

minute: The minute of an hour.

second: The second of a minute.

msec: The milli-second.

PROPOSITIONS:

1. Hours must be greater than or equal to zero and less than 24.

2. Minutes must be greater than or equal to zero and less than 60.

3. Seconds must be greater than or equal to zero and less than 60.

4. Milli-seconds must be less than 1000 and greater than or equal to zero.

4.4.1.8 APPROVAL HISTORY

This entity captures the historical information of approvals over time.

```
*)
ENTITY approval_history;
  history : LIST [1 : #] OF approval;
WHERE
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

history: The list of approval in date order.

```
*)
RULE approval_history_rule FOR (approval_history);
  LOCAL
    i : INTEGER;
  END_LOCAL;
  REPEAT i := 2 TO SIZEOF(history);
    IF (history[i].approval_date < history[i-1].approval_date)
    THEN VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. The approvals are in increasing date order.

4.4.1.9 APPROVAL

This entity is intended to capture an approval status. The attribute release (= TRUE) indicates that the responsible organization has designated that the "thing" being approved is sufficiently complete to allow for its subsequent use by another person or organization.

```
*)
ENTITY approval;
  approval_date     : date_time;
  approving_org     : person_and_organization;
  approval_purpose  : STRING;
  release           : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

approval_date: The date and time when approval was given.

approving_org: The name of the approving organization (company, department, section, person, and/or project).

approval_purpose: A description of the purpose of the approval.

release: TRUE if it is deemed that "thing" being approved may be used by another person or organization, otherwise it is FALSE.

4.4.1.10 SECURITY CLASSIFICATION HISTORY

This entity allows for capturing the historical information for changes in the security classification assigned over time.

```
*)
ENTITY security_classification_history;
  history : LIST [1 : #] OF security_classification;
WHERE
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

history: The time history of security classifications.

```
*)
RULE security_classification_history_rule FOR
                    (security_classification_history);
  LOCAL
    i : INTEGER;
  END_LOCAL;
  REPEAT i := 2 TO SIZEOF(history);
    IF (history[i].classification_date <
        history[i-1].classification_date) THEN
      VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. The security classifications must be in increasing date order.

4.4.1.11 SECURITY CLASSIFICATION

This entity captures a particular security classification state.

```
*)
ENTITY security_classification;
  security_level                  : security_class_level;
  classification_date             : date;
  declassification_date           : OPTIONAL date;
  classification_control_officer  : person_and_organization;
WHERE
  declassification_date > classification_date;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

security_level: The assigned level of security.

classification_date: The effective date for the classification.

declassification_date: The date on which the classification is planned to be changed.

classification_control_officer: The name of the controlling officer (company, department, section, person, and/or project) for the classification.

4.4.1.12 UNDEFINED

This entity represents an undefined piece of information. Its main purpose is to enable "incomplete" entities to be defined.

```
*)
ENTITY UNDEFINED;
END_ENTITY;
(*
```

4.4.1.13 USER DEFINED ENTITY

This is a first attempt at specifying an entity that can be used for conveying user defined information. The syntax is defined, but not the semantics.

```
*)
ENTITY user_defined_entity;
  creator       : person_and_organization;
  creation_date : date;
  type_id       : STRING;
  references    : OPTIONAL LIST [0:#] OF GENERIC;
  integers      : OPTIONAL LIST [0:#] OF INTEGER;
  reals         : OPTIONAL LIST [0:#] OF REAL;
  logicals      : OPTIONAL LIST [0:#] OF LOGICAL;
  strings       : OPTIONAL LIST [0:#] OF STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

creator: The person/organization which created the definition of the semantics of the entity.

creation_date: The date when the semantics of the entity was defined.

type_id: An indicator of the the type of user defined entity. This is used to distinguish between user defined entities with the same creator and creation date that have different semantics.

references: A list of entity references.

integers: A list of integer numbers.

reals: A list of real numbers.

logicals: A list of logical values.

strings: A list of strings.

4.4.2 Units of Measure

A *system of units* is centered on a small number of *base units*. These relate to the fundamental standards of length, mass and time, together with a few others to extend the system to a wider range of physical measurements. There are also two geometrical units which belong to a class known as *supplementary units*. Table 1 is a listing of the fundamental units (base and supplementary) and the symbols used here to represent them.

QUANTITY	UNITS	QUANTITY	UNITS
length	L	mass	M
time	T	electric current	i
thermodynamic temperature	θ	amount of substance	m
luminous intensity	λ		
plane angle	α	solid angle	σ

Table 1: Symbology for the Fundamental Units

These few base units can be combined to form a large number of *derived units*. For example, units of area, velocity and acceleration are formed from units of length and time. Table 2 gives the dimensional values for some common measures in terms of the fundamental units.

4.4.2.1 UNITS

The units entity defines the base (or fundamental) units of measure. All other units can be derived from this set.

```
*)
ENTITY units;
  base_length_unit                : scaled_length_unit;
  base_mass_unit                  : scaled_mass_unit;
  base_time_unit                  : scaled_time_unit;
  base_electric_current_unit      : scaled_current_unit;
  base_temperature_unit           : scaled_temperature_unit;
  base_amount_of_substance_unit   : scaled_amount_unit;
  base_luminous_intensity_unit    : scaled_luminous_intensity_unit;
```

QUANTITY	UNITS	QUANTITY	UNITS
area	L^2	volume	L^3
velocity	LT^{-1}	angular velocity	αT^{-1}
acceleration	LT^{-2}	angular acceleration	αT^{-2}
frequency	T^{-1}	density	$L^{-3}M$
momentum	LMT^{-1}	angular momentum	L^2MT^{-1}
moment of inertia	L^2M	force	LMT^{-2}
pressure, stress	$L^{-1}MT^{-2}$	work, energy	L^2MT^{-2}
power	L^2MT^{-3}		
surface tension	MT^{-2}	dynamic viscosity	$L^{-1}MT^{-1}$
kinematic viscosity	L^2T^{-1}	diffusivity	L^2T^{-1}
quantity of heat	L^2MT^{-2}	thermal conductivity	$LMT^{-3}\theta^{-1}$
heat capacity	$L^2MT^{-2}\theta^{-1}$	specific heat capacity	$L^2T^{-2}\theta^{-1}$
specific latent heat	L^2T^{-2}		
electric charge	iT	electric potential	$i^{-1}L^2MT^{-3}$
electric resistance	$i^{-2}L^2MT^{-3}$	resistivity	$i^{-2}L^3MT^{-3}$
electric conductance	$i^2L^{-2}M^{-1}T^3$	electric capacitance	$i^2L^{-2}M^{-1}T^4$
inductance	$i^{-2}L^2MT^{-2}$	magnetic flux	$i^{-1}L^2MT^{-2}$
magnetic flux density	$i^{-1}MT^{-2}$	magnetomotive force	i
luminous flux	$\lambda\sigma$	illumination	$\lambda\sigma L^{-2}$
radiation activity	T^{-1}	radiation absorbed dose	L^2T^{-2}

Table 2: Dimensions of some Derived Units of Measure

```
  supplementary_plane_angle_unit : scaled_plane_angle_unit;
  supplementary_solid_angle_unit : scaled_solid_angle_unit;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

base_length_unit: The scaled unit of length (e.g meter).

base_mass_unit: The scaled unit of mass (e.g gram).

base_time_unit: The scaled unit of time (e.g fortnight).

base_electric_current_unit: The scaled unit of electric current (e.g ampere).

base_temperature_unit: The scaled unit of temperature (e.g Kelvin).

base_amount_of_substance_unit: The scaled unit of the amount of substance (e.g mole).

base_luminous_intensity_unit: The scaled unit of luminous intensity (e.g candela).

supplementary_plane_angle_unit: The scaled unit of plane angle (e.g degree).

supplementary_solid_angle_unit: The scaled unit of solid angle (e.g steradian).

4.4.2.2 LENGTH UNIT

The units of length.

A *metre* is the length equal to 1650763.73 wavelengths in vacuum of the radiation corresponding to the transition between the levels $2p_{10}$ and $5d_5$ of the krypton-86 atom.

Conversion factors are given in Table 3 with respect to the *metre* and in Table 4 for Imperial units with respect to the *inch*.

UNIT	METRES
angstrom	10^{-10}
micron	10^{-6}
metre	1
inch	2.54×10^{-2}

Table 3: Length Units in terms of Metres

UNIT	INCHES
mil	10^{-3}
inch	1
foot	12
yard	3×12
mile	$1760 \times 3 \times 12$

Table 4: Imperial Length Units in terms of Inches

```
*)
TYPE length_unit = ENUMERATION OF
     (angstrom,
      micron,
      metre,
      mil,
      inch,
      foot,
      yard,
      mile,
      nautical_mile,
      astronomical_unit,
      light_year);
END_TYPE;
(*
```

4.4.2.3 SCALED LENGTH UNIT

A unit of length and a scale factor.

```
*)
ENTITY scaled_length_unit;
  unit         : length_unit;
  scale_factor : OPTIONAL REAL;
WHERE
  scale_factor > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

unit: The length unit.

scale_factor: An optional multiplicative factor applied to the unit.

PROPOSITIONS:

1. If present, the scale factor must be greater than zero.

4.4.2.4 MASS UNIT

The units of mass.
A *kilogram* is the mass of the international prototype of the kilogram, which is in the custody of the Bureau International des Poids et Mesures (BIPM) at sevres near Paris, France.
Conversion factors are given in Table 5 with respect to the *kilogram* and Table 6 provides conversion factors for Imperial units.

UNIT	KILOGRAM
kilogram	1
tonne	10^3
pound	0.45359237

Table 5: Mass Units in terms of Kilograms

UNIT	POUND
ounce	1/16
pound	1

Table 6: Imperial Mass Units in terms of Pounds

```
*)
TYPE mass_unit = ENUMERATION OF
     (kilogram,
      tonne,
      ounce,
      pound,
      slug,
      dram,
      grain,
      carat);
END_TYPE;
(*
```

4.4.2.5 SCALED MASS UNIT

A unit of mass and a scale factor.

```
*)
```

```
ENTITY scaled_mass_unit;
  unit         : mass_unit;
  scale_factor : OPTIONAL REAL;
WHERE
  scale_factor > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

unit: The **mass unit**.

scale_factor: An optional multiplicative factor applied to the **unit**.

PROPOSITIONS:

1. If present, the **scale factor** must be greater than zero.

4.4.2.6 TIME UNIT

The units of time.

A *second* is the duration of 9192631770 periods of the radiation corresponding to the transition between the two hyperfine levels of the ground state of the caesium-133 atom.

Conversion factors are given in Table 7 with respect to the *second*.

UNIT	SECOND
second	1
minute	60
hour	60 × 60
day	24 × 60 × 60
week	7 × 24 × 60 × 60
year	approx 365.25 × 24 × 60 × 60

Table 7: Time Units in terms of Seconds

```
*)
TYPE time_unit = ENUMERATION OF
     (second,
      minute,
      hour,
      day,
      week,
      year);
END_TYPE;
(*
```

4.4.2.7 SCALED TIME UNIT

A unit of time and a scale factor.

```
*)
ENTITY scaled_time_unit;
  unit         : time_unit;
  scale_factor : OPTIONAL REAL;
WHERE
  scale_factor > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

unit: The time unit.

scale_factor: An optional multiplicative factor applied to the **unit.**

PROPOSITIONS:

1. If present, the **scale factor** must be greater than zero.

4.4.2.8 CURRENT UNIT

The units of electric current.

An *ampere* is the constant current which, if maintained in two straight parallel conductors of infinite length of negligible circular cross-section, and placed one metre apart in vacuum, would produce between these conductors a force equal to 2×10^{-7} newton per metre of length.

```
*)
TYPE current_unit = ENUMERATION OF
     (ampere);
END_TYPE;
(*
```

4.4.2.9 SCALED CURRENT UNIT

A unit of current and a scale factor.

```
*)
ENTITY scaled_current_unit;
  unit         : current_unit;
  scale_factor : OPTIONAL REAL;
WHERE
  scale_factor > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

unit: The current unit.

scale_factor: An optional multiplicative factor applied to the **unit.**

PROPOSITIONS:

1. If present, the **scale factor** must be greater than zero.

4.4.2.10 TEMPERATURE UNIT

The units of thermodynamic temperature.

A *kelvin* is the fraction 1/273.16 of the thermodynamic temperature of the triple point of water. The triple point is the point where water, ice and water vapour are in equilibrium.

Conversion factors are given in Table 8 with respect to the *kelvin*.

UNIT	KELVIN
kelvin	1
celsius	+273.15
fahrenheit	32 + (9/5) × 273.15

Table 8: Temperatures in terms of Kelvin

```
*)
TYPE temperature_unit = ENUMERATION OF
     (kelvin,
      celsius,
      fahrenheit,
      rankine);
END_TYPE;
(*
```

4.4.2.11 SCALED TEMPERATURE UNIT

A unit of temperature and a scale factor.

```
*)
ENTITY scaled_temperature_unit;
  unit         : temperature_unit;
  scale_factor : OPTIONAL REAL;
WHERE
  scale_factor > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

unit: The temperature unit.

scale_factor: An optional multiplicative factor applied to the unit.

PROPOSITIONS:

1. If present, the scale factor must be greater than zero.

4.4.2.12 AMOUNT UNIT

The units of amount of substance.

A *mole* is the amount of substance of a system which contains as many elementary entities as there are atoms in 0.012 kilogram of carbon-12.

```
*)
TYPE amount_unit = ENUMERATION OF
     (mole);
END_TYPE;
(*
```

4.4.2.13 SCALED AMOUNT UNIT

A unit of amount of substance and a scale factor.

```
*)
ENTITY scaled_amount_unit;
  unit         : amount_unit;
  scale_factor : OPTIONAL REAL;
WHERE
  scale_factor > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

unit: The **amount unit.**

scale_factor: An optional multiplicative factor applied to the **unit.**

PROPOSITIONS:

1. If present, the scale factor must be greater than zero.

4.4.2.14 LUMINOUS INTENSITY UNIT

The units of luminous intensity.
A *candela* is the luminous intensity, in the perpendicular direction, of a surface of 1/600000 square metre of a black body at the temperature of freezing platinum under a pressure of 101325 newtons per square metre.

```
*)
TYPE luminous_intensity_unit = ENUMERATION OF
     (candela);
END_TYPE;
(*
```

4.4.2.15 SCALED LUMINOUS INTENSITY UNIT

A unit of luminous intensity and a scale factor.

```
*)
ENTITY scaled_luminous_intensity_unit;
  unit         : luminous_intensity_unit;
  scale_factor : OPTIONAL REAL;
```

```
WHERE
  scale_factor > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

unit: The luminous intensity unit.

scale_factor: An optional multiplicative factor applied to the **unit**.

PROPOSITIONS:

1. If present, the **scale factor** must be greater than zero.

4.4.2.16 PLANE ANGLE UNIT

The units of plane angle.
A *radian* is the plane angle between two radii of a circle which cut off on the circumference an arc equal in length to the radius.
A complete circle subtends a plane angle at its center of 2π radians.
Conversion factors are given in Table 9 with respect to degrees.

UNIT	DEGREE
second	$1/60 \times 1/60$
minute	$1/60$
degree	1
radian	$180/\pi$

Table 9: Plane Angle Units in terms of Degrees

```
*)
TYPE plane_angle_unit = ENUMERATION OF
      (second,
       minute,
       degree,
       grad,
       radian);
END_TYPE;
(*
```

4.4.2.17 SCALED PLANE ANGLE UNIT

A unit of plane angle and a scale factor.

```
*)
ENTITY scaled_plane_angle_unit;
  unit         : plane_angle_unit;
  scale_factor : OPTIONAL REAL;
```

```
WHERE
  scale_factor > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

unit: The plane angle unit.

scale_factor: An optional multiplicative factor applied to the **unit.**

PROPOSITIONS:

1. If present, the **scale factor** must be greater than zero.

4.4.2.18 SOLID ANGLE UNIT

The units of solid angle.
A *steradian* is the solid angle which having its vertex at the center of a sphere, cuts off an area of the surface of the sphere equal to that of a square having sides of length equal to the radius of the sphere.
A complete sphere subtends a solid angle at its center equal to 4π steradians.

```
*)
TYPE solid_angle_unit = ENUMERATION OF
     (steradian);
END_TYPE;
(*
```

4.4.2.19 SCALED SOLID ANGLE UNIT

A unit of solid angle and a scale factor.

```
*)
ENTITY scaled_solid_angle_unit;
  unit         : solid_angle_unit;
  scale_factor : OPTIONAL REAL;
WHERE
  scale_factor > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

unit: The **solid angle unit.**

scale_factor: An optional multiplicative factor applied to the **unit.**

PROPOSITIONS:

1. If present, the **scale factor** must be greater than zero.

4.5 Geometry

```
*)
SCHEMA ipim_resources_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_geometry_schema,
         ipim_topology_schema,
         ipim_shape_schema,
         ipim_tolerances_schema,
         ipim_material_schema);
(*
```

4.5.1 Geometry Introduction

This section represents the STEP Integrated Product Information Model for Geometry.

```
*)
SCHEMA ipim_geometry_schema;

  EXPORT EVERYTHING;
(*
```

The mathematical symbol convention used in this section is given in Table 10.

Symbol	Definition
a	Scalar quantity
A	Vector quantity
$\langle\rangle$	Vector normalisation
a	Normalised vector (e.g a = $\langle$A$\rangle$ = A/|A|)
*	Vector (cross) product
.	Scalar product
$\lambda(u)$	Parametric curve
$C(x,y,z)$	Analytic curve
$\sigma(u,v)$	Parametric surface
$S(x,y,z)$	Analytic surface
C_x	Partial differential of C with respect to x
σ_u	Partial differential of $\sigma(u,v)$ with respect to u
S_x	Partial differential of S with respect to x

Table 10: Geometry Mathematical Symbology

4.5.1.1 Parameterization of Analytic Curves and Surfaces

All the curves and surfaces specified here have a defined parameterization. In some instances the definitions are in parametric terms. In others, the conic curves and quadric surfaces, the definition is in geometric terms.

In this latter case a local coordinate system (LCS) is used to define the parameterization. The geometric definitions contain some, but not all, of the data required for this. All the entities include a point which forms the origin of the LCS. Two direction vectors are used to complete the definition of the LCS. One is the local Z axis direction and the other is an approximation to the local X axis direction. Let z be the local Z axis direction and a be the approximate local X axis direction. There are two methods, mathematically identical but numerically different, for calculating the local X and Y axis directions.

1. The vector a is projected onto the plane defined by the origin point P and the vector z to give the local X axis direction as $x = \langle a - (a.z)z \rangle$. The local Y axis direction is then given by $y = \langle z * x \rangle$.

2. The local Y axis direction is calculated as $y = \langle z * a \rangle$ and then the local X axis direction is given by $x = \langle y * z \rangle$.

The first method is likely to be the more numerically stable of the two.

4.5.2 Geometry TYPE Definitions

4.5.2.1 CURVE TRANSITION CODE

This type conveys the continuity properties of a composite curve as represented in the sending system. The continuity referred to is geometric continuity, for example **cont same gradient** implies that the tangent vectors of adjacent segments have the same direction but not necessarily the same magnitude.

```
*)
TYPE curve_transition_code = ENUMERATION OF
     (continuous,
      cont_same_gradient,
      cont_same_gradient_same_curvature);
 END_TYPE;
(*
```

4.5.2.2 ENUMERATION CURVE1 PCURVES1

```
*)
TYPE enumeration_curve1_pcurves1 = ENUMERATION OF
      (curve_1,
       pcurve_s1);
END_TYPE;
(*
```

4.5.2.3 BSPLINE CURVE FORM

```
*)
TYPE bspline_curve_form = ENUMERATION OF
      (line_segment,
       circular_arc,
       elliptic_arc,
       parabolic_arc,
```

```
      hyperbolic_arc);
END_TYPE;
(*
```

4.5.2.4 BSPLINE SURFACE FORM

```
*)
TYPE bspline_surface_form = ENUMERATION OF
     (plane_surface,
      cylindrical_surface,
      conical_surface,
      spherical_surface,
      toroidal_surface,
      surface_of_revolution,
      ruled_surface,
      quadric_surface);
END_TYPE;
(*
```

4.5.2.5 INTERSECTION ENUMERATION

```
*)
TYPE intersection_enumeration = ENUMERATION OF
     (basis_curve,
      pcurve_s1,
      pcurve_s2);
END_TYPE;
(*
```

4.5.2.6 UNIFORM TYPE

```
*)
TYPE uniform_type = ENUMERATION OF
     (nonuniform_knots,
      uniform_knots,
      bezier_knots);
END_TYPE;
(*
```

4.5.3 GEOMETRY

Except where specifically stated, all geometry is defined in a right handed cartesian coordinate system. The curve and surface definitions can all be traced to points and/or vectors and/or scalar (length) values.

```
*)
ENTITY geometry
  SUPERTYPE OF (coordinate_system XOR
                point XOR
```

```
                    vector XOR
                    axis_placement XOR
                    transformation XOR
                    curve XOR
                    surface);
  local_coordinate_system : OPTIONAL coordinate_system;
END_ENTITY;
(*
```

4.5.3.1 COORDINATE SYSTEM

```
*)
ENTITY coordinate_system
  SUBTYPE OF (geometry);
  reference_coordinate_system : OPTIONAL coordinate_system;
  axis_set                    : transformation;
WHERE
  axis_set.scale = 1.0 OR
  NOT EXISTS(axis_set.scale);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

reference_coordinate_system: The coordinate system to which the attributes of the axis set refer. (Default is global coordinate system).

axis_set: This takes its origin from the local origin in the transformation and the axis directions from the transformation axis values.

PROPOSITIONS:

4.5.3.2 POINT

A point is a location in some coordinate space.

```
*)
ENTITY point
  SUPERTYPE OF (cartesian_point XOR
                point_on_curve XOR
                point_on_surface)
  SUBTYPE OF (geometry);
END_ENTITY;
(*
```

4.5.3.3 CARTESIAN POINT

This entity represents a point defined in three dimensions.

```
*)
```

```
ENTITY cartesian_point
  SUBTYPE OF (point);
  x_coordinate : REAL;
  y_coordinate : REAL;
  z_coordinate : OPTIONAL REAL;
DERIVE
  space : INTEGER := coordinate_space(z_coordinate);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

x_coordinate: The X coordinate of the point location.

y_coordinate: The Y coordinate of the point location.

z_coordinate: The Z coordinate of the point location.

PROPOSITIONS:

1. The entity is defined in a three dimensional space or in a two dimensional space as determined by the coordinate space function.

4.5.3.4 POINT ON CURVE

A point on curve is a point located in the parametric space of a curve.

```
*)
ENTITY point_on_curve
  SUBTYPE OF (point);
  basis_curve     : curve;
  point_parameter : REAL;
WHERE
  {parametric_lower_limit(basis_curve)
                  <=  point_parameter <=
   parametric_upper_limit(basis_curve)};

  coordinate_space(point_on_curve) = coordinate_space(basis_curve);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_curve: A parameterized curve.

point_parameter: The parameter value of the point location.

PROPOSITIONS:

1. The parametric value must not be outside the parametric range of the curve.
2. The definition space of the point is the same as the curve.

4.5.3.5 POINT ON SURFACE

A point on surface is a point located in the parametric space of a surface.

```
*)
ENTITY point_on_surface
  SUBTYPE OF (point);
  basis_surface      : surface;
  point_parameter_1 : REAL;
  point_parameter_2 : REAL;
WHERE
  {parametric_lower_limit(basis_surface)[1]
          <= point_parameter_1 <=
   parametric_upper_limit(basis_surface)[1]};

  {parametric_lower_limit(basis_surface)[2]
          <= point_parameter_2 <=
  parametric_upper_limit(basis_surface)[2]};
  coordinate_space(point_on_surface) = 3;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_surface: A parameterized surface.

point_parameter_1: The first parameter value of the point location.

point_parameter_2: The second parameter value of the point location.

PROPOSITIONS:

1. The parametric values must not be outside the parametric range of the surface.
2. The point is a three dimensional point.

4.5.3.6 VECTOR

The vector entity collects the two types of vector; direction and vector with magnitude.

```
*)
ENTITY vector
  SUPERTYPE OF (direction XOR
                vector_with_magnitude)
  SUBTYPE OF (geometry);
END_ENTITY;
(*
```

4.5.3.7 DIRECTION

This entity defines a general direction vector.

```
*)
ENTITY direction
  SUPERTYPE OF (default_x_direction XOR
                default_y_direction XOR
                default_z_direction)
  SUBTYPE OF (vector);
  x  :  REAL;
  y  :  REAL;
  z  :  OPTIONAL REAL;
DERIVE
  space : INTEGER := coordinate_space(direction.z);
WHERE
  vector_magnitude(direction) > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

x: The direction ratio with respect to the X axis.

y: The direction ratio with respect to the Y axis.

z: The direction ratio with respect to the Z axis.

space: The dimensionality of the vector.

PROPOSITIONS:

1. The magnitude of the direction vector must not be zero.
2. The direction is defined in three dimensional space or in a two dimensional space as evaluated by the coordinate space function.

4.5.3.8 DEFAULT X DIRECTION

A three space direction parallel to the X axis.

```
*)
ENTITY default_x_direction
  SUBTYPE OF (direction);
WHERE
  x = 1.0;
  y = 0.0;
  z = 0.0
END_ENTITY;
(*
```

PROPOSITIONS:

1. The direction is parallel to the X axis.

4.5.3.9 DEFAULT Y DIRECTION

A three space direction parallel to the Y axis.

```
*)
ENTITY default_y_direction
  SUBTYPE OF (direction);
WHERE
  x = 0.0;
  y = 1.0;
  z = 0.0
END_ENTITY;
(*
```

PROPOSITIONS:

1. The direction is parallel to the Y axis.

4.5.3.10 DEFAULT Z DIRECTION

A three space direction parallel to the Z axis.

```
*)
ENTITY default_z_direction
  SUBTYPE OF (direction);
WHERE
  x = 0.0;
  y = 0.0;
  z = 1.0
END_ENTITY;
(*
```

PROPOSITIONS:

1. The direction is parallel to the Z axis.

4.5.3.11 VECTOR WITH MAGNITUDE

This entity describes a direction vector and the magnitude of the vector. The value of the magnitude attribute is independent of any calculation of the magnitude from considering the components of the orientation attribute.

```
*)
ENTITY vector_with_magnitude
  SUBTYPE OF (vector);
  orientation : direction;
  magnitude   : REAL;
WHERE
  coordinate_space(vector_with_magnitude) =
                   coordinate_space(orientation);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

orientation: The direction.

magnitude: The magnitude of the vector.

PROPOSITIONS:

1. The entity takes its coordinate space from the orientation attribute.

4.5.3.12 AXIS PLACEMENT

An axis placement entity defines the local environment for the definition of a geometry entity. It locates the entity to be defined and gives its orientation.
If a value is defaulted, it takes its default value from the enclosing coordinate system.

```
*)
ENTITY axis_placement
  SUPERTYPE OF (axis1_placement XOR
                axis2_placement)
  SUBTYPE OF (geometry);
END_ENTITY;
(*
```

4.5.3.13 AXIS1 PLACEMENT

An axis placement defined in terms of a locating point and an axis direction.

```
*)
ENTITY axis1_placement
  SUBTYPE OF (axis_placement);
  location : point;
  axis     : OPTIONAL direction;
DERIVE
  z : direction := third_axis(axis);
WHERE
  coordinate_space(location) = 3;
  coordinate_space(axis) = 3;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: The local origin.

axis: The direction of the local Z axis.

z: The normalized direction of the local Z axis.

PROPOSITIONS:

1. axis is a three dimensional direction.

4.5.3.14 AXIS2 PLACEMENT

An axis placement defined in terms of a point and two (orthogonal) axes.

```
*)
ENTITY axis2_placement
  SUBTYPE OF (axis_placement);
  location      : cartesian_point;
  axis          : OPTIONAL direction;
  ref_direction : OPTIONAL direction;
DERIVE
  dim : INTEGER := space_dimension(location,NULL,NULL,NULL);
  x   : direction := place_axis(dim,1,axis,ref_direction);
  y   : direction := place_axis(dim,2,axis,ref_direction);
  z   : direction := place_axis(dim,3,axis,ref_direction);
WHERE
  coordinate_space(axis2_placement) = dim;
  coordinate_space(axis) = coordinate_space(location);
  coordinate_space(ref_direction) = coordinate_space(location);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: The origin of the local coordinate system.

axis: The direction of the local Z axis if **dim** = 3; the direction of the local Y system if **dim** = 2.

ref_direction: The approximate direction of the local X axis.

x: The normalized direction of the local X axis.

y: The normalized direction of the local Y axis

z: The normalized direction of the local Z axis, if **dim** = 3; equal to NULL if **dim** = 2.

PROPOSITIONS:

1. The two axes must not be parallel or anti-parallel.
2. If **axis** and/or **ref direction** is defaulted, these directions are taken from the enclosing coordinate system.
3. The definition space of both **axis** and **ref direction** must be three.

4.5.3.15 TRANSFORMATION

A **transformation** defines a general geometric transformation including translation, rotation, mirroring and scaling.

The normalized vectors $\mathbf{u}_1$, $\mathbf{u}_2$ and $\mathbf{u}_3$ define the columns of an orthogonal matrix **T**. If $|\mathbf{T}| = -1$ then mirroring is included. The local origin point **A**, the scale value S and the matrix **T** together define a transformation.

The transformation for a point with position vector **P** is defined by

$$\mathbf{P} \rightarrow \mathbf{A} + S\mathbf{T}\mathbf{P}$$

```
*)
ENTITY transformation
  SUBTYPE OF (geometry);
  axis1       : OPTIONAL direction;
  axis2       : OPTIONAL direction;
  axis3       : OPTIONAL direction;
  local_origin : OPTIONAL cartesian_point;
  scale       : OPTIONAL REAL;
DERIVE
  dim : INTEGER := space_dimension(local_origin,axis1,axis2,axis3);
  scl : REAL := base_scale(scale);
  org : cartesian_point := base_origin(dim,local_origin);
  u1  : direction := base_axis(dim,1,axis1,axis2,axis3);
  u2  : direction := base_axis(dim,2,axis1,axis2,axis3);
  u3  : direction := base_axis(dim,3,axis1,axis2,axis3);
WHERE
  coordinate_space(transformation) = dim;
  coordinate_space(local_origin) = coordinate_space(axis1);
  coordinate_space(local_origin) = coordinate_space(axis2);
  coordinate_space(local_origin) = coordinate_space(axis3);
  coordinate_space(axis1) = coordinate_space(axis2);
  coordinate_space(axis1) = coordinate_space(axis3);
  coordinate_space(axis2) = coordinate_space(axis3);
  scl > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

axis1: The approximate direction of the required X axis.

axis2: The approximate direction of the required Y axis if **dim** = 3; the exact direction of required Y axis if **dim** = 2.

axis3: The exact direction of the required Z axis if **dim** = 3; ignored if **dim** = 2, NULL may be passed.

local_origin: The required translation.

scale: The required scaling value.

scl: The required scaling value, equal to 1 if not specified.

org: The required translation, equal to (0,0,0) if not specified.

u1: The normalized exact direction of the required X axis.

u2: The normalized exact direction of the required Y axis.

u3: The normalized exact direction of the required Z axis if **dim** = 3; equal to NULL if **dim** = 2.

PROPOSITIONS:

1. **Local origin** defaults to the global origin.

2. Scale defaults to unity and must be greater than zero.

3. The **transformation** is a three-space transformation.

4. The **local origin** is a three dimensional **point**.

5. The order of derivation **u3**, **u1** and finally **u2** is significant.

4.5.3.16 CURVE

Informally, a **curve** can be envisioned as the path of a point moving in its coordinate space.

```
*)
ENTITY curve
  SUPERTYPE OF (line XOR
                conic XOR
                bounded_curve XOR
                curve_on_surface XOR
                offset_curve)
  SUBTYPE OF (geometry);
WHERE
  arcwise_connected(curve);
  arc_length_extent(curve) > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:
PROPOSITIONS:

1. Curves are arcwise connected.

2. Curves have a non zero length.

4.5.3.17 LINE

A **line** is an unbounded **curve** defined by a **point** and a **direction**. The positive direction of the line is in the positive direction of the direction vector.
The curve is parameterised as follows:

$$\begin{aligned} \mathbf{P} &= \text{PNT} \\ \mathbf{V} &= \text{DIR} \\ \lambda(u) &= \mathbf{P} + u\langle\mathbf{V}\rangle \end{aligned}$$

and the parametric range is $-\infty \leq u \leq \infty$.

```
*)
ENTITY line
  SUBTYPE OF (curve);
  pnt      : cartesian_point;
  dir      : direction;
WHERE
```

```
  coordinate_space(pnt) = coordinate_space(dir);
  coordinate_space(line) = coordinate_space(pnt);
  arc_length_extent(line) = infinity;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

pnt: The location of the line.

dir: The direction of the line.

PROPOSITIONS:

1. Pnt and dir must both be 2D or both be 3D entities.
2. The line is unbounded.

4.5.3.18 CONIC

A conic curve is defined in terms of its intrinsic parameters rather than being described in terms of other geometry.
A conic entity always has a local coordinate system defined by **axis placement.**

```
*)
ENTITY conic
  SUPERTYPE OF (circle XOR
                ellipse XOR
                hyperbola XOR
                parabola)
  SUBTYPE OF (curve);
END_ENTITY;
(*
```

4.5.3.19 CIRCLE

A circle is defined by a radius and its location and orientation.
Interpretation of the data should be as follows:

$$\begin{aligned} C &= \text{POSITION.LOCATION} \\ x &= \text{POSITION.X} \\ y &= \text{POSITION.Y} \\ z &= \text{POSITION.Z} \\ R &= \text{RADIUS} \end{aligned}$$

and the circle is parameterised as

$$\lambda(u) = C + R(\cos ux + \sin uy)$$

The parameterisation range is $0 \leq u \leq 360$ degrees.

In the Local Coordinate System defined above the circle is the equation $C = 0$, where

$$C(x, y, z) = x^2 + y^2 - R^2$$

The positive sense of the curve at any point is in the tangent direction, **T**, to the curve at the point, where

$$\mathbf{T} = (-C_y, C_x, 0).$$

```
*)
ENTITY circle
  SUBTYPE OF (conic);
  radius    : REAL;
  position : axis2_placement;
WHERE
  radius > 0.0;
  coordinate_space(circle) = coordinate_space(position);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

radius: The radius of the circle.

position: The location and orientation of the circle. **position.location** defines the center of the circle.

PROPOSITIONS:

1. The **radius** must be greater than zero.

4.5.3.20 ELLIPSE

An **ellipse** is defined by the lengths of the semi-major and semi-minor diameters and the position (center or mid point of the line joining the foci) and orientation of the curve.

Interpretation of the data should be as follows:

C	=	POSITION.LOCATION
x	=	POSITION.X
y	=	POSITION.Y
z	=	POSITION.Z
R_{maj}	=	SEMI_AXIS_1
R_{min}	=	SEMI_AXIS_2

and the ellipse is parameterised as

$$\lambda(u) = \mathbf{C} + R_{maj} \cos u\mathbf{x} + R_{min} \sin u\mathbf{y}$$

The parameterisation range is $0 \leq u \leq 360$ degrees.

In the Local Coordinate System defined above the ellipse is the equation $C = 0$, where

$$C(x, y, z) = x^2 / R_{maj}^2 + y^2 / R_{min}^2 - 1$$

The positive sense of the curve at any point is in the tangent direction, **T**, to the curve at the point, where

$$\mathbf{T} = (-C_y, C_x, 0).$$

```
*)
ENTITY ellipse
  SUBTYPE OF (conic);
  semi_axis_1 : REAL;
  semi_axis_2 : REAL;
  position    : axis2_placement;
WHERE
  semi_axis_1 > 0.0;
  semi_axis_2 > 0.0;
  coordinate_space(ellipse) = coordinate_space(location);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

semi_axis_1: The major radius of the ellipse.

semi_axis_2: The minor radius.

position: The location and orientation of the curve.

PROPOSITIONS:

1. Both the Major and Minor radii must be greater than zero.

4.5.3.21 HYPERBOLA

A **hyperbola** is defined by the lengths of the major and minor radii and the position (mid point of the line joining two foci) and orientation of the curve.
The data should be interpreted as follows:

$$\begin{aligned} C &= \text{POSITION.LOCATION} \\ x &= \text{POSITION.X} \\ y &= \text{POSITION.Y} \\ z &= \text{POSITION.Z} \\ R_{maj} &= \text{SEMI\_AXIS} \\ R_{min} &= \text{SEMI\_IMAG\_AXIS} \end{aligned}$$

and the hyperbola is parameterised as

$$\lambda(u) = C + R_{maj} \cosh ux + R_{min} \sinh uy$$

The parameterisation range is $-\infty < u < \infty$.
In the Local Coordinate System defined above, the hyperbola is represented by the equation $C = 0$, where

$$C(x, y, z) = x^2/R_{maj}^2 - y^2/R_{min}^2 - 1$$

The positive sense of the curve at any point is in the tangent direction, T, to the curve at the point, where

$$T = (-C_y, C_x, 0).$$

The branch of the hyperbola represented is that pointed to by the X direction.

```
*)
ENTITY hyperbola
  SUBTYPE OF (conic);
  semi_axis       : REAL;
  semi_imag_axis  : REAL;
  position        : axis2_placement;
WHERE
  semi_axis > 0.0;
  semi_imag_axis > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

semi_axis: The length of the "major radius" of the hyperbola.

semi_imag_axis: The length of the "minor radius".

position: The location and orientation of the curve.

PROPOSITIONS:

1. The lengths of both radii must be greater than zero.

4.5.3.22 PARABOLA

A **parabola** is defined by its focal length and the position (vertex) and orientation of the curve. Interpretation of the data is as follows:

C = POSITION.LOCATION
x = POSITION.X
y = POSITION.Y
z = POSITION.Z
F = FOCAL_DIST

and the parabola is parameterised as

$$\lambda(u) = \mathbf{C} + F(u^2\mathbf{x} + 2u\mathbf{y})$$

The parameterisation range is $-\infty < u < \infty$.
In the Local Coordinate System defined above, the parabola is represented by the equation $C = 0$, where

$$C(x, y, z) = 4Fx - y^2$$

The positive sense of the curve at any point is in the tangent direction, **T**, to the curve at the point, where

$$\mathbf{T} = (-C_y, C_x, 0).$$

```
*)
ENTITY parabola
  SUBTYPE OF (conic);
  focal_dist : REAL;
```

```
  position    : axis2_placement;
WHERE
  focal_dist <> 0.0;
  arc_length_extent(parabola) = infinity;
  coordinate_space(parabola) = coordinate_space(location)
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

focal_dist: The distance of the focal point from the vertex point.

position: The location and orientation of the curve.

PROPOSITIONS:

1. The focal distance must not be zero.
2. The curve is unbounded.

4.5.3.23 BOUNDED CURVE

A **bounded curve** is a curve of finite arc length.

```
*)
ENTITY bounded_curve
  SUPERTYPE OF (poly_line XOR
                b_spline_curve XOR
                trimmed_curve XOR
                composite_curve)
  SUBTYPE OF (curve);
END_ENTITY;
(*
```

4.5.3.24 POLYLINE

A **polyline** is a **bounded curve** defined by a list of n **points**, $P_0, P_1 \ldots P_n$.
The curve is parameterised as follows:

$$\lambda(u) = \mathbf{P}_i(i + 1 - u) + \mathbf{P}_{i+1}(u - 1)$$

where $i \leq u \leq i + 1$ and with parametric range of $0 \leq u \leq n$.

```
*)
ENTITY line_segment
  SUBTYPE OF (bounded_curve);
  points : LIST [2:#] OF cartesian_point;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

points: The points forming the polyline.

4.5.3.25 B-SPLINE CURVE

Interpretation of the data is as follows:

1. All weights must be positive and the curve is given by:

$$\lambda(u) = \frac{\sum_{i=0}^{k} w_i \mathbf{P}_i N_i(u)}{\sum_{i=0}^{k} w_i N_i(u)}.$$

where

$k+1$ = number of control points,
$\mathbf{P}_i$ = control points,
w_i = weights, and
d = degree.

N_i are the normalized B-spline basis functions of degree d defined on the knot set:

$$u_{i-d}, \ldots, u_{i+1} \qquad u_{j+1} \geq u_j \quad \text{(i.e. non-decreasing)}.$$

2. The knot multiplicities m_i should all be in the range $(1 \ldots \text{degree} + 1)$ and should satisfy:

$$\sum_{i=1}^{L} m_i = \text{degree} + k + 2.$$

In evaluating the basis functions, a knot u_j of, e.g., multiplicity 3 is interpreted as a string u_j, u_j, u_j, in the knot set.

3. The **form number** is used to identify special cases of conic curves. The form number is included for information only. In cases where there is a conflict between the B-spline data and the form number, the B-spline information takes precedence.

4. A flag is provided to indicate whether the curve self intersects or not. An UNDEFINED value for the flag should be interpreted as self intersection is not known.

Identification of B-spline curve default values and options is important for file structure considerations and for efficiency issues in performing computations.

1. A B-spline is *rational* if and only if the weights are not all identical. If it is non-rational, the weights may be defaulted to all being 1.

2. A B-spline is *uniform* if and only if the knots are of multiplicity (degree+1) at the ends, of multiplicity 1 elsewhere,the start knot is 0, and they differ by 1 from the preceeding knot. In this case, the knots and knot multiplicities may be defaulted.

3. A B-spline is piecewise Bezier if it is uniform except that the interior knots have multiplicity degree rather than having multiplicity one. Note that a piecewise Bezier which has only two knots, each of multiplicity (degree+1), is Bezier.

4. Note that defaulting weights and knots may be done independently.

5. Note also that one could have defaulted everything except the control points and degree=1 in the case where the B-spline is a polyline.

It should be noted that every **Bezier curve** has an equivalent representation as a **B-spline curve** but not every **B-spline curve** can be represented as a **Bezier curve**.

To define a piecewise Bezier spline as a B-spline:

- The first (degree+1)-many knots are 0.0.
- The next degree-many knots are 1.0 (we have now defined the knots for one segment, unless it is the last one).
- The next degree-many knots are 1.0 (we have now defined the knots for two segments, again unless the second is the last one).
- Continue to the end of the last segment, call it the n-th segment, at the end of which (degree+1)-many knots with value n are added.

Examples:

- A one segment cubic Bezier curve would have knot sequence(0,1) with multiplicity sequence (4,4).
- A two segment Bezier spline would have knot sequence (0,1,2) with multiplicity sequence (4,3,4).

Doing some arithmetic, if d is the degree, m is the number of knots with multiplicity d and N is the total number of knots for the spline, then

$$\begin{aligned} (d+2+k) &= N \\ &= (d+1)+md+(d+1) \\ \Longrightarrow m &= (k-d)/d \end{aligned}$$

So the knot sequence is $(0, 1, \ldots m, (m+1))$ with multiplicities $(d+1, d, \ldots d, d+1)$.

The setup for surfaces is the same except that m is always 0 as there is only one patch.

```
*)
ENTITY b_spline_curve
  SUBTYPE OF (bounded_curve);
  degree                  : INTEGER;
  upper_index_on_control_points : INTEGER;
  control_points          : ARRAY [0:upper_index_on_control_points]
                                  OF cartesian_point;
  uniform                 : OPTIONAL uniform_type;
  upper_index_on_knots    : INTEGER;
  knot_multiplicities     : OPTIONAL ARRAY [1:upper_index_on_knots]
                                       OF INTEGER;
  knots                   : OPTIONAL ARRAY [1:upper_index_on_knots]
                                       OF REAL;
  weights                 : OPTIONAL
                            ARRAY [0:upper_index_on_control_points]
                                  OF REAL;
  form_number             : OPTIONAL bspline_curve_form;
  self_intersect          : true_false_or_undefined;
```

```
WHERE
  sum(knot_multiplicities) =
      degree + upper_index_on_control_points + 2;
  coordinate_space(b_spline_curve) =
      coordinate_space(control_points);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

degree: The algebraic degree of the basis functions.

upper_index_on_control_points: The upper index on the array of control points; the lower index is 0.

control_points: The control points used to define the geometry of the curve.

uniform: Indication of the type of knot set.

upper_index_on_knots: The upper index on the knot arrays; the lower index is 1.

knot_multiplicities: The multiplicities of the knots.

knots: The set of knots used to define the B-Spline basis functions.

weights: The weights associated with the control points in the rational (homogeneous) case.

form_number: Used to identify particular types of curve.

self_intersect: Flag to indicate whether the curve self intersects or not.

```
*)
RULE b_spline_curve_rule FOR (b_spline_curve);
  LOCAL
    i : INTEGER;
  END_LOCAL;
  IF EXISTS(upper_index_on_knots) THEN
    REPEAT i := 2 TO upper_index_on_knots;
      IF (knots[i] < knots[i-1]) THEN
        VIOLATION;
      END_IF;
    END_REPEAT;
    REPEAT i := 1 TO upper_index_on_knots;
      IF (knot_multiplicities[i] > (degree+1)) THEN
        VIOLATION;
      END_IF;
    END_REPEAT;
  END_IF;
END_RULE;
(*
```

PROPOSITIONS:

1. The Knot set is non-decreasing.
2. The definition space of all the points must be the same.

4.5.3.26 TRIMMED CURVE

Trimming of a curve is permitted

- by parametric value or
- by geometric position or
- by both.

At least one of these must be specified at each end of the curve.
The sense makes it possible to unambiguously define any segment of a closed curve such as a circle.

- SENSE = TRUE if the curve is being traversed in the direction of increasing parametric value;
- SENSE = FALSE otherwise.

The combinations of sense and ordered endpoints makes it possible to define four distinct directed segments connecting two different points on a circle or other closed curve for this purpose. Cyclic properties of the parameter range are assumed.
For an open curve, SENSE = FALSE if PARAMETER_1 > PARAMETER_2. If PARAMETER_2 > PARAMETER_1, then SENSE = TRUE. The sense information is redundant in this case but is essential for a closed curve.

```
*)
ENTITY trimmed_curve
  SUBTYPE OF (bounded_curve);
  basis_curve    : curve;
  parameter_1    : OPTIONAL REAL;
  parameter_2    : OPTIONAL REAL;
  point_1        : OPTIONAL cartesian_point;
  point_2        : OPTIONAL cartesian_point;
  sense          : LOGICAL;
WHERE
  coordinate_space(basis_curve) = coordinate_space(point_1);
  coordinate_space(basis_curve) = coordinate_space(point_2);
  EXISTS(parameter_1) OR EXISTS(point_1);
  EXISTS(parameter_2) OR EXISTS(point_2);
  parameter_1 <> parameter_2;
  ((NOT closed(basis_curve) AND
  (sense = (parameter_2 > parameter_1))) OR (closed(basis_curve)));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_curve: The curve to be trimmed.

parameter_1: The first trimming point in the parametric space of basis curve.

parameter_2: The second trimming point in the parametric space of basis curve.

point_1: The first trimming point in cartesian space.

point_2: The second-trimming point in cartesian space.

sense: Flag to indicate whether the direction of the trimmed curve agrees with or is opposed to the direction of basis curve.

PROPOSITIONS:

1. The basis curve and the two points must all be defined in the same definition space.
2. Parameter 1 and parameter 2 must lie in the parametric range of the referenced curve.
3. At least one of parameter 1 and point 1 must be specified.
4. At least one of parameter 2 and point 2 must be specified.
5. If both parameter 1 and parameter 2 are specified, they must have different values.
6. For curves with multiple representations, parameter 1 and parameter 2 refer to the master representation only.

4.5.3.27 COMPOSITE CURVE

A composite curve is a collection of curves joined end to end.

The parameter range defines the parameter values assigned to the curve transition points. Each constituent curve receives a *rescale parameter*. This ensures that the transmitted composite curve is subject to a simple linear re-parameterisation. If param range is omitted the parametric length l_i of each arc is taken from the referenced bounded curve with a simple cumulative parameterisation of the composite curve from 0 to $\sum l_i$.

CLOSED = TRUE denotes a closed curve with coincident first and last point; For such a curve the Kth transition refers to the continuity at the start and finish of the curve. If CLOSED = FALSE the list of transitions will contain $K - 1$ entries.

The list of senses is used to denote whether or not the sense of a component curve is the same or opposite to its sense as originally defined.

```
*)
ENTITY composite_curve
  SUBTYPE OF (bounded_curve);
  closed_curve    : LOGICAL;
  segments        : LIST [2:#] OF bounded_curve;
  senses          : ARRAY [1:k] OF LOGICAL;
  transitions     : ARRAY [1:trans] OF curve_transition_code;
  param_range     : OPTIONAL ARRAY [1:k] OF REAL;
  self_intersect : true_false_or_undefined;
DERIVE
  k : INTEGER := SIZEOF(segments);
WHERE
  (closed_curve AND trans = k) OR
  (NOT closed_curve AND trans = k-1);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

closed_curve: Indication of whether the curve is closed.

segments: The component bounded curves.

senses: Indicators of whether the sense of the component curves agree or differ from the sense of the composite curve.

transitions: Form of transition between one component curve and the next.

param_range: Composite curve parameter value at the join of adjacent component curve. Optional.

self_intersect: Indication of whether the curve self intersects or not.

```
*)
RULE composite_curve_rule FOR (composite_curve);
  LOCAL
    k : INTEGER := 0;
  END_LOCAL;
  REPEAT FOR EACH bounded_curve IN composite_curve.segments;
    k := k + 1;
    IF (composite_curve.senses[k] <>
        agreement(bounded_curve, composite_curve)) THEN
          VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. The curve must have at least two component curves.

4.5.3.28 CURVE ON SURFACE

A curve on surface is...

```
*)
ENTITY curve_on_surface
  SUPERTYPE OF (pcurve XOR
                surface_curve XOR
                intersection_curve XOR
                composite_curve_on_surface)
  SUBTYPE OF (curve);
END_ENTITY;
(*
```

4.5.3.29 PCURVE

A pcurve is a curve defined in the two-dimensional parametric space of a reference surface. Although it is defined by a curve of type 2D, the variables involved are u and v rather than the x, y cartesian coordinates.

The curve is only defined within the parametric range of the surface.

```
*)
ENTITY pcurve
  SUBTYPE OF (curve_on_surface);
  basis_surface : surface;
  basis_curve   : curve;
WHERE
  coordinate_space(basis_curve) = 2;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_surface: The surface on which the curve lies.

basis_curve: A 2D curve.

PROPOSITIONS:

1. The basis curve must be a two dimensional curve.

4.5.3.30 SURFACE CURVE

A surface curve is a curve on a surface. The curve is represented as a curve (curve 1) in three dimensional space and possibly as a curve (pcurve s1) in the two dimensional parametric space of the surface.

Pcurve s1 must reference surface 1 in its definition and be parameterised to have the same sense as curve 1.

```
*)
ENTITY surface_curve
  SUBTYPE OF (curve_on_surface);
  surface_1              : surface;
  curve_1                : curve;
  pcurve_s1              : OPTIONAL pcurve;
  master_representation  : OPTIONAL enumeration_curve1_pcurves1;
WHERE
  coordinate_space(curve_1) = 3;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

surface_1: The surface on which the curve lies.

curve_1: The curve which is the three dimensional image of the surface curve.

pcurve_s1: The curve which is the two dimensional parametric image of surface curve.

master_representation: Indication of representation "preferred" by the sending system. If NULL then curve 1 is preferred.

PROPOSITIONS:

1. Curve 1 must be defined in three space.

4.5.3.31 INTERSECTION CURVE

An **intersection curve** is defined by intersecting two surfaces. The multiple representation permits the receiving system to recompute the intersection curve if it has the capability.

The **master representation** indicates the preference of the generating system for the preferred accurate representation.

```
*)
ENTITY intersection_curve
  SUBTYPE OF (curve_on_surface);
  pcurve_s1              : OPTIONAL pcurve;
  surface_s1             : OPTIONAL surface;
  pcurve_s2              : OPTIONAL pcurve;
  surface_s2             : OPTIONAL surface;
  basis_curve            : curve;
  master_representation  : OPTIONAL intersection_enumeration;
  self_intersect         : true_false_or_undefined;
WHERE
  coordinate_space(basis_curve) = 3;
  EXISTS(pcurve_s1) OR EXISTS(surface_s1);
  EXISTS(pcurve_s1) OR EXISTS(surface_s1);
  direction_fun(pcurve_s1) = direction_fun(basis_curve);
  direction_fun(pcurve_s2) = direction_fun(basis_curve);
  direction_fun(pcurve_s1) = direction_fun(pcurve_s2.pcurve);
  domain(pcurve_s1) = domain(basis_curve);
  domain(pcurve_s2) = domain(basis_curve);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_curve: The intersection curve of the two surfaces in cartesian space.

pcurve_s1: The intersection curve of the two surfaces defined in the parametric space of the first surface.

surface_s1: The first of the two intersecting surfaces.

pcurve_s2: The intersection curve of the two surfaces defined in the parametric space of the second surface.

surface_s2: The second surface.

self_intersect: Indication of whether the curve self intersects.

master_representation: Indicator of which curve the sending system viewed as the "master" representation. If it is NULL then the **basis curve** is preferred.

PROPOSITIONS:

1. The directions of all three representative curves, if they exist, must be the same.
2. All three curves, if they exist, must represent the same point set.

4.5.3.32 COMPOSITE CURVE ON SURFACE

A composite curve on surface is an assembly of curves on a surface.
Each curve on surface in the segment list must be on the reference surface. It may be

- a surface curve or
- an intersection curve or
- a composite curve on surface.

There must be at least positional continuity between adjacent segments.
CLOSED = TRUE denotes a closed curve with coincident first and last point. For such a curve the Kth transition refers to the continuity at the start and finish of the curve. If CLOSED = FALSE the list of transitions will contain $K - 1$ entries. CLOSED refers to the nature of the curve of type 3D, not to the PCURVEs.

```
*)
ENTITY composite_curve_on_surface
  SUBTYPE OF (curve_on_surface);
  basis_surface : surface;
  closed_curve  : LOGICAL;
  segments      : LIST [2:#] OF curve_on_surface;
  transitions   : ARRAY [1:trans] OF curve_transition_code;
  senses        : ARRAY [1:k] OF LOGICAL;
 DERIVE
  k : INTEGER := SIZEOF(segments);
WHERE
  SIZEOF(segments) > 1;
  NOT (pcurve IN segments);
  NOT self_intersect(composite_curve_on_surface);
  (closed_curve AND trans = k) OR
  (NOT closed_curve AND trans = k-1);
 END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_surface: The surface on which the curve is defined.

closed_curve: Indication of whether the curve is closed.

segments: The component curves.

transitions: The transition code for adjacent component curves. This applies to 3-D representations of curves, not to pcurves.

senses: Indication of whether the sense of a component curve agrees with the sense of the composite curve.

```
*)
RULE composite_curve_on_surface_rule FOR
```

```
                (composite_curve_on_surface);
  LOCAL
    k : INTEGER := 0;
  END_LOCAL;
  REPEAT FOR EACH curve_on_surface IN
                              composite_curve_on_surface.segments;
    k := k + 1;
    IF (senses[k] <>
        agreement(curve_on_surface, composite_curve_on_surface))
      THEN VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. The curve is non-self-intersecting.
2. The list of senses is used to denote whether or not the sense of a segment is the same or opposite to its sense as originally defined.
3. The list of segments must not include any pcurves.

4.5.3.33 OFFSET CURVE

An **offset curve** is a curve a constant distance from a basis curve.

The underlying curve must have a well defined tangent at every point. In the case of a composite curve the transition code between each segment must be at least **cont same gradient.**

```
*)
ENTITY offset_curve
  SUPERTYPE OF (d2_offset_curve XOR
                d3_offset_curve)
  SUBTYPE OF (curve);
END_ENTITY;
(*
```

4.5.3.34 D2 OFFSET CURVE

This defines a simple plane-offset curve by offsetting by **distance** along the normal to **basis curve** in the plane of **basis curve.**

Distance may be positive or negative. A positive value of **distance** defines an offset in the direction N * T, where N is the normal to the plane of the curve (usually the *z* axis) and T is the unit tangent vector at the given point.

The underlying curve must be 2D.

```
*)
ENTITY d2_offset_curve
  SUBTYPE OF (offset_curve);
```

```
  basis_curve    : curve;
  distance       : REAL;
  self_intersect : true_false_or_undefined;
WHERE
  coordinate_space(basis_curve) = 2;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_curve: The curve that is being offset.

distance: The distance of the offset curve from the basis curve.

self_intersect: An indication of whether the curve self intersects.

PROPOSITIONS:

1. The underlying curve must be defined in two space.

4.5.3.35 D3 OFFSET CURVE

This defines a curve offset in three-space.
The offset curve at any point (parameter) on the basis curve is in the direction V * T where V is the fixed reference direction and T is the unit tangent to the basis curve.

```
*)
ENTITY d3_offset_curve
  SUBTYPE OF (offset_curve);
  basis_curve    : curve;
  distance       : REAL;
  self_intersect : true_false_or_undefined;
  ref_direction  : direction;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_curve: The curve that is being offset.

distance: The distance of the offset curve from the basis curve.

self_intersect: An indication of whether the curve self intersects.

ref_direction: The direction used to define the direction of the offset curve from the basis curve.

4.5.3.36 SURFACE

A surface can informally be envisioned as a continuously changing curve moving in space, formally it is a 2-manifold in 3-space.

```
*)
ENTITY surface
  SUPERTYPE OF (elementary_surface XOR
                swept_surface XOR
                bounded_surface XOR
                offset_surface)
  SUBTYPE OF (geometry);
WHERE
  area_extent(surface) > 0;
  arcwise_connected(surface);
END_ENTITY;
(*
```

PROPOSITIONS:

1. Surfaces have non zero area.
2. Surfaces are arcwise connected.
3. Surfaces have no "dangling edges".

4.5.3.37 ELEMENTARY SURFACE

```
*)
ENTITY elementary_surface
  SUPERTYPE OF (plane XOR
                cylindrical_surface XOR
                conical_surface XOR
                spherical_surface XOR
                toroidal_surface)
  SUBTYPE OF (surface);
END_ENTITY;
(*
```

4.5.3.38 PLANE

A plane is defined by a point on the surface and the normal direction to the surface.
The data is to be interpreted as follows:

$$\begin{aligned} C &= \text{POSITION.LOCATION} \\ z &= \langle \text{POSITION.Z} \rangle \\ x &= \langle \text{POSITION.X} \rangle \\ y &= \langle \text{POSITION.Y} \rangle \end{aligned}$$

and the surface is parameterized as

$$\sigma(u, v) = C + xu + yv$$

where the parameterisation range is $-\infty < u, v < \infty$.

```
*)
ENTITY plane
  SUBTYPE OF (elementary_surface);
  position : axis2_placement;
WHERE
  coordinate_space(position.location) = 3;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

position: The location and orientation of the surface.

PROPOSITIONS:

1. The point defining a point on the surface must be 3-D.

4.5.3.39 CYLINDRICAL SURFACE

A **cylindrical surface** is defined by its radius and its orientation and location.
The data is to be interpreted as follows:

$$\begin{aligned} C &= \text{POSITION.LOCATION} \\ x &= \text{POSITION.X} \\ y &= \text{POSITION.Y} \\ z &= \text{POSITION.Z} \\ R &= \text{RADIUS} \end{aligned}$$

and the surface is parameterized as

$$\sigma(u, v) = \mathbf{C} + R(\cos u\mathbf{x} + \sin u\mathbf{y}) + v\mathbf{z}$$

where the parameterisation range is $0 \leq u \leq 360$ degrees and $-\infty \leq v \leq \infty$.
In the Local Coordinate System defined above, the surface is represented by the equation $S = 0$, where

$$S(x, y, z) = x^2 + y^2 - R^2$$

The positive direction of the normal to the surface at any point on the surface is given by

$$\mathbf{N} = (S_x, S_y, S_z)$$

or

$$\mathbf{N}(u, v) = \cos u\mathbf{x} + \sin u\mathbf{y}$$

```
*)
ENTITY cylindrical_surface
  SUBTYPE OF (elementary_surface);
  radius     : REAL;
  position   : axis2_placement;
WHERE
  coordinate_space(position.location) = 3;
  radius > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

radius: The radius of the cylinder.

position: The location and orientation of the cylinder.

PROPOSITIONS:

1. The defining location must be 3D.
2. The radius must be greater than zero.

4.5.3.40 CONICAL SURFACE

A **conical surface** is defined by the semi-angle, the location and orientation and by the radius of the cone in the plane passing through the locating point normal to the cone axis.
The data is to be interpreted as follows:

$$\begin{aligned} \mathbf{C} &= \text{POSITION.LOCATION} \\ \mathbf{x} &= \text{POSITION.X} \\ \mathbf{y} &= \text{POSITION.Y} \\ \mathbf{z} &= \text{POSITION.Z} \\ R &= \text{RADIUS} \\ S &= \text{SEMILANGLE} \end{aligned}$$

and the surface is parameterised as

$$\sigma(u, v) = \mathbf{C} + (R + v \tan S)(\cos u\mathbf{x} + \sin u\mathbf{y}) + v\mathbf{z}$$

where the parameterisation range is $0 \leq u \leq 360$ degrees and $-\infty \leq v \leq \infty$.
In the Local Coordinate System defined above, the surface is represented by the equation $\mathcal{S} = 0$, where

$$\mathcal{S}(x, y, z) = x^2 + y^2 - (R - z \tan S)^2$$

The positive direction of the normal to the surface at any point on the surface is given by

$$\mathbf{N} = (\mathcal{S}_x, \mathcal{S}_y, \mathcal{S}_z)$$

or

$$\mathbf{N}(u, v) = (\cos u\mathbf{x} + \sin u\mathbf{y} - \tan S\mathbf{z})/(1 + (\tan S)^2)$$

If the radius is zero then the cone apex is at the point (0,0,0) in the Local Coordinate System (i.e at POSITION.LOCATION).

```
*)
ENTITY conical_surface
  SUBTYPE OF (elementary_surface);
  semi_angle : REAL;
  radius     : REAL;
  position   : axis2_placement;
WHERE
  coordinate_space(position.location) = 3;
  (0 < semi_angle < 90.0);
  radius >= 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

semi_angle: The cone semi-angle in degrees.

radius: The radius of the circular curve of intersection between the cone and a plane perpendicular to the axis of the cone passing through the location point.

position: The location and orientation of the surface.

PROPOSITIONS:

1. The defining position must be 3D.
2. The semi-angle must be between 0 and 90 degrees.
3. The radius must not be less than zero.

4.5.3.41 SPHERICAL SURFACE

A **spherical surface** is defined by the radius and the location and orientation of the surface. The data is to be interpreted as follows:

$$\begin{aligned} C &= \text{POSITION.LOCATION} \\ x &= \text{POSITION.X} \\ y &= \text{POSITION.Y} \\ z &= \text{POSITION.Z} \\ R &= \text{RADIUS} \end{aligned}$$

and the surface is parameterized as

$$\sigma(u, v) = C + R\cos v(\cos ux + \sin uy) + R\sin vz$$

where the parameterisation range is $0 \leq u \leq 360$ degrees and $-90 \leq v \leq 90$ degrees.

In the Local Coordinate System defined above, the surface is represented by the equation $S = 0$, where

$$S(x, y, z) = x^2 + y^2 + z^2 - R^2$$

The positive direction of the normal to the surface at any point on the surface is given by

$$N = (S_x, S_y, S_z)$$

or

$$N(u, v) = \cos v(\cos ux + \sin uy) + \sin vz$$

that is, it is directed away from the center of the sphere.

```
*)
ENTITY spherical_surface
  SUBTYPE OF (elementary_surface);
  radius    : REAL;
  position : axis2_placement;
WHERE
  coordinate_space(position.location) = 3;
  radius > 0.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

radius: The radius of the sphere.

position: The location and orientation of the surface.

PROPOSITIONS:

1. The position must be defined in three space.
2. The radius must be greater than zero.

4.5.3.42 TOROIDAL SURFACE

A toroidal surface is defined by the major and minor radii and the position and orientation of the surface.
The data is to be interpreted as follows:

$$\begin{aligned} \mathbf{C} &= \text{POSITION.LOCATION} \\ \mathbf{x} &= \text{POSITION.X} \\ \mathbf{y} &= \text{POSITION.Y} \\ \mathbf{z} &= \text{POSITION.Z} \\ R &= \text{MAJOR\_RADIUS} \\ r &= \text{MINOR\_RADIUS} \end{aligned}$$

and the surface is parameterized as

$$\sigma(u, v) = \mathbf{C} + (R + r\cos u)(\cos v\mathbf{x} - \sin v\mathbf{y}) + r\sin u\mathbf{z}$$

where the parameterisation range is $0 \leq u, v \leq 360$ degrees.
In the Local Coordinate System defined above, the surface is represented by the equation $S = 0$, where

$$S(x, y, z) = x^2 + y^2 + z^2 - 2R\sqrt{x^2 + y^2} - r^2 + R^2$$

The positive direction of the normal to the surface at any point on the surface is given by

$$\mathbf{N} = (S_x, S_y, S_z)$$

or

$$\mathbf{N}(u, v) = \cos v(\cos u\mathbf{x} + \sin u\mathbf{y}) + \sin v\mathbf{z}$$

```
*)
ENTITY toroidal_surface
  SUBTYPE OF (elementary_surface);
  major_radius : REAL;
  minor_radius : REAL;
  position     : axis2_placement;
WHERE
  coordinate_space(position.location) = 3;
  minor_radius > 0.0;
  major_radius > minor_radius;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

major_radius: The major radius of the torus.

minor_radius: The minor radius of the torus.

position: The location and orientation of the surface.

PROPOSITIONS:

1. The location must be defined in three space.
2. The Minor Radius must be greater than zero.
3. The Major Radius must be greater than the Minor Radius.

4.5.3.43 SWEPT SURFACE

A **swept surface** is one that is constructed by sweeping a curve by another curve.

```
*)
ENTITY swept_surface
  SUPERTYPE OF (surface_of_revolution OR
                surface_of_linear_extrusion)
  SUBTYPE OF (surface);
END_ENTITY;
(*
```

4.5.3.44 SURFACE OF REVOLUTION

A **surface of revolution** is the surface obtained by rotating a curve a complete revolution about an axis.

The data should be interpreted as below.

The parameterisation is as follows, where the curve has a parameterisation $\lambda(v)$:

$$\begin{aligned} C &= \text{AXIS\_POINT} \\ V &= \text{AXIS} \\ \sigma(u,v) &= C + (\lambda(v) - C)\cos u + ((\lambda(v) - C).V)V(1 - \cos u) + V*(\lambda(v) - C)\sin u \end{aligned}$$

In order to produce a single valued surface with a complete revolution, the curve should be such that when expressed in a cylindrical coordinate system (r, ϕ, z) centred at C with axis V no two distinct parametric points on the curve should have the same values for (r, z).

For a complete surface of revolution the parametric range is $0 \leq u \leq 360$ degrees.

The parameter range for v is defined with the referenced curve.

The geometric form of the surface is not dependent upon the curve parameterisation.

```
*)
ENTITY surface_of_revolution
  SUBTYPE OF (swept_surface);
  position       : axis1_placement;
  revolved_curve : curve;
```

```
WHERE
  coordinate_space(position.location) =
      coordinate_space(revolved_curve);

  coordinate_space(position.axis) =
      coordinate_space(revolved_curve);

  NOT self_intersect(surface_of_revolution);
  NOT pcurve IN TYPEOF(revolved_curve);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

position: A point on the axis of revolution and the direction of the axis of revolution.

revolved_curve: The curve that is revolved about the axis line.

PROPOSITIONS:

1. The axis and location must be defined in the same space (two or three) as the revolved curve.
2. The surface must not self intersect.
3. The revolved curve must not be a pcurve.

4.5.3.45 SURFACE OF LINEAR EXTRUSION

This surface is a simple swept surface or a generalised cylinder obtained by sweeping a curve in a given direction.
The parameterisation is as follows, where the curve has a parameterisation $\lambda(u)$:

$$\begin{aligned} V &= \text{AXIS} \\ \sigma(u,v) &= \lambda(u) + vV \end{aligned}$$

The parameterisation range for v is $-\infty \leq v \leq \infty$ and for u is defined by the curve parameterisation.

```
*)
ENTITY surface_of_linear_extrusion
  SUBTYPE OF (swept_surface);
  axis           : direction;
  extruded_curve : curve;
WHERE
  manifold(surface_of_linear_extrusion);
  NOT pcurve IN TYPEOF(extruded_curve);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

axis: The direction of extrusion.

extruded_curve: The curve to be swept.

PROPOSITIONS:

1. The surface must be manifold.
2. The extruded curve must not be a pcurve.

4.5.3.46 BOUNDED SURFACE

```
*)
ENTITY bounded_surface
  SUPERTYPE OF (b_spline_surface XOR
                rectangular_trimmed_surface XOR
                curve_bounded_surface XOR
                rectangular_composite_surface)
  SUBTYPE OF (surface);
END_ENTITY;
(*
```

4.5.3.47 B-SPLINE SURFACE

The data is to be interpreted as follows:

1. The symbology used here is:

$$K1 = \text{UPPER\_INDEX\_ON\_U\_CONTROL\_POINTS}$$

$$K2 = \text{UPPER\_INDEX\_ON\_V\_CONTROL\_POINTS}$$

$$\mathbf{P}_{ij} = \text{CONTROL\_POINTS}$$

$$w_{ij} = \text{WEIGHTS}$$

$$d1 = \text{U\_DEGREE}$$

$$d2 = \text{V\_DEGREE}$$

2. The control points and weights are ordered as

$$\mathbf{P}_{00}, \mathbf{P}_{10}, \mathbf{P}_{20}, \cdots\cdots, \mathbf{P}_{(K1-1)K2}, \mathbf{P}_{K1K2}$$

3. The restrictions on knot multiplicities and the interpretations of multiple knots are as defined for **bspline curve.**
4. The form number is used to identify specific quadric surface types which must have degree two, ruled surfaces and surfaces of revolution. As with the **bspline curve**, the form number is informational only and the spline data takes precedence.
5. The surface is to be interpreted as follows:

$$\sigma(u,v) = \frac{\sum_{i=0}^{K1}\sum_{j=0}^{K2} w_{ij}\mathbf{P}_{ij}N_i(u)N_j(v)}{\sum_{i=0}^{K1}\sum_{j=0}^{K2} w_{ij}N_i(u)N_j(v)}$$

6. For further details, see the bspline curve description.

```
*)
ENTITY b_spline_surface
  SUBTYPE OF (bounded_surface);
  u_degree           : INTEGER;
  v_degree           : INTEGER;
  u_upper            : INTEGER;
  v_upper            : INTEGER;
  control_points     : ARRAY [0:u_upper] OF
                       ARRAY [0:v_upper] OF cartesian_point;
  knot_u_upper       : INTEGER;
  u_multiplicities   : ARRAY [1:knot_u_upper] OF INTEGER;
  u_knots            : ARRAY [1:knot_u_upper] OF REAL;
  knot_v_upper       : INTEGER;
  v_multiplicities   : ARRAY [1:knot_v_upper] OF INTEGER;
  v_knots            : ARRAY [1:knot_v_upper] OF REAL;
  weights            : OPTIONAL ARRAY [0:u_upper] OF
                                ARRAY [0:v_upper] OF REAL;
  form_number        : OPTIONAL bspline_surface_form;
WHERE
  coordinate_space(control_points) = 3;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

u_degree: Algebraic degree of basis functions in u.

v_degree: Algebraic degree of basis functions in v..

u_upper: Upper index on control points in u direction.

v_upper: Upper index on control points in v direction.

control_points: Array of control points defining surface geometry.

knot_u_upper: Upper index on u multiplities and u knots.

u_multiplicities: Knot multiplicities for u knots.

u_knots: Knot set defining basis functions in u.

knot_v_upper: Upper index on v multiplities and v knots.

v_multiplicities: Knot multiplicities for v knots.

v_knots: Knot set defining basis functions in v.

weights: The weights associated with the control points in the rational (homogeneous) case.

form_number: Indicator of special surface type.

PROPOSITIONS:

1. The control points are all 3-D.

4.5.3.48 RECTANGULAR TRIMMED SURFACE

The trimmed surface is a simple **bounded surface** in which the boundaries are the constant parametric lines u_{min} = UMIN, u_{max} = UMAX, v_{min} = VMIN and v_{max} = VMAX. All these values should be within the parametric range of the referenced surface with $u_{min} < u_{max}$ and $v_{min} < v_{max}$, unless the surface is closed in the given parametric direction.

If any of u_{min}, u_{max}, v_{min} or v_{max} are beyond the parametric range of the surface, a default value equal to the natural surface boundary is assumed.

```
*)
ENTITY rectangular_trimmed_surface
  SUBTYPE OF (bounded_surface);
  basis_surface : surface;
  umin          : REAL;
  umax          : REAL;
  vmin          : REAL;
  vmax          : REAL;
WHERE
  umin <> umax;
  vmin <> vmax;
  embedded(rectangular_trimmed_surface, basis_surface);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_surface: Surface being trimmed.

umin: First u parametric value.

umax: Second u parametric value.

vmin: First v parametric value.

vmax: Second v parametric value.

PROPOSITIONS:

1. **Umin** and **umax** must have different values.
2. **Vmin** and **vmax** must have different values.
3. The domain of the trimmed surface must be within the domain of the surface being trimmed.

4.5.3.49 CURVE BOUNDED SURFACE

The **curve bounded surface** is a parametric surface with curved boundaries defined by one or more **composite curve on surface**. One of these may be the outer boundary; any number of inner boundaries is permissible.

If the outer boundary is omitted, the natural boundaries of the surface are assumed.

The interior of the bounded surface is defined to be in the direction of **N*T** from any point on the boundary, where **N** is the surface normal and **T** the boundary curve tangent vector at this point. The region so defined must be arcwise connected.

```
*)
ENTITY curve_bounded_surface
  SUBTYPE OF (bounded_surface);
  basis_surface : surface;
  boundaries    : SET [1:#] OF composite_curve_on_surface;
  outer_present : LOGICAL;
WHERE
  embedded(bounded_surface, basis_surface);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_surface: The surface to be bounded.

boundaries: The bounding curves of the surface.

outer_present: Indication of whether the outer boundary is explicitly present (TRUE) or is defaulted to the natural parametric boundary of the surface.

```
*)
RULE bounded_surface_rule FOR (bounded_surface);
  LOCAL
    k : INTEGER := 0;
  END_LOCAL;
  k := SIZEOF(boundaries);
  REPEAT i := 1 TO k;
    IF (boundaries[i].closed <> TRUE) THEN
      VIOLATION;
    END_IF;
    REPEAT j := 1 TO k;
      IF ((i<>j) AND
           connected(boundaries[i], boundaries[j])) THEN
        VIOLATION;
      END_IF;
    END_REPEAT;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. Each boundary must be a **composite curve on surface** where **closed** is TRUE. If it includes any parts of the natural (constant parametric limit) boundary of the surface, this must appear explicitly in the composite curve definition.

2. The boundaries must be disjoint with each other and with the outer boundary.

3. Each **boundary** must be in the domain of the **basis surface**.

4.5.3.50 OFFSET SURFACE

This is a procedural definition of a simple offset surface at a normal distance from the originating surface. **Distance** may be positive or negative to indicate the preferred side of the surface. The positive side is defined as follows:

1. Define unit tangent vectors (just like those for the definition of the offset curve) of the base surface in the u and v directions; denote these by σ_u and σ_v.
2. Take the cross product, $N = \sigma_u * \sigma_v$, of these (which had must be linearly independent, or there is no offset surface).
3. Normalize to get a unit normal (to the surface) vector.
4. Move the offset distance (which may be zero, as it also may be with the offset curve) along that vector to find the point on the offset surface.

```
*)
ENTITY offset_surface
  SUBTYPE OF (surface);
  basis_surface  : surface;
  distance       : REAL;
  self_intersect : true_false_or_undefined;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

basis_surface: The surface that is to be offset.

distance: The offset distance.

self_intersect: Flag to indicate self-intersection status of the surface.

PROPOSITIONS:

1. The definition allows the offset surface to be self-intersecting.

4.5.3.51 RECTANGULAR COMPOSITE SURFACE

This is a composite surface having a simple rectangular topology with **n_u** patches in the u direction and **n_v** patches in the v direction. Each component surface must be bounded and topologically rectangular and therefore be either a **bspline surface** or a **rectangular trimmed surface**. It is required that there is at least positional continuity between adjacent surfaces.

The lists of surfaces and senses are ordered as (1,1), (2,1) ... (n_u, n_v). If every **u sense** is TRUE adjacent patches in the u direction are adjoined such that the u-high boundary of the first patch coincides with the u-low boundary of the second patch. **U sense** being FALSE for a patch would result in adjoining its u-low boundary to the u-low boundary of the next patch in the sequence. Similar considerations apply to the **v sense**.

Each patch is, if necessary, reparameterised from 0 to 1 and the resulting composite surface has parametric range 0 to n_u, 0 to n_v.

```
*)
ENTITY rectangular_composite_surface
  SUBTYPE OF (bounded_surface);
  n_u       : INTEGER;
  n_v       : INTEGER;
  surfaces  : ARRAY [1:n_u] OF
              ARRAY [1:n_v] OF surface;
  u_senses  : ARRAY [1:n_u] OF
              ARRAY [1:n_v] OF LOGICAL;
  v_senses  : ARRAY [1:n_u] OF
              ARRAY [1:n_v] OF LOGICAL;
WHERE
  n_u > 0;
  n_v > 0;
  TYPEOF(surfaces) <> curve_bounded_surface;
  TYPEOF(surfaces) <> rectangular_composite_surface;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

n_u: Number of surfaces in U direction.

n_v: Number of surfaces in V direction.

surfaces: Rectangular array of component surfaces.

u_senses: Rectangular array of flags to indicate whether sense of U parameterization in composite agrees with (TRUE) original surface.

v_senses: Rectangular array of flags to indicate whether sense of V parameterization in composite agrees with (TRUE) original surface.

PROPOSITIONS:

1. There must be at least one surface.
2. Neither **curve bounded surfaces** nor **rectangular composite surfaces** can be part of the composition of a **rectangular composite surface.**

4.5.3.52 Deprecated Entities

The use of the following entities is deprecated. Schemas referencing these entities should be adjusted accordingly.

4.5.3.52.1 CARTESIAN THREE COORDINATE

```
*)
ENTITY cartesian_three_coordinate;
  location : cartesian_point;
WHERE
  EXISTS(location.z_coordinate);
```

```
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: The geometric definition.

PROPOSITIONS:

1. This is a three dimensional point.

4.5.3.52.2 CARTESIAN TWO COORDINATE

```
*)
ENTITY cartesian_two_coordinate;
  location : cartesian_point;
WHERE
  NOT EXISTS(location.z_coordinate);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: The geometric definition.

PROPOSITIONS:

1. This is a two dimensional point.

4.5.3.52.3 THREE SPACE DIRECTION

```
*)
ENTITY three_space_direction;
  definition : direction;
WHERE
  EXISTS(definition.z);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The geometric definition.

PROPOSITIONS:

1. This is a three dimensional direction vector.

4.5.3.52.4 TWO SPACE DIRECTION

```
*)
ENTITY two_space_direction;
  definition : direction;
WHERE
  NOT EXISTS(definition.z);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The geometric definition.

PROPOSITIONS:

1. This is a two dimensional direction vector.

4.5.4 Geometry FUNCTION Definitions

4.5.4.1 ARC LENGTH EXTENT

This function returns the arc length of the curve argument.

```
*)
FUNCTION arc_length_extent(arg : curve) : REAL;
END_FUNCTION;
(*
```

4.5.4.2 AREA EXTENT

This function returns the area of the surface argument.

```
*)
FUNCTION area_extent(arg : surface) : REAL;
END_FUNCTION;
(*
```

4.5.4.3 AXIS FUNCTIONS

4.5.4.3.1 BASE AXIS

```
*)
FUNCTION base_axis(dim, axis_number : INTEGER;
                   axis1, axis2, axis3 : direction) : direction;
  LOCAL
    vec, u1, u2, u3 : direction;
    factor          : REAL;
  END_LOCAL;
  IF (dim = 3) THEN
    u3 := third_axis(axis3);
```

```
    u1 := first_proj_axis(u3,axis1);
    u2 := second_proj_axis(u3,u1,axis2);
  ELSE
    u3 = NULL;
    IF EXISTS(axis1) THEN
      u1 := normalize(axis1);
      u2 := orthogonal_complement(u1);
      IF EXISTS(axis2) THEN
        factor := dot_product(axis2,u2);
        IF (factor < 0.0) THEN
          u2.x := -u2.x;
          u2.y := -u2.y;
        END_IF;
      END_IF;
    ELSE IF EXISTS(axis2) THEN
      u2 := normalize(axis2);
      u1 := orthogonal_complement(u2);
      u1.x := -u1.x;
      u1.y := - u1.y;
    ELSE
      u1.x := 1.0;
      u1.y := 0.0;
      u2.x := 0.0;
      u2.y := 1.0;
    END_IF;
  END_IF;
  IF (axis_number = 1) THEN
    vec := u1;
  END_IF;
  IF (axis_number = 2) THEN
    vec := u2;
  END_IF;
  IF (axis_number = 3) THEN
    vec := u3;
  END_IF;
  RETURN(vec);
END_FUNCTION;
(*
```

4.5.4.3.2 PLACE AXIS

```
*)
FUNCTION place_axis(dim, axis_number : INTEGER;
                    axis, ref_direction : direction) : direction;
  LOCAL
    vec, u1, u2, u3 : direction;
    factor          : REAL;
  END_LOCAL;
  IF (dim = 3) THEN
```

```
    u3 := third_axis(axis);
    u1 := first_proj_axis(u3,ref_direction);
    u2 := second_axis(u3,u1);
  ELSE
    u3 = NULL;
    IF EXISTS(ref_direction) THEN
      u1 := normalize(ref_direction);
      u2 := orthogonal_complement(u1);
    ELSE
      u1.x := 1.0;
      u1.y := 0.0;
      u2.x := 0.0;
      u2.y := 1.0;
    END_IF;
  END_IF;
  IF (axis_number = 1) THEN
    vec := u1;
  END_IF;
  IF (axis_number = 2) THEN
    vec := u2;
  END_IF;
  IF (axis_number = 3) THEN
    vec := u3;
  END_IF;
  RETURN(vec);
END_FUNCTION;
(*
```

4.5.4.3.3 ORTHOGONAL COMPLEMENT

```
*)
FUNCTION orthogonal_complement(vec : two_space_direction) :
                                          two_space_direction;
  LOCAL
    vec_local : two_space_direction;
  END_LOCAL;
  vec_loc.x := vec.y;
  vec_loc.y := -vec.x;
  RETURN(vec_loc);
END_FUNCTION;
(*
```

4.5.4.3.4 THIRD AXIS

This function returns the normalized value of the input vector or, if the input vector is NULL, the vector (0,0,1).

```
*)
FUNCTION third_axis(arg: three_space_direction)
```

```
                                        : three_space_direction;
  LOCAL
    z_axis : default_z_axis;
  END_LOCAL;
  IF EXISTS(arg) THEN
    RETURN(z_axis);
  ELSE
    RETURN(normalize(arg));
  END_IF;
END_FUNCTION;
(*
```

4.5.4.3.5 FIRST PROJ AXIS

The following function returns the normalized vector that is the projection of ARG onto the plane normal to the vector Z-AXIS. If ARG is NULL then the projection of the vector (1,0,0) onto Z-AXIS is returned.

```
*)
FUNCTION first_proj_axis(z_axis,arg: direction) : direction;
  LOCAL
    x_axis     : direction;
    v          : direction;
  END_LOCAL;
  IF NOT EXISTS(arg) THEN
    v := default_x_axis;
  ELSE
    IF coordinate_space(arg) = 2 THEN
      VIOLATION;
    ELSE
      v := arg;
    END_IF;
  END_IF;
  x_axis := scalar_times_vector(dot_product(v, z_axis), z_axis);
  x_axis := vector_diff(v, x_axis);
  x_axis := normalize(x_axis);
  RETURN(x_axis);
END_FUNCTION;
(*
```

4.5.4.3.6 SECOND AXIS

This function returns the normalized cross product of the two input vectors.

```
*)
FUNCTION second_axis(z_axis,x_axis: direction) : direction;
  LOCAL
    y_axis : direction;
  END_LOCAL;
```

```
    y_axis := normalize(cross_product(z_axis, x_axis));
  RETURN(y_axis);
END_FUNCTION;
(*
```

4.5.4.3.7 SECOND PROJ AXIS

This function returns the normalized vector that is simultaneously the projection of ARG onto the plane normal to the vector Z-AXIS and onto the plane normal to the vector X-AXIS. If ARG is NULL then the projection of the vector (0,1,0) onto Z-AXIS is returned.

```
*)
FUNCTION second_proj_axis(z_axis,x_axis,arg: direction)
                                              : direction;
  LOCAL
    y_axis : direction;
    v      : direction;
    temp   : direction;
  END_LOCAL;
  IF NOT EXISTS(arg) THEN
    v := default_y_axis;
  ELSE
    v := arg;
  END_IF;
  temp := scalar_times_vector(dot_product(v, z_axis), z_axis);
  y_axis := vector_diff(v, temp);
  temp := scalar_times_vector(dot_product(v, x_axis), x_axis);
  y_axis := vector_diff(y_axis, temp);
  y_axis := normalize(y_axis);
  RETURN(y_axis);
END_FUNCTION;
(*
```

4.5.4.4 BASE ORIGIN

```
*)
FUNCTION base_origin(dim : INTEGER;
                     origin : cartesian_point) : cartesian_point;
  LOCAL
    point : cartesian_point;
  END_LOCAL;
  IF NOT EXISTS(origin) THEN
    point.x_coordinate := 0.0;
    point.y_coordinate := 0.0;
    IF (dim = 3) THEN
      point.z_coordinate := 0.0;
    END_IF;
  ELSE
    point = origin;
```

```
  END_IF;
  RETURN(point);
END_FUNCTION;
(*
```

4.5.4.5 BASE SCALE

```
*)
FUNCTION base_scale(scale : REAL) : REAL;
  LOCAL
    scl : REAL;
  END_LOCAL;
  IF NOT EXISTS(scale) THEN
    scl := 1.0;
  ELSE
    scl = scale;
  END_IF;
  RETURN(scl);
END_FUNCTION;
*)
```

4.5.4.6 PARAMETRIC LOWER LIMIT

```
*)
FUNCTION parametric_lower_limit(arg: GENERIC): list_of_real;
LOCAL
  k : INTEGER;
  parametric_coords : list_of_real;
END_LOCAL;
  k := dimensionality(arg);
  (* pseudo code for K lower parametric limit(s)
     and put into PARAMETRIC_COORDS *)
RETURN(parametric_coords);
END_FUNCTION;
(*
```

returns the minimum parameter value(s) of its argument, which must be decomposable to a geometric entity.

4.5.4.7 PARAMETRIC UPPER LIMIT

```
*)
FUNCTION parametric_upper_limit(arg: GENERIC): list_of_real;
LOCAL
  k : INTEGER;
  parametric_coords : list_of_real;
END_LOCAL;
  k := dimensionality(arg);
  (* pseudo code for K upper parametric limit(s)
```

```
      and put into PARAMETRIC_COORDS *)
  RETURN(parametric_coords);
END_FUNCTION;
(*
```

returns the maximum parameter value(s) of its argument, which must be decomposable to a geometric entity.

4.5.4.8 SPACE DIMENSION

```
*)
FUNCTION space_dimension(origin : cartesian_point;
                         e1, e2, e3 : direction) : INTEGER;
  LOCAL
    dim : INTEGER;
  END_LOCAL;
  dim := 3;
  IF (coordinate_space(origin) = 2 OR
      coordinate_space(e1) = 2 OR
      coordinate_space(e2) = 2 OR
      coordinate_space(e3) = 2) THEN
    dim := 2;
  END_IF;
  RETURN(dim);
END_FUNCTION;
(*
```

4.5.4.9 VECTOR OPERATIONS

This Section defines various functions that operate on vectors.

4.5.4.9.1 CROSS PRODUCT

```
*)
FUNCTION cross_product (vec1,vec2: vector): vector;
  IF coordinate_space(vec1) = 2 OR
     coordinate_space(vec2) = 2 THEN
    VIOLATION;
  END_IF;
END_FUNCTION;
(*
```

returns the vector that is the cross product of the two input vectors, which must be of the same length (C = A * B). If the two vectors are either parallel or anti-parallel a NULL vector is returned.

Note: This function will give inconsistent results with vector with magnitude.

4.5.4.9.2 DOT PRODUCT

```
*)
```

```
FUNCTION dot_product (vec1,vec2: vector): REAL;
  LOCAL
    scalar : REAL;
  END_LOCAL;
  IF coordinate_space(vec1) <> coordinate_space(vec2) THEN
    VIOLATION;
  ELSE
    scalar := 0.0;
    REPEAT i := 1 TO coordinate_space(vec1);
      scalar := scalar + vec1[i]*vec2[i];
    END_REPEAT;
  RETURN(scalar);
  END_IF;
END_FUNCTION;
(*
```

This function returns the dot (scalar) product of the two input vectors, which must be of the same length. That is A.B = $AB\cos\theta$, where θ is the angle between the two vectors. If the vectors are perpendicular then A.B = 0.0.

Note: This function will give inconsistent results with **vector with magnitude.**

4.5.4.9.3 NORMALIZE

This function returns the vector that is the normalization of the input vector.

```
*)
FUNCTION normalize(v: vector): vector;
  LOCAL
    i,dim,magv,size : REAL;
    result          : vector;
  END_LOCAL;
    dim := coordinate_space(v);
    IF vector_with_magnitude IN TYPEOF(v) THEN
      result.magnitude := 1.0;
    END_IF;
    size := 0.0;
    REPEAT i := 1 TO dim;
      size := size + v[i]*v[i];
    END_REPEAT;
    size := SQRT(size);
    IF size <> 0.0 THEN
      BEGIN
        REPEAT i := 1 TO dim;
          result[i] := v[i]/size;
        END_REPEAT;
      END;
    ELSE
      VIOLATION;
    END_IF;
```

```
  RETURN(result);
END_FUNCTION;
(*
```

4.5.4.9.4 SCALAR TIMES VECTOR

This function returns the vector that is the scalar multiple if the input vector.

```
*)
FUNCTION scalar_times_vector(scalar: NUMBER; vec: vector): vector;
  LOCAL
    output_vector : vector;
    i             : INTEGER;
  END_LOCAL;
  REPEAT i := 1 TO coordinate_space(vec);
    output_vector[i] := scalar*vec[i];
  END_REPEAT;
  RETURN(output_vector);
END_FUNCTION;
(*
```

Note: This function will give inconsistent results with vector with magnitude.

4.5.4.9.5 VECTOR SUM

This function returns the sum of the input vectors.

```
*)
FUNCTION vector_sum(arg1,arg2: vector): vector;
  LOCAL
      temp : vector;
      result : vector;
  END_LOCAL;
  IF TYPEOF(arg1) = vector_with_magnitude THEN
    REPEAT i := 1 TO (SIZEOF(arg1)-1);
      temp[i] := arg1[i] + arg2[i];
      result[i] := temp[i];
    END_REPEAT;
    result[SIZEOF(arg1)] := vector_magnitude(temp);
  ELSE
    REPEAT i := 1 TO SIZEOF(arg1);
      result[i] := arg1[i] + arg2[i];
    END_REPEAT;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

Note: This function will give inconsistent results with vector with magnitude.

4.5.4.9.6 VECTOR_DIFF

This function returns the difference of the input vectors as (ARG1 - ARG2).

```
*)
FUNCTION vector_diff(arg1,arg2: vector): vector;
  LOCAL
    temp : vector;
    result : vector;
  END_LOCAL;
  IF TYPEOF(arg1) = vector_with_magnitude THEN
    REPEAT i := 1 TO (SIZEOF(arg1)-1);
      temp[i] := arg1[i] - arg2[i];
      result[i] := temp[i];
    END_REPEAT;
    result[SIZEOF(arg1)] := vector_magnitude(temp);
  ELSE
    REPEAT i := 1 TO SIZEOF(arg1);
      result[i] := arg1[i] - arg2[i];
    END_REPEAT;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

Note: This function will give inconsistent results with **vector with magnitude.**

4.5.4.9.7 VECTOR MAGNITUDE

This function returns the magnitude of the input vector.

```
*)
FUNCTION vector_magnitude(v: vector): REAL;
  LOCAL
    dim,size : REAL;
    i        : INTEGER;
  END_LOCAL;
  dim := coordinate_space(v);
  size := 0.0;
  IF vector_with_magnitude IN TYPEOF(v) THEN
    size := v.magnitude:
  ELSE
    REPEAT i := 1 TO dim;
      size := size + v[i]*v[i];
    END_REPEAT;
    size := SQRT(size);
  END_IF;
  RETURN(size);
END_FUNCTION;
(*
```

4.5.4.10 VOLUME_EXTENT

This function returns the volume of the solid parameter.

```
*)
FUNCTION volume_extent(arg : solid_model) : REAL;
END_FUNCTION;
(*
```

4.5.5 Geometry Classification Structure

The following indented list provides the geometry classification structure.

```
GEOMETRY
  AXIS PLACEMENT
    AXIS1 PLACEMENT
    AXIS2 PLACEMENT
  CURVE
    BOUNDED CURVE
      B-SPLINE CURVE
      COMPOSITE CURVE
      POLYLINE
      TRIMMED CURVE
    CONIC
      CIRCLE
      ELLIPSE
      HYPERBOLA
      PARABOLA
    CURVE ON SURFACE
      COMPOSITE CURVE ON SURFACE
      INTERSECTION CURVE
      PCURVE
      SURFACE CURVE
    LINE
    OFFSET CURVE
      D2 OFFSET CURVE
      D3 OFFSET CURVE
  POINT
    CARTESIAN POINT
    POINT ON CURVE
    POINT ON SURFACE
  SURFACE
    BOUNDED SURFACE
      B-SPLINE SURFACE
      CURVE BOUNDED SURFACE
      RECTANGULAR TRIMMED SURFACE
      RECTANGULAR COMPOSITE SURFACE
    ELEMENTARY SURFACE
      CONICAL SURFACE
      CYLINDRICAL SURFACE
```

```
      PLANE
      SPHERICAL SURFACE
      TOROIDAL SURFACE
    OFFSET SURFACE
    SWEPT SURFACE
      SURFACE OF LINEAR EXTRUSION
      SURFACE OF REVOLUTION
  TRANSFORMATION
  VECTOR
    DIRECTION
        DEFAULT X AXIS
        DEFAULT Y AXIS
        DEFAULT Z AXIS
    VECTOR WITH MAGNITUDE
CARTESIAN THREE COORDINATE
CARTESIAN TWO COORDINATE
THREE SPACE DIRECTION
TWO SPACE DIRECTION

*)
END_SCHEMA; -- end GEOMETRY schema
(*
```

4.6 Topology

4.6.1 Topology Introduction

This Section contains the Topology Integrated Product Information Model.

```
*)
SCHEMA ipim_topology_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_geometry_schema);
(*
```

In mechanical CAD systems the role of topology has been traditionally limited to its use in defining Boundary Representation (Brep) Solid Models. Topology, though, is used in other types of CAD systems, for example electrical CAE, to capture logical connections between system components such as those electronic components and their interconnections forming an electronic assembly.

The topological entities, vertex, edge etc., specified here have been defined independently of any "use" that may be made of them, either use by higher level topological entities or use by a CAD system. Minimal constraints have been placed on each entity with the intention that any additional constraints will be specified by the using entity or by a defined context in which the entity is used. The intent is to avoid limiting the context or the use made of the entities.

The topological entities have been defined in a hierarchical manner with the vertex being the primitive entity. That is, all other topological entities are defined either directly or indirectly in terms of vertices.

Each entity has its own set of constraints. A higher level entity may impose constraints on a lower level entity. At the higher level, the constraints on the lower level entity are the sum of the constraints imposed by each entity in the chain between the higher and lower level entities.

4.6.1.1 Nomenclature and Symbology

An attempt has been made to define precisely the constraints that must be met by the topological entities. In many cases these are defined symbolically. This Section describes the notation used for this purpose. It should be noted that the definitions given here are independent of EXPRESS definitions and usage.

TOPOLOGICAL ENTITIES The topological entities are **vertex**, **edge**, **path**, **loop**, **face** (and **subface**), **shell** and **region**. These will be referred to by the following symbols V, E, P, L, F, S and R, repectively.

Some of these entities take particular forms and a superscript is used to distinguish between these forms if necessary. A **loop**, for example, may be a **vertex loop**, an **edge loop** or a **polyloop**. These forms are denoted as L^v, L^e, L^p. Table 11 lists the forms used in this section.

In some instances of the entity definitions, a topological attribute may take the form of a (topological + logical) pair. A subscript is used to distinguish between the topological and the (topological + logical) pairing. For example, E and E_l or S° and S°_l.

SET A Set is an unordered homogeneous collection with no duplicate members. A set is represented by an enclosing pair of braces. Thus, $\{A\}$ is a set of entities of type A.

Symbol	Definition
V	Vertex
$\mathcal{V}$	Number of unique vertices
E	Edge
$\mathcal{E}$	Number of unique edges
G^e	Edge genus
P	Path
$\mathcal{P}$	Number of unique paths
G^p	Path genus
L	Loop
$\mathcal{L}$	Number of unique loops
L^e	Edge-loop
L^p	Poly-loop
L^v	Vertex-loop
G^l	Loop genus
F	Face
$\mathcal{F}$	Number of unique faces
G^f	Face genus
S	Shell
$\mathcal{S}$	Number of unique shells
S^c	Closed-shell
S^o	Open-shell
S^v	Vertex-shell
S^w	Wire-shell
G^s	Shell genus
R	Region
$\mathcal{R}$	Number of unique regions
G	Genus
$\mathcal{G}$	Sum of genus

Table 11: Topology Symbol Definitions

LIST A List is an unordered homogenous collection with possibly duplicate members. A list is represented by an enclosing pair of brackets. Thus, $[A]$ is a list of entities of type A.

OWNER It is useful to be able to define a list or a set whose members are "owned" by another entity; for example, the list of faces comprising a shell. An Owner is denoted by enclosing it in parentheses. Thus $(S)[F]$ is the list of faces comprising the shell S. There can be a chain of owners; $((S)\{F\})[L]$ is the list of loops of the set of faces belonging to the shell S.

SIZE OF Enclosing a set or list in a pair of vertical lines denotes the size (number of members) of the set or list. Thus $|(S)[F]|$ is the number of unique faces comprising S.

SET and LIST EQUALITY Two sets or two lists or a set and a list are said to be equal if they have exactly the same members, irrespective of order. Thus, $(S)\{F\} = (S)[F]$ says that the set of faces of S and the list of faces of S are the same (i.e each face in S is unique). By definition, $(S)\{F\} \neq (S)\{L\}$ is always TRUE as Face and Loop are different types.

Several topological entities use an Orientation Flag to indicate whether the direction of a referenced entity agrees with or is opposed to the direction of the referencing entity. If the Flag is TRUE then the direction of the referenced entity is correct but if the Flag is FALSE then the direction of the referenced entity should be (conceptually) reversed. It can happen that there are several Orientation Flags in the chain of entities from the high level referencing entity to the low level referenced entity. The direction of a low level entity with respect to a high level entity is obtained by evaluating the *not exclusive or* ($\odot$) of the chain of Orientation Flags. For example, a Face references a Loop + Loopflag, a Loop references an Edge + Edgeflag and an Edge references a Curve + Curveflag. The Face "FaceCurveflag" is given by

$$\text{FaceCurveflag} = \text{Loopflag} \odot \text{Edgeflag} \odot \text{Curveflag}$$

where *not exclusive or* is defined by:

$$\begin{aligned} T \odot T &= T \\ T \odot F &= F \\ F \odot F &= T. \end{aligned}$$

Thus

$$F \odot T \odot F = T$$

4.6.1.2 Graph Traversal

A connected graph can be completely traversed, starting and ending at the same vertex, such that every edge is traversed exactly twice, once in each direction, and every vertex is "passed through" the same number of times as there are edges using the vertex. If an (edge + edge traversal direction) is considered as a unit then each unique (edge + direction) combination must occur once and only once in the traversal of a graph. During the traversal of a graph it will be found that there are one or more sets of alternating vertices and (edge + direction) units that form closed cycles. A closed cycle is called a *loop*. Every edge in the graph is used by at least one and at most two loops. Every (edge + direction) combination is used by one and only one loop. The *Genus* of a connected graph, G^g, is the number of closed cycles of vertices and edges in the graph. The number of loops in a general graph is given by $M \leq L \leq (M + G^g)$ where M is the multiplicity (the number of connected graphs in the graph) and L is the number of Loops.

The following *maze traversal* algorithm (P. R. Wilson, " Euler Formulas and Geometric Modeling," IEEE Computer Graphics & Applications, Vol 5, No 8, pp 24-36, August 1985) may be used to traverse a graph and to determine the genus of the graph and, if the graph is disconnected, the number of connected sub-graphs.

1. Set the genus and connected graph counters to zero.
2. Start at any (unvisited) vertex. If there is no unvisited vertex then **STOP**. Mark the vertex as *visited*. Increment the connected graph counter. Traverse any edge at the vertex, marking the edge with the travel direction.
3. After traversing an edge PQ to reach the vertex Q, do the following:
   - When reaching a vertex for the first time, mark the edge just traveled as the *advent edge* of the vertex.
   - If this is the first traversal of the edge and the vertex Q has previously been visited, increment the genus counter.
   - Select an exit edge from the vertex according to the following rules:
     (a) No edge may be selected that has been previously been traversed in the direction away from the vertex Q.
     (b) Select any edge, except the advent edge of Q, that meets rule (a).
     (c) Select the advent edge of Q, provided it meets rule (a).
   - Traverse the selected exit edge and mark it with the travel direction.
4. If no edge was selected in the previous step, then go to step 2.

At the end of this algorithm the number of connected graphs are known as well as the genus.
A graph satisfies the following form of the Euler equation

$$\chi_w = (\mathcal{V} - \mathcal{E}) - (M - \mathcal{G}^g) = 0 \qquad (1)$$

where $\mathcal{V}$ and $\mathcal{E}$ are the number of unique vertices and edges in the graph. M is the multiplicity (the number of connected graphs in the graph) of the graph and $\mathcal{G}^g$ is the sum of the genus of the connected component graphs.

4.6.1.3 DEFINITIONS

A number of informal definitions are given here which will later be used to describe and constrain the topological entities.

ARCWISE CONNECTED An entity is arcwise connected if any two arbitrary points in its domain can be connected by an arc within the domain that does not cross a boundary of the entity.

BOUNDARY A boundary defines the limits of a domain. An domain may or may not include its boundaries. An entity of dimensionality D has a boundary of dimensionality d, where $0 \leq d < D$.

CLOSED An entity of dimensionality D is closed if it divides R^{D+1} into exactly two regions.

DIMENSIONALITY The number of parameters required to specify the geometric form of a topological entity. Edges (curves), Faces (surfaces) and volumes have dimensionality of 1, 2 and 3 respectively. By convention a Vertex (point) has dimensionality of 0.

DOMAIN The space (region) over which an entity exists (is defined).

EXTENT The measure of the content of the region occupied by an entity (i.e its domain), measured in units appropriate to the dimensionality of the entity, thus length, area and volume for dimensionalities of 1, 2 and 3. Where necessary the symbol Ξ will be used to denote extent.

GRAPH A set of *vertices* and *edges*. The edges of a graph do not intersect except at their boundaries (i.e vertices). A graph is disconnected if it is not arcwise connected.

MANIFOLD Let m and n be non-negative integers and X be a subset of R^m. Let X be given the relative topology inherited from R^m. Then X is said to be a topological $n-$manifold provided each point of X has an open neighbourhood that is homeomorphic to the open unit ball of R^n.

MULTIGRAPH A (sub-) graph where two or more edges have the same boundaries (vertex pairs).

OVERLAP A domain of dimensionality D overlaps if at every point of the domain an open ball of diameter ϵ, where $\epsilon \rightarrow 0$, in R^{D+1} can be drawn that is partitioned into exactly two regions, and two or more points in the domain correspond to the same point in R^3.

PSEUDOGRAPH A graph that contains self-loops and/or multigraphs.

R^3 Three dimensional real space.

SELF-INTERSECTION A domain self-intersects if it is neither manifold nor overlapping.

SELF-LOOP An edge that has the same boundary (vertex) at either end.

4.6.2 TOPOLOGY

```
*)
ENTITY topology
  SUPERTYPE OF (vertex XOR
                edge XOR
                path XOR
                loop XOR
                face XOR
                subface XOR
                shell XOR
                connected_edge_set XOR
                connected_face_set XOR
                region);
END_ENTITY;
(*
```

4.6.2.1 VERTEX

A vertex is a unique (topological) point in R^3 with dimensionality 0 and extent 0. A geometric point may be associated with a vertex to locate it in a coordinate space.

```
*)
ENTITY vertex
  SUBTYPE OF (topology);
```

```
  vertex_point : OPTIONAL point;
WHERE
  dimensionality(vertex) = 0;
  extent(vertex) = 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

vertex_point: Optional geometric point associated with vertex.

```
*)
RULE vertex_geometry FOR (vertex);
  IF ((vertex_point <> NULL) AND
      (domain(vertex) <> domain(vertex_point))) THEN
    VIOLATION;
  END_IF;
END_RULE;
(*
```

PROPOSITIONS:

1. The domain of the vertex is defined by the domain of the point, if it is specified.

4.6.2.2 EDGE

An edge is an arcwise connected bounded directed finite manifold (topological) curve in R^3 with dimensionality 1. The bounds of an edge are two vertices. The edge is positively directed from the first to the second vertex. An edge may form a self-loop. The domain of the edge does not include its bounds and $0 < \Xi < \infty$.

Associated with an edge may be a geometric curve to locate the edge in a coordinate space. The curve must be manifold within the domain of the edge. As the topological and geometric directions may be opposed, an indicator is used to identify whether the edge and curve directions agree or are opposed. The logical value indicates whether the curve direction agrees with (TRUE) or is in the opposite direction (FALSE) to the edge direction. Any geometry associated with the vertices of the edge must be consistent with the edge geometry.

An edge must satisfy the Euler equation (1) with $\mathcal{E} = 1, M = 1$. That is

$$\chi_e = \mathcal{V} - (2 - G^e) = 0 \tag{2}$$

where $\mathcal{V}$ is the number of unique vertices of the edge and G^e is the genus of the edge.

The genus of the edge may be determined by traversing the edge from the start vertex to the end vertex. The genus equals the number of times the first traversal of an edge leads to a vertex that has already been used in the traversal.

Specifically, the topological edge defining data must satisfy:

- An edge has two vertices,

$$|E[V]| = 2$$

- The vertices need not be distinct,

$$1 \leq |E\{V\}| \leq 2$$

- Equation 2 must hold

$$|E\{V\}| - 2 + G^e = 0$$

```
*)
ENTITY edge
  SUBTYPE OF (topology);
  edge_start    : vertex;
  edge_end      : vertex;
  edge_curve    : OPTIONAL curve_logical_structure;
WHERE
  dimensionality(edge) = 1;
  {0 < extent(edge) < infinity};
    (* Extent is finite and non-zero *)
  manifold(edge);
  arcwise_connected(edge);
  intersect(edge_start, domain(edge)) = NULL;
  intersect(edge_end, domain(edge)) = NULL;
    (* An edge domain does not include its boundaries *)
  {0 <= genus(edge) <= 1};
  (edge_start = edge_end) OR (edge_start <> edge_end);
  constraints_geometry_edge(edge);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

edge_start: Start point (vertex) of the edge.

edge_end: End point (vertex) of the edge.

edge_curve: Optional curve logical structure which associates geometric curve information with edge.

```
*)
RULE edge_geometry FOR (edge);
  REPEAT FOR EACH edge IN MODEL;
    IF NOT UNIQUE domain(edge) THEN
      VIOLATION;
    END_IF;
  END_REPEAT;
    (* Multiple EDGEs can reference the same curve,
       but if they do they must reference different
       geometric curve domains *)
END_RULE;
(*
```

PROPOSITIONS:

1. The same vertex can be used for both edge start and edge end.

2. Extent of an edge is finite and non-zero.

3. An edge is arcwise connected.

4. Vertex geometry must be consistent with edge geometry.

5. The function constraints geometry edge returns TRUE where the function evaluates the geometric constraints.

6. Multiple edges can reference the same curve, but if they do they must have different domains.

4.6.2.3 CURVE LOGICAL STRUCTURE

This entity associates a LOGICAL value with a curve.

```
*)
ENTITY curve_logical_structure;
  curve_element : curve;
  flag          : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

curve_element: Geometric curve.

flag: LOGICAL value associated with curve.

4.6.2.4 PATH

A **path** is a piecewise open oriented arcwise connected manifold curve in R^3 of dimensionality 1 and genus zero. The domain of the path does not include its bounds and $0 < \Xi < \infty$.

In graph terms a path is an ordered list of (N+1) unique vertices and N (edge + direction) components, where $N \geq 1$. The first and last vertices in the list are different. An edge can only appear once in the list and must not be a self-loop.

A path is represented by an ordered collection of (edge + logical) pairs as the vertices may be determined by examining the edge data. The path is ordered from the first vertex in the path to the last. The LOGICAL value is used to indicate whether the Edge direction agrees with the direction of the Path (TRUE) or is in the opposite direction (FALSE).

The genus of a Path can be determined by traversing the edges of the Path, starting at the initial vertex, in the given order. The genus is the number of times the first traversal of an edge leads to a vertex that has been previously met in the traversal.

A path must satisfy the the Euler formula, equation (1), with $M = 1$, $G^p = 0$. That is

$$\chi_p = (\mathcal{V} - \mathcal{E}) - 1 = 0 \tag{3}$$

where $\mathcal{V}$ and $\mathcal{E}$ are the number of unique vertices and edges in the path. Specifically, the topological attributes of a Path must meet the following constraints

- The edges in the Path are unique,

$$(P)[E] = (P)\{E\}$$

- In the list $((P)[E])[V]$, two vertices appear once only and every other vertex appears exactly twice.

- The Path genus is zero.
- Equation 3, as

$$|((P)[E])\{V\}| - |(P)\{E\}| - 1 = 0$$

```
*)
ENTITY path
  SUBTYPE OF (topology);
  open_edge_list  : LIST [1 : #] OF UNIQUE edge_logical_structure;
WHERE
  dimensionality (path) = 1;
  (0 < extent(path) < infinity);
  open(path);
  manifold(path);
  arcwise_connected(path);
  genus(path) = 0;
  constraints_topology_path(path);
  constraints_geometry_path(path);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

open_edge_list: List of edge logical structures such that the path does not close on itself.

PROPOSITIONS:

1. A path is manifold and arcwise connected.
2. There must be at least one edge in the path.
3. An individual edge can only be referenced once by an individual path.
4. An edge can be referenced by multiple paths.
5. An edge can exist independently of a path.
6. The function constraints topology path returns TRUE, where the function evaluates the topological constraints on a path.
7. The function constraints geometry path returns TRUE where the function evaluates the geometric constraints on a path.

4.6.2.5 EDGE LOGICAL STRUCTURE

This entity associates a LOGICAL value with an edge.

```
*)
ENTITY edge_logical_structure;
  edge_element : edge;
  flag : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

edge_element: Edge entity.

flag: A LOGICAL value associated with the edge.

4.6.2.6 LOOP

A loop is a piecewise closed oriented arcwise connected non self-intersecting curve in R^3 with dimensionality 0 or 1. As the curve is closed the location of the bound (closure) on the curve is arbitrary. The domain of the loop includes the bound and $0 \leq \Xi < \infty$.

In graph terms a loop is an ordered list of (N+1) vertices and N (edge + direction) components, where $N \geq 0$ and the first and last vertices in the list are the same. When N=0, the loop has dimensionality 0, otherwise it has dimensionality 1.

A loop must satisfy the Euler formula, equation (1), with $M = 1$. That is

$$\chi_l = (\mathcal{V} - \mathcal{E}) - (1 - G^l) = 0 \tag{4}$$

where $\mathcal{V}$ and $\mathcal{E}$ are the number of unique vertices and edges in the loop and G^l is the genus of the loop.

The loop genus can be determined by traversing the edges of the loop, starting at any vertex, in the order given. The loop genus equals the number of times the first traversal of an edge leads to a vertex that has been met before in the traversal.

A loop is represented by a single vertex or by an ordered collection of (edge + logical) pairs or by an ordered collection of points.

```
*)
ENTITY loop
  SUPERTYPE OF (vertex_loop XOR
                edge_loop XOR
                poly_loop)
  SUBTYPE OF (topology);
WHERE
  (0 <= dimensionality(loop) <= 1);
  (0 <= extent(loop) < infinity);
  closed(loop);
  genus(loop) >= 0;
END_ENTITY;
(*
```

PROPOSITIONS:

1. A loop has dimensionality less than or equal to one.
2. A loop has a non-infinite extent.
3. A loop is arcwise connected.
4. A loop describes a closed (topological) curve.

4.6.2.7 VERTEX LOOP

A **vertex loop** is a **loop** of zero genus consisting of a single **vertex**.
The topological data must satisfy the following constraints:

- The loop consists of a single vertex

$$|(L)\{V\}| = |(L)[V]| = 1$$

- Equation 4 must be satisfied

$$|(L)\{V\}| - 1 = 0$$

```
*)
ENTITY vertex_loop
  SUBTYPE OF (loop);
  loop_vertex   : vertex;
UNIQUE
  loop_vertex;
WHERE
  dimensionality(vertex_loop) = 0;
  extent(vertex_loop) = 0;
  genus(vertex_loop) = 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

loop_vertex: The unique vertex which composes the entire loop.

PROPOSITIONS:

1. Loop vertex must be UNIQUE.
2. A vertex loop has zero extent and dimensionality.
3. A vertex can be used by no more than one vertex loop.
4. A vertex can exist independently of a vertex loop.

4.6.2.8 EDGE LOOP

An **edge loop** is a **loop** with non-zero extent. It is represented by an ordered collection of (edge + logical) pairs such that adjacent edges in the loop are adjacent in the collection. The vertices of the loop may be determined by examining the edge data. The logical indicator is used to identify whether the positive edge direction agrees with (TRUE) or is opposed to (FALSE) the positive direction of the loop.
The defining topological entities of an Edge Loop must conform to the following constraints:

- Each (edge + logical) pair must be unique.

$$(L^e)\{E_l\} = (L^e)[E_l]$$

- The number of vertices in the list of Loop vertices must be twice the number of (edge + logical) pairs.

$$2|(L^e)[E_l]| - |((L^e)[E_l])[V]| = 0$$

This condition ensures that the Loop is closed but allows the graph to have branches and edges to be self-loops.

- The genus of the loop must be greater than zero.
- Equation 4 must be satisfied

$$|((L\{E\})\{V\}| - |(L)\{E\}| - 1 + G^l = 0$$

```
*)
ENTITY edge_loop
  SUBTYPE OF (loop);
  loop_edges : LIST [1 : #] OF UNIQUE edge_logical_structure;
WHERE
  dimensionality(edge_loop) = 1;
  {0 < extent(edge_loop) < infinity};
  arcwise_connected(edge_loop);
  manifold(edge_loop) or overlap(edge_loop);
  genus(edge_loop) >= 1;
  constraints_topology_edge_loop(edge_loop);
  constraints_geometry_edge_loop(edge_loop);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

loop_edges: List of EDGE/LOGICAL pairs.

PROPOSITIONS:

1. An **edge loop** has finite extent.
2. An **edge loop** has dimensionality 1.
3. An **edge loop** may overlap itself.
4. An **edge loop** is arcwise connected.
5. An **edge** may not be referenced more than twice by a single **edge loop**.
6. Each Edge/Logical combination in an **edge loop** must be unique.
7. The function **constraints topology edge loop** returns TRUE after evaluating the topological constraints.
8. The function **constraints geometry edge loop** returns TRUE after evaluating the geometric constraints.
9. An **edge** can exist independently of an **edge loop**.
10. An **edge** can be referenced by multiple **edge loops**.

4.6.2.9 POLY LOOP

A **poly loop** is a **loop** of genus 1 where the loop is represented by an ordered co-planar collection of **points** forming the vertices of the loop. The loop is composed of straight line segments joining a point in the collection to the succeeding point in the collection. The closing segment is from the last to the first point in the collection. The direction of the loop is in the direction of the line segments.

A **poly loop** must conform to the following topological constraints:

- The loop has a genus of one.
- Equation 4 must be satisfied

$$|(L)\{V\}| - |(L)\{E\}| = 0$$

```
*)
ENTITY poly_loop
  SUBTYPE OF (loop);
  polygon : LIST [3 : #] OF UNIQUE point;
WHERE
  dimensionality(poly_loop) = 1;
  {0 < extent(poly_loop) < infinity};
  manifold(poly_loop);
  arcwise_connected(poly_loop);
  planar(poly_loop);
  genus(poly_loop) = 1;
  constraints_topology_loop(poly_loop);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

polygon: List of **points** defining the loop. There are no repeated **points** in the list.

PROPOSITIONS:

1. A **poly loop** has finite extent.
2. A **poly loop** has dimensionality 1.
3. A **poly loop** is manifold and arcwise connected.
4. All **points** in a **poly loop** must be co-planar.
5. The function **constraints topology loop** returns TRUE where the function evaluates the topological constraints on the **loop**.
6. A **point** can exist independently of a **poly loop**.

4.6.2.10 FACE

A **face** is an oriented bounded arcwise connected manifold (topological) surface in R^3 of dimensionality 2. A face need not be simply connected, that is, it may have holes in it. The face domain does not include its bounds and $0 < \Xi < \infty$. The domain must be mappable to a plane. A face must have

at least one bound, and the bounds must be disjoint. A bound must have dimensionality of either 0 or 1.

A face is represented by embedding a graph in a (topologically) plane surface. The graph will partition the plane into one or more regions, one of which will have semi-infinite extent, the others, if any, will be finite. Each region in the plane is a face. The boundary of each face is comprised of one of more loops which define the domain of the face. A face has a topological normal **n** and the tangent to a loop is **t**. For a loop defining the bound of a face the cross-product **n**X**t** points towards the interior of the face. In other words, when a point in the face and near a boundary is viewed in the opposite direction to the face normal, the loop direction is locally counter clockwise. An indicator is used to signify whether the loop direction is oriented correctly with respect to the face normal (TRUE) or it should be reversed (FALSE).

As a face is mappable to a plane, it is possible in principle, to determine the "outer" loop of the face. All other loops are contained within the outer loop. An inner loop must not contain any other loop of the face.

A geometric surface, which must be manifold and mappable to a plane within the domain of the face, may be associated with the face. (Note that a spherical surface with a single boundary point is mappable to a plane). As both a face and a geometric surface have defined normal directions an indicator is used to indicate whether the surface normal agrees with (TRUE) or is opposed to (FALSE) the face normal direction. The geometry associated with any component of the loops of the face must be consistent with the surface geometry.

The genus (number of holes) of a face can be determined in the following manner from the graph of edges and vertices defining the bounds of the face:

- Delete all edges from the graph that are referenced twice by the loops of the face.
- Delete all vertices that have no associated edges.
- The number of holes in the face is the maximum of 0 and (M-1), where M is the multiplicity of the resulting graph.

Alternatively, the genus of a face is one less than the sum of the genus of the loops of the face. That is,

$$G^f = \sum_{i=1}^{L} G_i^l - 1$$

The Euler formula, equation (1) with $L = M$, must be satisfied. That is,

$$\chi_f = (\mathcal{V} - \mathcal{E}) - (\mathcal{L} - \mathcal{G}^l) = 0 \tag{5}$$

or

$$\chi_f = (\mathcal{V} - \mathcal{E}) - \mathcal{L} + (1 + G^f) = 0$$

where $\mathcal{V}, \mathcal{E}$ and $\mathcal{L}$ are the numbers of unique vertices, edges and loops in the face, $\mathcal{G}^l$ is the sum of the genus of the loops and G^f is the genus of the face.

More specifically, the following topological constraints must be met:

- The loops are unique
$$(F)\{L\} = (F)[L]$$
- In the list $((F)[L])[E]$ an individual edge occurs no more than twice.

- Each (edge + logical) pair is unique

$$((F)[L])\{E_i\} = ((F)[L])[E_i]$$

- Equation 5 must be satisfied

$$|(((F)[L^e])\{E\})\{V\}| + |((F)[L^v])\{V\}| - |((F)[L])\{E\}| - |(F)[L]| + \sum G^i = 0$$

```
*)
ENTITY face
  SUBTYPE OF (topology);
  outer_bound  : OPTIONAL loop_logical_structure;
  bounds       : SET [1 : #] OF loop_logical_structure;
  face_surface : OPTIONAL surface_logical_structure;
WHERE
  dimensionality(face) = 2;
  (0 < extent(face) < infinity);
  manifold(domain(face));
  arcwise_connected(face);
  genus(face) >= 0;
  constraints_topology_face(face);
  constraints_geometry_face(face);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

outer_bound: Outside boundary of face. Optional.

bounds: List of all Boundaries of the face.

face_surface: Geometric surface underlying face.

```
*)
RULE face_geometry FOR (face);
  IF EXISTS(outer_bound) AND NOT (outer_bound IN bounds) THEN
    VIOLATION;
  END_IF;
  REPEAT FOR EACH face IN MODEL;
    IF (NOT UNIQUE domain(face)) THEN
      VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. Poly loops may not be mixed with other loop types.
2. All poly loops must be co-planar with each other and the surface geometry.

3. There are no repeated loops in a face.
4. Each loop in a face is unique.
5. One loop (the outer) contains all the other loops.
6. No loop, except the outer can contain another loop of the face.
7. The loops of the face are disconnected.
8. A surface may be used by more than one face.
9. Lower level geometry (vertex points and edge curves) must be consistent with the face geometry.
10. The function constraints topology face returns TRUE, where the function evaluates the face topological constraints.
11. The function constraints geometry face returns TRUE, where the function evaluates the face geometric constraints.

4.6.2.11 LOOP LOGICAL STRUCTURE

This entity associates a LOGICAL value with a loop.

```
*)
ENTITY loop_logical_structure;
  loop_element : loop;
  flag         : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

loop_element: A loop.

flag: A LOGICAL value associated with the loop.

4.6.2.12 SURFACE LOGICAL STRUCTURE

This entity associates a surface and a LOGICAL.

```
*)
ENTITY surface_logical_structure;
  surface_element : surface;
  flag            : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

surface_element: A surface.

flag: A LOGICAL value associated with the surface.

4.6.2.13 SUBFACE

A subface is a portion of the domain of a face, or another subface.

The topological constraints on a subface are the same as on a face.

```
*)
ENTITY subface
  SUBTYPE OF (topology);
  outer_bound : loop_logical_structure;
  bounds      : SET [1 : #] OF loop_logical_structure;
  trimming    : select_face_or_subface;
WHERE
  outer_bound IN bounds;
  embedded(subface, trimming);
  constraints_topology_face(subface);
  constraints_geometry_subface(subface);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

outer_bound: The outside boundary of the subface.

bounds: The list of all the boundaries of the subface.

trimming: The face or subface which contains the subface.

PROPOSITIONS:

1. The outer bound of the subface is contained in the list of all the bounds.
2. The domain of the subface is a subset of the domain of the face or subface being trimmed.
3. All the topological constraints on a face apply to a subface.
4. The type of loops in the subface must match the type of loops in the trimming entity.
5. The function constraints geometry subface returns TRUE, where the function evaluates the subface geometric constraints.

4.6.2.14 SHELL

A shell is a piecewise arcwise connected entity of fixed dimensionality D, where $0 \leq D \leq 2$. The extent of a shell includes its boundaries and $0 \leq \Xi < \infty$.

A shell of dimensionality 0 is represented by a graph consisting of a single vertex. The vertex must not have any associated edges.

A shell of dimensionality 1 is represented by a connected graph of dimensionality 1.

An open shell of dimensionality 2 is a piecewise arcwise connected manifold oriented finite (topological) surface that is mappable to a plane. It is represented by a set of faces.

A closed shell of dimensionality 2 is a piecewise arcwise connected closed oriented manifold (topological) surface mappable to an n-fold torus. The extent includes the boundary, and $0 < \Xi < \infty$. The shell divides R^3 into two regions, one finite and the other infinte. The topological normal of the shell is directed from the finite to the infinite region. It is represented by a set of faces.

```
*)
ENTITY shell
  SUPERTYPE OF (vertex_shell XOR
                wire_shell XOR
                open_shell XOR
                closed_shell)
  SUBTYPE OF (topology);
WHERE
  (0 <= dimensionality(shell) <= 2);
  (0 <= extent(shell) < infinity);
  genus(shell) >= 0;
END_ENTITY;
(*
```

PROPOSITIONS:

1. A shell has a dimensionality between zero and two.
2. A shell has a non-infinite extent.
3. A shell has a genus of zero or more.

4.6.2.15 VERTEX SHELL

A vertex shell is a shell consisting of a single vertex loop.

```
*)
ENTITY vertex_shell
  SUBTYPE OF (shell);
  vertex_shell_boundary  : vertex_loop;
UNIQUE
  vertex_shell_boundary;
WHERE
  dimensionality(vertex_shell) = 0;
  extent(vertex_shell) = 0;
  genus(vertex_shell) = 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

vertex_shell_boundary: Single vertex loop which constitutes the boundary of this type of shell.

PROPOSITIONS:

1. Vertex shell boundary must be UNIQUE.
2. The extent and dimensionality of a vertex shell are both zero.
3. A vertex loop can only be used by a single vertex shell.
4. A vertex loop can exist independently of a vertex shell.

4.6.2.16 WIRE SHELL

A **wire shell** is a **shell** of dimensionality 1. It is represented by a connected graph, specifically by the set of loops forming the graph.

The graph must satisfy the following version of the Euler formula, equation (1)

$$\chi_g = (\mathcal{V} - \mathcal{E}) - (\mathcal{L} - \mathcal{G}^l) = 0 \tag{6}$$

where $\mathcal{V}, \mathcal{E}$ and $\mathcal{L}$ are the numbers of unique vertices, edges and loops in the graph and $\mathcal{G}^l$ is the sum of the genus of the loops.

More specifically, the following topological constraints must be met:

- The loops must be unique

$$(S^w)\{L\} = (S^w)[L]$$

- Each edge must either be referenced by two loops, or twice by a single loop. That is, in the list $((S^w)[L])[E]$ each edge appears exactly twice.

$$|((S^w)[L])[E]| = 2|((S^w)[L])\{E\}|$$

- Each (edge + logical) pair must be unique

$$((S^w)[L])\{E_l\} = ((S^w)[L])[E_l]$$

- Equation 6 must be satisfied

$$|(((S^w)[L])\{E\})\{V\}| - |((S^w)[L])\{E\}| - |(S^w)[L]| + \sum G^l = 0$$

```
*)
ENTITY wire_shell
  SUBTYPE OF (shell);
  wire_shell_boundary  : SET [1 : #] OF edge_loop;
WHERE
  dimensionality(wire_shell) = 1;
  extent(wire_shell) > 0;
  arcwise_connected(wire_shell);
  genus(wire_shell) >= 0;
  constraints_topology_wire_shell(wire_shell);
  constraints_geometry_wire_shell(wire_shell);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

wire_shell_boundary: List of **edge loops** composing the **shell.**

PROPOSITIONS:

1. The **shell** is arcwise connected.
2. The set of **edge loops** must be connected.

3. Each loop is unique.

4. The loops do not intersect except at common edges or vertices.

5. The function constraints topology wire shell returns TRUE, where the function evaluates the topological constraints.

6. The function constraints geometry wire shell returns TRUE, where the function evaluates the geometric constraints.

4.6.2.17 OPEN SHELL

An open shell is a shell of dimensionality 2 and is a piecewise arcwise connected manifold oriented finite topological surface that is mappable to a plane. It is represented by a collection of faces which do not intersect except at their boundaries.

The connected requirement means that every face must have at least one edge in common with another face in the shell.

The manifold requirement means that not more than two faces may have the same edge as part of their boundary. Alternatively, a vertex may be associated with no more than two of the edges which are referenced only once by the loops of the faces.

A LOGICAL indicator is used to indicate whether the topological normal of a face agrees with (TRUE) or is opposed to (FALSE) the topological normal of the shell.

The following version of the Euler formula must be satisfied

$$\chi_{os} = \mathcal{V} - \mathcal{E} + 2\mathcal{F} - \mathcal{L}_l - (1 - G') = 0 \tag{7}$$

where $\mathcal{V}, \mathcal{E}, \mathcal{F}$ and $\mathcal{L}_l$ are the number of unique vertices, edges, faces and loop uses in the shell and G' is the genus of the shell.

The number of holes (genus) of the shell may be determined in a similar manner as for a face, from the graph of edges and vertices forming the loops of the faces.

- Delete all edges from the graph that are referenced twice by the loops of the faces of the shell.
- Delete all vertices that have no associated edges.
- The genus is the maximum of 0 and (M-1), where M is the multplicity of the resulting graph.
- **A necessary condition for the shell to be manifold is that each of the connected components of this graph has two loops.**

Specifically, the following topological constraints must be met:

- Each face in the shell is unique

$$(S^{\circ})\{F\} = (S^{\circ})[F]$$

- Each loop in the shell is unique

$$((S^{\circ})[F])\{L\} = ((S^{\circ})[F])[L]$$

- Each (edge + logical) pair in the shell is unique

$$(((S^{\circ})[F])[L])\{E_l\} = (((S^{\circ})[F])[L])[E_l]$$

- In the list $(((S^2)[F])[L])[E]$ there is at least one edge that only appears once and no edges appear more than twice; the singleton edges are on the boundary of the shell.
- Equation 7 must be satisfied

$$1 - G' = |((((S^0)[F])\{L^e\})\{E\})\{V\}| + |(((S^0)[F])\{L^v\})\{V\}| - |(((S^0)[F])\{L\})\{E\}| + 2|(S^0)[F]| - |((S^0)[F])[L]|$$

```
*)
ENTITY open_shell
  SUBTYPE OF (shell);
  shell_boundary  : SET [1 : #] OF face_logical_structure;
WHERE
  manifold(open_shell);
  arcwise_connected(open_shell);
  dimensionality(open_shell) = 2;
  genus(open_shell) >= 0;
  constraints_topology_open_shell(open_shell);
  constraints_geometry_open_shell(open_shell);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

shell_boundary: The set of face and logical flag combinations composing the **open shell.**

PROPOSITIONS:

1. An **open shell** must have at least one **face.**
2. The **shell** is arcwise connected.
3. The **faces** in an **open shell** do not intersect except at their boundaries.
4. A given **face** may exist in more than one **open shell.**
5. A **face** may exist without being part of an **open shell.**
6. The **loops** of the **shell** must not be a mixture of **poly loops** and other **loop** types.
7. **Constraints topology open shell** returns TRUE when the function evaluates the topological constraints.
8. **Constraints geometry open shell** returns TRUE when the function evaluates the geometric constraints.

4.6.2.18 FACE LOGICAL STRUCTURE

This entity associates a **face** and a LOGICAL.

```
*)
ENTITY face_logical_structure;
  face_element : face;
  flag         : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

face_element: A face.

flag: A LOGICAL value associated with the face.

4.6.2.19 CLOSED SHELL

A **closed shell** is a shell of dimensionality 2 and is a piecewise closed oriented manifold topological surface mappable to an n-fold torus. The domain of the shell includes it's boundary, and $0 < \Xi < \infty$. The shell divides R^3 into two regions, one finite and the other infinite. The topological normal of the shell is directed from the finite to the infinite region.

The shell is represented by a collection of faces. The faces do not intersect except at their boundaries. Associated with each face in the shell is a logical value which indicates whether the face normal agrees with (TRUE) or is opposed to (FALSE) the shell normal.

Every edge must be referenced twice by the loops of the faces and each (edge + direction) reference must be unique. Providing the shell is well formed, the genus G is given by

$$G = \frac{1}{2}(L_s + L_g) - F - (M - 1) \geq 0 \text{ and integer} \qquad (8)$$

where L_s and L_g are the numbers of shell and graph loops respectively, F is the number of faces in the shell and M is the multiplicity of the graph. A non-integer value for G denotes an ill formed shell, but an integer value does not necessarily imply a well formed shell

The following version of the Euler formula must be satisfied for the shell:

$$\chi_{cs} = \mathcal{V} - \mathcal{E} + 2\mathcal{F} - \mathcal{L}_l - 2(1 - G') = 0 \qquad (9)$$

where $\mathcal{V}, \mathcal{E}, \mathcal{F}$ and $\mathcal{L}_l$ are the numbers of unique vertices, edges, faces and loop uses in the shell and G' is the genus of the shell.

Specifically, the following topological constraints must be satisfied:

- Each face in the shell is unique

$$(S^c)\{F\} = (S^c)[F]$$

- Each loop in the shell is unique

$$((S^c)[F])\{L\} = ((S^c)[F])[L]$$

- Each (edge + logical) pair in the shell is unique

$$(((S^c)[F])[L])\{E_l\} = (((S^c)[F])[L])[E_l]$$

- Each edge in the shell is either used by exactly two loops or is used twice by one loop

$$|(((S^c)[F])[L])\{E_l\}| = 2|(((S^c)[F])[L])\{E\}|$$

That is, in the list $(((S^c)[F])[L])[E]$ each edge appears exactly twice.

- Equation 9 must be satisfied

$$\begin{aligned} 2 - 2G' \;=\; & |((((S^\circ)[F])\{L^e\})\{E\})\{V\}| + |(((S^\circ)[F])\{L^v\})\{V\}| \\ & -|(((S^\circ)[F])\{L\})\{E\}| + 2|(S^\circ)[F]| - |((S^\circ)[F])[L]| \end{aligned}$$

```
*)
ENTITY closed_shell
  SUBTYPE OF (shell);
  cshell_boundary  : SET [1 : #] OF face_logical_structure;
WHERE
  manifold(closed_shell);
  arcwise_connected(closed_shell);
  dimensionality(closed_shell) = 2;
  genus(closed_shell) >= 0;
  constraints_topology_closed_shell(closed_shell);
  constraints_geometry_closed_shell(closed_shell);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

cshell_boundary: The set of **face** and logical flag combinations composing the **closed shell.**

PROPOSITIONS:

1. A **closed shell** must have at least one **face.**
2. The **shell** is manifold and arcwise connected.
3. The **faces** of the **shell** are unique.
4. The **faces** of a **closed shell** do not intersect each other, except along their boundaries.
5. A given **face** may exist in more than one **closed shell.**
6. A **face** may exist without being part of a **closed shell.**
7. The **loops** of the **shell** must not be a mixture of **poly loops** and other **loop** types.
8. **Constraints topology closed shell** returns TRUE when the function evaluates the topological constraints.
9. **Constraints geometry closed shell** returns TRUE when the function evaluates the geometric constraints.

4.6.2.20 REGION

A **region** is a (finite) arcwise connected portion of R^3 with dimensionality 3. The extent of a region does not include its boundaries and $0 < \Xi \leq \infty$.

The boundaries of a region are one or more shells of dimensionality D where $0 \leq D \leq 2$. The shells of a region must not intersect except at a vertex or along an edge and may overlap along common vertices, edges or faces.

When the boundaries of the region are non-closed shells the region extent is semi-infinite (i.e., there is an implied boundary at infinity).

When one of the boundaries is a closed shell, the topological normal of the shell is directed away from the extent (interior) of the region. In this case an indicator is used to signify whether the shell normal, as defined in the shell, points towards the exterior (TRUE) or the interior (FALSE) of the region.

```
*)
ENTITY region
  SUBTYPE OF (topology);
  outer_region_boundary : OPTIONAL shell_logical_structure;
  region_boundaries     : SET [1 : #] OF shell_logical_structure;
WHERE
  dimensionality(region) = 3;
  (0 < extent(region) <= infinity);
  arcwise_connected(region);
  constraints_topology_region(region);
  constraints_geometry_region(region);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

outer_region_boundary: Shell constituting the outer boundary of the **region.** This attribute is optional and is primarily for the use of possible future extensions to non-manifold solid Brep representations.

region_boundaries: List of **shells** describing all the boundaries of the **region.**

PROPOSITIONS:

1. A **region** is arcwise connected.
2. The **shells** of a region may overlap at common **faces, edges** or **vertices.**
3. There must be at least one **shell** associated with a **region.**
4. A **shell** may be used by more than one **region.**
5. A **region** is not necessary to the existance of a **shell.**
6. **Faces** may not touch or intersect except along their boundaries.
7. Each **shell** in a **region** is unique.
8. The function **constraints topology region** returns TRUE, where the function evaluates the topological constraints.
9. The function **constraints geometry region** returns TRUE, where the function evaluates the geometric constraints.

4.6.2.21 SHELL LOGICAL STRUCTURE

This entity associates a LOGICAL value with a **shell.**

```
*)
ENTITY shell_logical_structure;
  shell_element : shell;
  flag          : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

shell_element: A shell.

flag: A LOGICAL value associated with the shell.

4.6.2.22 CONNECTED EDGE SET

A connected edge set is a set consisting of connected edges.

```
*)
ENTITY connected_edge_set
  SUBTYPE OF (topology);
  ces_edges : SET [1:#] OF edges;
WHERE
  arcwise_connected(connected_edge_set);
  dimensionality(connected_edge_set) = 1;
  constraints_topology_connected_edge_set(connected_edge_set);
  constraints_geometry_connected_edge_set(connected_edge_set);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

ces_edges: Set of edges arcwise connected at common vertices.

PROPOSITIONS:

1. The connected edge set is arcwise connected.
2. The dimensionality of the connected edge set is 1.

4.6.2.23 CONNECTED FACE SET

A connected face set is a set consisting of connected faces.

```
*)
ENTITY connected_face_set
  SUBTYPE OF (topology);
  cfs_faces : SET [1:#] OF faces;
WHERE
  arcwise_connected(connected_face_set);
  dimensionality(connected_face_set) = 2;
  constraints_topology_connected_face_set(connected_face_set);
  constraints_geometry_connected_face_set(connected_face_set);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

cfs_faces: Set of faces arcwise connected along common edges.

PROPOSITIONS:

1. The connected face set is arcwise connected.

2. The dimensionality of the connected face set is 2.

4.6.3 Topology FUNCTION Definitions

4.6.3.1 Topology Geometric Constraints

Included here are functions that check the geometric constraints on the topological entities.

4.6.3.1.1 CONSTRAINTS GEOMETRY VERTEX

```
*)
FUNCTION constraints_geometry_vertex(v : vertex) : LOGICAL;
  RETURN(EXISTS(v.vertex_point) AND
         domain(v) <> domain(v.vertex_point))
         OR
         NOT EXISTS(v.vertex_point);
END_FUNCTION;
(*
```

4.6.3.1.2 CONSTRAINTS GEOMETRY EDGE

This function evaluates the geometric constraints on an edge and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_edge(e: edge): LOGICAL;
  LOCAL
    result : LOGICAL := TRUE;
  END_LOCAL;
  REPEAT FOR EACH vertex IN set_edge_vertices(e);
    IF distance(vertex, e.edge_curve) <> 0 THEN
      result := FALSE;
    (* vertex geometry must be consistent with curve geometry *)
    END_IF;
  END_REPEAT;
  IF NOT embedded(e, e.edge_curve) THEN
    result := FALSE;
  (* the domain of the curve must include the domain of the edge *)
  END_IF;
  IF e.edge_curve.flag <> agreement(e, e.edge_curve.curve) THEN
    result := FALSE;
  (* the flag is set to make the edge and curve directions agree *)
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.1.3 CONSTRAINTS GEOMETRY PATH

This function evaluates the geometric constraints on a path and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_path(p: path): LOGICAL;
  LOCAL
    result : LOGICAL := TRUE;
  END_LOCAL;
  REPEAT i := 1 TO SIZEOF(p.open_edge_list);
    IF (open_edge_list[i].flag <>
       agreement(p, open_edge_list[i].edge_element)) THEN
        result := FALSE;
    (* the flag is set to make each edge direction agree with
       the path direction *)
    END_IF;
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.1.4 CONSTRAINTS GEOMETRY EDGE LOOP

This function evaluates the geometric constraints on an edge loop and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_edge_loop(l: edge_loop): LOGICAL;
  LOCAL
    result : LOGICAL := TRUE;
  END_LOCAL;
  REPEAT i := 1 TO SIZEOF(l.loop_edges);
    IF (loop_edges[i].flag <>
        agreement(l, loop_edges[i].edge_element)) THEN
      result := FALSE;
    (* the flag is set to make each edge direction agree with
       the loop direction *)
    END_IF;
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.1.5 CONSTRAINTS GEOMETRY FACE

This function evaluates the geometric constraints on a face and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_face(f: face): LOGICAL;
  LOCAL
```

```
  result     : LOGICAL := TRUE;
  outer      : loop;
  loop_list  : list_of_loop := list_face_loops(f);
  loop_set   : set_of_loop := set_face_loops(f);
END_LOCAL;
IF mixed_loop_type_set(loop_set) THEN
  result := FALSE;
  (* the loops must not be a mixture of poly-loops and
     other loop types *)
END_IF;
IF TYPEOF(loop_list[1]) = poly_loop THEN
  BEGIN
    IF (NOT co_planar(f, loop_set)) OR
       (NOT co_planar(f, f.face_surface)) THEN
      result := FALSE;
    (* a face defined by poly-loops must be planar *)
    END_IF;
  END;
END_IF;
IF self_intersect(loop_set) THEN
  result := FALSE;
  (* the loops are disjoint *)
END_IF;
REPEAT FOR EACH loop IN loop_set;
  IF (intersect(loop, domain(f)) <> NULL) THEN
    result := FALSE;
  END_IF;
  (* the domain of a face does not include its boundaries *)
END_REPEAT;
(* get outer bound of face *)
IF EXISTS(f.outer_bound) THEN
  outer := f.outer_bound.loop_element;
ELSE
  (* pseudo code for calculating outer boundary loop
  outer := loop;
     end of pseudo code *);
END_IF;
REPEAT FOR EACH loop IN loop_set;
  IF loop <> outer THEN
    BEGIN
      IF NOT inside(outer, loop, f) THEN
        result := FALSE;
      END_IF;
      (* each face bound is inside the outer bound *)
      REPEAT i := 1 TO SIZEOF(loop_list);
        IF (loop <> loop_list[i]) AND
            outer <> loop_list[i] AND
           (inside(loop, loop_list[i], f)) THEN
          result := FALSE;
```

```
          END_IF;
          (* no loop, except the outer, encloses another loop *)
        END_REPEAT;
      END;
    END_IF;
  END_REPEAT;
  IF EXISTS(f.face_surface) THEN
    BEGIN
      IF (f.face_surface.flag <>
          agreement(f, f.face_surface.surface_element)) THEN
        result := FALSE;
      END_IF;
      (* the flag sets the direction of the face and
         surface to match *)
      REPEAT FOR EACH loop IN loop_set;
        IF distance(loop, f.face_surface) <> 0 THEN
          result := FALSE;
        END_IF;
        (* vertex and edge geometry must be consistent with
           face geometry *)
      END_REPEAT;
    END;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.1.6 CONSTRAINTS GEOMETRY SUBFACE

This function evaluates the geometric constraints on a subface and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_subface(f: subface): LOGICAL;
  LOCAL
    result    : LOGICAL := TRUE;
    outer     : loop;
    loop_list : list_of_loop := list_face_loops(f);
    loop_set  : set_of_loop := set_face_loops(f);
  END_LOCAL;
  IF mixed_loop_type_set(loop_set) THEN
    result := FALSE;
    (* the loops must not be a mixture of poly-loops and
       other loop types *)
  END_IF;
  IF TYPEOF(loop_list[1]) = poly_loop THEN
    BEGIN
      IF (TYPEOF(f.trimming.bounds[i].loop_element) <>
          poly_loop) THEN
        result := FALSE;
```

```
      END_IF;
      (* the loops of the subface must be of the same type as
         the face being trimmed *)
    END;
  END_IF;
  IF self_intersect(loop_set) THEN
    result := FALSE;
    (* the loops are disjoint *)
  END_IF;
  REPEAT FOR EACH loop IN loop_set;
    IF (intersect(loop, domain(f)) <> NULL) THEN
      result := FALSE;
    END_IF;
    (* the domain of a subface does not include its boundaries *)
  END_REPEAT;
  (* get outer bound of subface *)
  IF EXISTS(f.outer_bound) THEN
    outer := f.outer_bound.loop_element;
  ELSE
    (* pseudo code for calculating outer boundary loop
    outer := loop;
       end of pseudo code *);
  END_IF;
  REPEAT FOR EACH loop IN loop_set;
    IF loop <> outer THEN
      BEGIN
        IF NOT inside(outer, loop, f) THEN
          result := FALSE;
        END_IF;
        (* each subface bound is inside the outer bound *)
        REPEAT i := 1 TO SIZEOF(loop_list);
          IF (loop <> loop_list[i]) AND
             (outer <> loop_list[i]) AND
             (inside(loop, loop_list[i], f)) THEN
            result := FALSE;
          END_IF;
          (* no loop, except the outer, encloses another loop *)
        END_REPEAT;
      END;
    END_IF;
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.1.7 CONSTRAINTS GEOMETRY WIRE SHELL

This function evaluates the geometric constraints on a wire shell and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_wire_shell(s: wire_shell): LOGICAL;
  LOCAL
    result    : LOGICAL := TRUE;
    loop_list : list_of_loop := list_shell_loops(s);
    loop_set  : set_of_loop := set_shell_loops(s);
  END_LOCAL;
  IF mixed_loop_type_set(loop_set) THEN
    result := FALSE;
    (* the loops must not be a mixture of poly-loops and
       other loop types *)
  END_IF;
  IF self_intersect(s) THEN
    result := FALSE;
    (* the loops do not intersect except at common edges
       and vertices *)
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.1.8 CONSTRAINTS GEOMETRY OPEN SHELL

This function evaluates the geometric constraints on an open shell and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_open_shell(s: open_shell): LOGICAL;
  LOCAL
    result    : LOGICAL := TRUE;
    face_list : list_of_face := list_shell_faces(s);
    face_set  : set_of_face := set_shell_faces(s);
    loop_set  : set_of_loop := set_shell_loops(s);
  END_LOCAL;
  IF mixed_loop_type_set(loop_set) THEN
    result := FALSE;
    (* the loops must not be a mixture of poly-loops and
       other loop types *)
  END_IF;
  REPEAT i := 1 TO SIZEOF(s.shell_boundary);
    IF (s.shell_boundary[i].flag <>
        agreement(s, s.shell_boundary[i].face_element)) THEN
      result := FALSE;
    END_IF;
    (* face normals must match the shell normal *)
  END_REPEAT;
  REPEAT FOR EACH face IN face_set;
    REPEAT i := 1 TO SIZEOF(face_list);
      IF (face <> face_list[i]) AND
```

```
          (intersect(face, face_list[i])) THEN
        result := FALSE;
      END_IF;
      (* the faces of the shell do not intersect, except
         at their boundaries *)
    END_REPEAT;
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.1.9 CONSTRAINTS GEOMETRY CLOSED SHELL

This function evaluates the geometric constraints on a closed shell and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_closed_shell(s: closed_shell)
                                                    : LOGICAL;
  LOCAL
    result    : LOGICAL := TRUE;
    face_list : list_of_face := list_shell_faces(s);
    face_set  : set_of_face := set_shell_faces(s);
    loop_set  : set_of_loop := set_shell_loops(s);
  END_LOCAL;
  IF mixed_loop_type_set(loop_set) THEN
    result := FALSE;
    (* the loops must not be a mixture of poly-loops and
       other loop types *)
  END_IF;
  REPEAT i := 1 TO SIZEOF(s.shell_boundary);
    IF (s.shell_boundary[i].flag <>
        agreement(s, s.shell_boundary[i].face_element)) THEN
      result := FALSE;
    END_IF;
    (* face normals must match the shell normal *)
  END_REPEAT;
  REPEAT FOR EACH face IN face_set;
    REPEAT i := 1 TO SIZEOF(face_list);
      IF (face <> face_list[i]) AND
          intersect(face, face_list[i]) THEN
        result := FALSE;
      END_IF;
      (* the faces of the shell do not intersect, except at
         their boundaries *)
    END_REPEAT;
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.1.10 CONSTRAINTS GEOMETRY REGION

This function evaluates the geometric constraints on a region and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_region(r: region): LOGICAL;
  LOCAL
    result : LOGICAL;
  END_LOCAL;
  result := TRUE;
  IF EXISTS(r.outer_region_boundary) THEN
    BEGIN
      IF (typeof(r.outer_region_boundary.shell_element) <>
          closed_shell) THEN
        result := FALSE;
      END_IF;
      (* if declared, the outer boundary must be a CLOSED SHELL *)
      IF NOT r.outer_region_boundary IN r.region_boundaries THEN
        result := FALSE;
      (* and must be included in the list of boundaries *)
      END_IF;
      REPEAT i := 1 TO SIZEOF(r.region_boundaries);
        IF (r.region_boundaries[i] <> r.outer_region_boundary) AND
           (NOT inside(r.outer_region_boundary.shell_element,
                       r.region_boundaries[i].shell_element, r))
                                                              THEN
             result := FALSE;
        END_IF;
        (* all shells must be within the outer shell *)
      END_REPEAT;
    END;
  END_IF;
  REPEAT i := 1 TO SIZEOF(r.region_boundaries);
    IF (r.region_boundaries[i].flag <>
        agreement(r.region_boundaries[i].shell_element, r)) THEN
      result := FALSE;
    END_IF;
    (* Shell and region orientations must match *)
    REPEAT j := 1 TO SIZEOF(r.region_boundaries);
      IF i <> j THEN
        BEGIN
          IF intersect(r.region_boundaries[i],
                       r.region_boundaries[j]) THEN
            result := FALSE;
          END_IF;
        END;
      END_IF;
      (* region boundaries do not intersect but may overlap at
         common faces, edges and vertices *)
    END_REPEAT;
```

```
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.1.11 CONSTRAINTS GEOMETRY CONNECTED FACE SET

```
*)
FUNCTION constraints_geometry_connected_face_set)
          (s : connected_face_set) : LOGICAL;
  LOCAL
    result    : LOGICAL := TRUE;
    face_list : list_of_face := list_faces(s);
    face_set  : set_of_face := set_faces(s);
    loop_set  : set_of_loop := set_loops(s);
    i,j       : INTEGER;
  END_LOCAL;
  IF mixed_loop_type_set(loop_set) THEN
    result := FALSE;
    (* the loops must not be a mixture of ply-loops and
       other loop types *)
  END_IF;
  REPEAT j := 1 TO SIZEOF(face_set);
    REPEAT i := 1 TO SIZEOF(face_list);
      IF (face_set[j] <> face_list[i]) AND
         (does_intersect(face_set[j], face_list[i])) THEN
        result := FALSE;
       (* faces must not intersect except at their boundaries *)
      END_IF;
    END_REPEAT;
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2 Topology Topological Constraints

Included here are functions that check the topological constraints on the topological entities.

4.6.3.2.1 CONSTRAINTS TOPOLOGY PATH

This function evaluates the topological constraints on a path and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_topology_path(p: path): LOGICAL;
  LOCAL
    edge_list         : list_of_edge;
    edge_set          : set_of_edge;
    vtx_list          : list_of_vertex;
```

```
    vtx_set          : set_of_vertex;
    no_of_singletons : INTEGER := 0;
    result           : LOGICAL := TRUE;
  END_LOCAL;
  IF TYPEOF(p) <> path THEN
    result := FALSE;
  END_IF;
    (* calculate list and set of path edges *)
  REPEAT i := 1 TO SIZEOF(p.open_edge_list);
    edge_list := edge_list + p.open_edge_list[i].edge_element;
  END_REPEAT;
  edge_set := edge_set + edge_list;
    (* check edge uniqueness *)
  IF (edge_list <> edge_set) THEN
    result := FALSE;
  END_IF;
    (* calculate list and set of path vertices *)
  REPEAT FOR EACH edge IN edge_set;
    vtx_list := vtx_list + list_edge_vertices(edge);
  END_REPEAT;
  vtx_set := vtx_set + vtx_list;
    (* calculate vertex occurances *)
  REPEAT FOR EACH vertex IN vtx_set;
    IF (occurs(vertex, vtx_list) = 1) THEN
      no_of_singletons := no_of_singletons +1;
    END_IF;
    IF (occurs(vertex, vtx_list) > 2) THEN
      result := FALSE;
    END_IF;
  END_REPEAT;
  IF (no_of_singletons <> 2) THEN
    result := FALSE;
  END_IF;
    (* check genus and euler formula *)
  IF (genus(p) <> 0) THEN
    result := FALSE;
  END_IF;
  IF (SIZEOF(vtx_set) - SIZEOF(edge_set) - 1 <> 0) THEN
    result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2.2 CONSTRAINTS TOPOLOGY LOOP

This function evaluates the topological constraints on a loop and returns TRUE if they are satisfied.

```
*)
```

```
FUNCTION constraints_topology_loop(l: loop): LOGICAL;
  LOCAL
    result : LOGICAL := TRUE;
  END_LOCAL;
  IF TYPEOF(l) <> loop THEN
    result := FALSE;
  END_IF;
  CASE TYPEOF(l) OF
    edge_loop   : result := constraints_topology_edge_loop(l);
    poly_loop   : BEGIN
                    REPEAT FOR EACH poly_loop IN MODEL;
                      IF NOT UNIQUE polygon THEN
                        result := FALSE;
                      END_IF;
                    END_REPEAT;
                  END;
    vertex_loop : BEGIN
                    REPEAT FOR EACH vertex_loop IN MODEL;
                      IF NOT UNIQUE loop.vertex THEN
                        result := FALSE;
                      END_IF;
                    END_REPEAT;
                  END;
  END_CASE;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2.3 CONSTRAINTS TOPOLOGY EDGE LOOP

This function evaluates the topological constraints on an edge loop and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_topology_edge_loop(l: edge_loop): LOGICAL;
  LOCAL
    edge_list     : list_of_edge := list_loop_edges(l);
    edge_set      : set_of_edge := set_loop_edges(l);
    edge_log_list : list_of_edge_logical_structure :=
                            list_loop_edge_logicals(l);
    edge_log_set  : set_of_edge_logical_structure;
    vtx_list      : list_of_vertex;
    vtx_set       : set_of_vertex := set_loop_vertices(l);
    result        : LOGICAL := TRUE;
  END_LOCAL;
  IF TYPEOF(l) <> edge_loop THEN
    result := FALSE;
  END_IF;
  (* calculate set of edge/logicals in loop *)
```

```
  edge_log_set := edge_log_set + edge_log_list;
  (* check edge/logical uniqueness *)
  IF (edge_log_list <> edge_log_set) THEN
    result := FALSE;
  END_IF;
  (* calculate set and list of vertices *)
  REPEAT FOR EACH edge IN edge_set;
    vtx_list := vtx_list + list_edge_vertices(edge);
  END_REPEAT;
  (* check vertices and edge/logical pairs *)
  IF SIZEOF(vtx_list) <> 2*SIZEOF(edge_log_set) THEN
    result := FALSE;
  END_IF;
  (* check genus *)
  IF (genus(1) < 1) THEN
    result := FALSE;
  END_IF;
  (* check Euler formula *)
  IF (SIZEOF(vtx_set) - SIZEOF(edge_set) - 1 + genus(1) <> 0) THEN
     result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2.4 CONSTRAINTS TOPOLOGY FACE

This function evaluates the topological constraints on a face or a subface and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_topology_face(f: select_face_or_subface)
                                                          : LOGICAL;
  LOCAL
    loop_list      : list_of_loop := list_face_loops(f);
    loop_set       : set_of_loop := set_face_loops(f);
    edge_list      : list_of_edge := list_face_edges(f);
    edge_set       : set_of_edge := set_face_edges(f);
    edge_log_list  : list_of_edge_logical_structure;
    edge_log_set   : set_of_edge_logical_structure;
    vtx_set        : set_of_vertex := set_face_vertices(f);
    loop_genus     : INTEGER := 0;
    result         : LOGICAL := TRUE;
  END_LOCAL;
  IF TYPEOF(f) <> select_face_or_subface THEN
    result := FALSE;
  END_IF;
  (* check uniqueness of loops *)
  IF (loop_list <> loop_set) THEN
```

```
    result := FALSE;
  END_IF;
  (* calculate set of edge/logicals in face *)
  REPEAT FOR EACH loop IN loop_set;
    loop_genus := loop_genus + genus(loop);
    edge_log_list := edge_log_list + list_loop_edge_logicals(loop);
  END_REPEAT;
  edge_log_set := edge_log_set + edge_log_list;
  (* check edge/logical uniqueness *)
  IF (edge_log_list <> edge_log_set) THEN
    result := FALSE;
  END_IF;
  (* check that an edge is not used more than twice *)
  REPEAT FOR EACH edge IN edge_set;
    IF occurs(edge, edge_list) > 2 THEN
      result := FALSE;
    END_IF;
  END_REPEAT;
  (* check genus *)
  IF (genus(f) < 0) THEN
    result := FALSE;
  END_IF;
  (* check Euler formula *)
  IF (SIZEOF(vtx_set) - SIZEOF(edge_set) - SIZEOF(loop_set) +
      loop_genus <> 0) THEN
    result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2.5 CONSTRAINTS TOPOLOGY SHELL

The function evaluates the topological constraints on a shell and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_topology_shell(s: shell): LOGICAL;
  LOCAL
    result : LOGICAL := TRUE;
  END_LOCAL;
  IF TYPEOF(s) <> shell THEN
    result := FALSE;
  END_IF;
  CASE TYPEOF(s) OF
    closed_shell : result := constraints_topology_closed_shell(s);
    open_shell   : result := constraints_topology_open_shell(s);
    wire_shell   : result := constraints_topology_wire_shell(s);
    vertex_shell : BEGIN
          REPEAT FOR EACH vertex_shell IN MODEL;
```

```
            IF NOT UNIQUE vertex_shell.vertex_shell_boundary THEN
              result := FALSE;
            END_IF;
          END_REPEAT;
                END;
  END_CASE;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2.6 CONSTRAINTS TOPOLOGY CLOSED SHELL

This function evaluates the topological constraints on a closed shell and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_topology_closed_shell(s: closed_shell)
                                                    : LOGICAL;
  LOCAL
    face_list      : list_of_face := list_shell_faces(s);
    face_set       : set_of_face := set_shell_faces(s);
    loop_list      : list_of_loop := list_shell_loops(s);
    loop_set       : set_of_loop := set_shell_loops(s);
    edge_list      : list_of_edge := list_shell_edges(s);
    edge_set       : set_of_edge := set_shell_edges(s);
    edge_log_list  : list_of_edge_logical_structure;
    edge_log_set   : set_of_edge_logical_structure;
    vtx_set        : set_of_vertex := set_shell_vertices(s);
    result         : LOGICAL := TRUE;
  END_LOCAL;
  IF TYPEOF(s) <> closed_shell THEN
    result := FALSE;
  END_IF;
  (* check face uniqueness *)
  IF (face_list <> face_set) THEN
    result := FALSE;
  END_IF;
  (* check uniqueness of loops *)
  IF loop_list <> loop_set THEN
    result := FALSE;
  END_IF;
  (* calculate set of edge/logicals in shell *)
  REPEAT FOR EACH loop IN loop_set;
    edge_log_list := edge_log_list + list_loop_edge_logicals(loop);
  END_REPEAT;
  edge_log_set := edge_log_set + edge_log_list;
  (* check edge/logical uniqueness *)
  IF (edge_log_list <> edge_log_set) THEN
    result := FALSE;
```

```
  END_IF;
  (* Check if each edge is used twice by the loops of the shell *)
  REPEAT FOR EACH edge IN edge_set;
    IF (occurs(edge, edge_list) <> 2) THEN
      result := FALSE;
    END_IF;
  END_REPEAT;
  (* check genus *)
  IF (genus(s) < 0) THEN
    result := FALSE;
  END_IF;
  (* check Euler formula *)
  IF (SIZEOF(vtx_set) - SIZEOF(edge_set) + 2*SIZEOF(face_set) -
      SIZEOF(loop_list) - 2*(1 - genus(s)) <> 0) THEN
    result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2.7 CONSTRAINTS TOPOLOGY OPEN SHELL

This function evaluates the topological constraints on an open shell and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_topology_open_shell(s: open_shell): LOGICAL;
  LOCAL
    face_list         : list_of_face := list_shell_faces(s);
    face_set          : set_of_face := set_shell_faces(s);
    loop_list         : list_of_loop := list_shell_loops(s);
    loop_set          : set_of_loop := set_shell_loops(s);
    edge_list         : list_of_edge := list_shell_edges(s);
    edge_set          : set_of_edge := set_shell_edges(s);
    edge_log_list     : list_of_edge_logical_structure;
    edge_log_set      : set_of_edge_logical_structure;
    vtx_set           : set_of_vertex := set_shell_vertices(s);
    no_of_singletons  : INTEGER := 0;
    result            : LOGICAL := TRUE;
  END_LOCAL;
  IF TYPEOF(s) <> open_shell THEN
    result := FALSE;
  END_IF;
  (* check face uniqueness *)
  IF (face_list <> face_set) THEN
    result := FALSE;
  END_IF;
  (* check uniqueness of loops *)
  IF loop_list <> loop_set THEN
```

```
    result := FALSE;
  END_IF;
  (* calculate set of edge/logicals in shell *)
  REPEAT FOR EACH loop IN loop_set;
    edge_log_list := edge_log_list + list_loop_edge_logicals(loop);
  END_REPEAT;
  edge_log_set := edge_log_set + edge_log_list;
  (* check edge/logical uniqueness *)
  IF (edge_log_list <> edge_log_set) THEN
    result := FALSE;
  END_IF;
  (* check edge occurances *)
  REPEAT FOR EACH edge IN edge_set;
    IF (occurs(edge, edge_list) = 1) THEN
        no_of_singletons := no_of_singletons + 1;
    END_IF;
    IF (occurs(edge, edge_list) > 2) THEN
      result := FALSE;
    END_IF;
  END_REPEAT;
  IF no_of_singletons < 1 THEN
    result := FALSE;
  END_IF;
  (* check genus *)
  IF (genus(s) < 0) THEN
    result := FALSE;
  END_IF;
  (* check Euler formula *)
  IF (SIZEOF(vtx_set) - SIZEOF(edge_set) + 2*SIZEOF(face_set) -
      SIZEOF(loop_list) - (1 - genus(s)) <> 0) THEN
    result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2.8 CONSTRAINTS TOPOLOGY WIRE SHELL

This function evaluates the topological constraints on a wire shell and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_topology_wire_shell(s: wire_shell): LOGICAL;
  LOCAL
    loop_list     : list_of_loop := list_shell_loops(s);
    loop_set      : set_of_loop := set_shell_loops(s);
    edge_list     : list_of_edge := list_shell_edges(s);
    edge_set      : set_of_edge := set_shell_edges(s);
    edge_log_list : list_of_edge_logical_structure;
```

```
    edge_log_set   : set_of_edge_logical_structure;
    vtx_set        : set_of_vertex := set_shell_vertices(s);
    loop_genus     : INTEGER := 0;
    result         : LOGICAL := TRUE;
  END_LOCAL;
  IF TYPEOF(s) <> wire_shell THEN
    result := FALSE;
  END_IF;
  (* check uniqueness of loops *)
  IF (loop_list <> loop_set) THEN
    result := FALSE;
  END_IF;
  (* calculate set of edge/logicals in shell *)
  REPEAT FOR EACH loop IN loop_set;
    edge_log_list := edge_log_list + list_loop_edge_logicals(loop);
  END_REPEAT;
  edge_log_set := edge_log_set + edge_log_list;
  (* check edge/logical uniqueness *)
  IF (edge_log_list <> edge_log_set) THEN
    result := FALSE;
  END_IF;
  (* check number of edge references *)
  REPEAT FOR EACH edge IN edge_set;
    IF (occurs(edge, edge_list) <> 2) THEN
      result := FALSE;
    END_IF;
  END_REPEAT;
  (* check genus *)
  IF (genus(s) < 0) THEN
    result := FALSE;
  END_IF;
  (* calculate sum of loop genus in shell *)
  REPEAT FOR EACH loop IN loop_set;
    loop_genus := loop_genus + genus(loop);
  END_REPEAT;
  (* check Euler formula *)
  IF (SIZEOF(vtx_set) - SIZEOF(edge_set) - SIZEOF(loop_set) +
      loop_genus <> 0) THEN
       result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2.9 CONSTRAINTS TOPOLOGY REGION

This function evaluates the topological constraints on a region and returns TRUE if they are satisfied.

```
*)
```

```
FUNCTION constraints_topology_region(r: region): LOGICAL;
  LOCAL
    shell_list : LIST [1 : #] OF shell;
    shell_set  : SET OF shell;
    result     : LOGICAL := TRUE;
  END_LOCAL;
  REPEAT FOR EACH shell_logical_structure IN
                                          region.region_boundaries;
    shell_list := shell_list +
                  shell_logical_structure.shell_element;
  END_REPEAT;
  shell_set := shell_set + shell_list;
  (* check shell uniqueness *)
  IF shell_set <> shell_list THEN
    result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.3.2.10 CONSTRAINTS TOPOLOGY CONNECTED FACE SET

```
*)
FUNCTION constraints_topology_connected_face_set
         (s : connected_face_set) : LOGICAL;
  LOCAL
    face_list : LIST [1:#] OF face := list_shell_faces(s);
    face_set  : SET [1:#] OF face := set_shell_faces(s);
    result    : LOGICAL := TRUE;
  END_LOCAL;
  (* check face uniqueness *)
  IF (face_list <> face_set) THEN
    result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.6.4 Topology Classification Structure

The following indented list provides the classification structure for the Topology Model Schema.

```
CURVE LOGICAL STRUCTURE
EDGE LOGICAL STRUCTURE
FACE LOGICAL STRUCTURE
LOOP LOGICAL STRUCTURE
SHELL LOGICAL STRUCTURE
TOPOLOGY
  CONNECTED EDGE SET
```

```
  CONNECTED FACE SET
  EDGE
  FACE
  LOOP
    EDGE LOOP
    POLY LOOP
    VERTEX LOOP
  PATH
  REGION
  SHELL
    CLOSED SHELL
    OPEN SHELL
    VERTEX SHELL
    WIRE SHELL
  SUBFACE
  VERTEX
SURFACE LOGICAL STRUCTURE

*)
END_SCHEMA; -- end TOPOLOGY schema
(*
```

4.7 Shape Representation

```
*)
SCHEMA ipim_shape_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_design_shape_schema,
         ipim_nominal_shape_schema,
         ipim_features_schema,
         ipim_shape_interface_schema);
(*
```

4.7.1 Design Shape

4.7.1.1 Introduction

This Section describes the Shape Representation Integrated Product Information Model for complete shape models.

```
*)
SCHEMA ipim_design_shape_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_nominal_shape_schema,
         ipim_features_schema,
         ipim_tolerances_schema,
         ipim_lep_schema,
         ipim_geometry_schema);
(*
```

4.7.1.2 DESIGN MODEL

This entity provides a general "entry point" into the Shape Design world.
The LEP MODEL is the Layered Electrical Product Model that has nearly obtained Sub-committee Draft status. It is included here mainly as a place holder for future integration. It is expected that other models, such as Drafting, may also be integrated as SUBTYPES of this entity.

```
*)
ENTITY design_model
  SUPERTYPE OF (assembly_model XOR
                part_model XOR
                layered_electrical_product);
END_ENTITY;
(*
```

4.7.1.3 ASSEMBLY MODEL

This entity collects together the components of an assembly of parts. Note that this is the first pass at a definition and will require rework and completion.

```
*)
ENTITY assembly_model
  SUBTYPE OF (design_model);
  sub_assemblies        : OPTIONAL LIST [0 : #] OF
                                   assembly_model_structure;
  piece_parts           : OPTIONAL LIST [0 : #] OF
                                   part_model_structure;
  assembly_model_units  : units;
  assembly_relations    : LIST [0 : #] OF UNDEFINED;
  assembly_tolerances   : LIST [0 : #] OF shape_tolerance;
WHERE
  SIZEOF(sub_assemblies) + SIZEOF(piece_parts) >= 2;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sub_assemblies: The sub-assemblies composing the assembly.

piece_parts: The individual pieceparts composing the assembly.

assembly_model_units: The (length) unit that applies to the location specification of the items within the assembly.

assembly_relations: The relations (assembly features?) between the components of the assembly.

assembly_tolerances: The tolerances between the assembly components.

PROPOSITIONS:

1. An assembly model must have at least two components.
2. Sub-assemblies and pieceparts are located with respect to the coordinate system and units of the assembly model.

4.7.1.4 ASSEMBLY MODEL STRUCTURE

This entity associates a Local Coordinate System with an assembly model.

```
*)
ENTITY assembly_model_structure;
  model_element : assembly_model;
  location      : axis2_placement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

model_element: An assembly model.

location: The location of the assembly model.

4.7.1.5 PART MODEL

This entity collects together the several types of information describing the complete shape of a piecepart.

Equivalent (subsidiary) descriptions of the part shape are allowed: for example the prefered nominal shape model might be a Brep Solid with subsidiary wire-frame and drafting models which describe different "application views" of the part shape.

```
*)
ENTITY part_model
  SUBTYPE OF (design_model);
  nominal_shape    : shape_model;
  model_units      : units;
  part_features    : OPTIONAL LIST [0 : #] OF form_feature;
  part_tolerances  : OPTIONAL LIST [0 : #] OF shape_tolerance;
  equivalents      : OPTIONAL LIST [0 : #] OF part_model_structure;
WHERE
  NOT (part_model IN equivalents.model_element);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

nominal_shape: The nominal shape of the part.

model_units: The (length) units in which the model is defined.

part_features: The features associated with the nominal part shape.

part_tolerances: The tolerances associated with the nominal part shape.

equivalents: Equivalent representations of the part.

PROPOSITIONS:

1. The list of **equivalents** must not include this instance of **part model**.

4.7.1.6 PART MODEL STRUCTURE

This entity associates a Local Coordinate System with a **part model**.

```
*)
ENTITY part_model_structure;
  model_element : part_model;
  location      : axis2_placement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

model_element: A **part model**.

location: The location of the **part model**.

4.7.1.7 Design Shape Classification Structure

The following indented list provides the classification structure for the Design Shape schema.

```
ASSEMBLY MODEL STRUCTURE
DESIGN MODEL
  ASSEMBLY MODEL
  PART MODEL
  LAYERED ELECTRICAL PRODUCT
PART MODEL STRUCTURE

*)
END_SCHEMA; -- end DESIGN SHAPE schema
(*
```

4.7.2 SHAPE MODEL

```
*)
SCHEMA ipim_nominal_shape_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_geometry_schema,
         ipim_topology_schema,
         ipim_solids_schema);
(*
```

A shape model is a representation of the nominal shape of a piecepart.

```
*)
ENTITY shape_model
  SUPERTYPE OF (solid_model XOR
                surface_model XOR
                wireframe_model XOR
                geometric_set);
END_ENTITY;
(*
```

4.7.2.1 SOLID MODEL

```
*)
SCHEMA ipim_solids_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_geometry_schema,
         ipim_topology_schema);
(*
```

A **solid model** is a complete representation of product nominal shape; any point can be classified as being inside, outside or on the boundary of a solid.

There are several different types of solid model representations.

```
*)
ENTITY solid_model
  SUPERTYPE OF (boolean_expression XOR
                csg_primitive XOR
                csg_solid XOR
                faceted_brep XOR
                half_space XOR
                manifold_solid_brep XOR
                solid_instance XOR
                swept_area_solid)
  SUBTYPE OF (shape_model);
END_ENTITY;
(*
```

4.7.2.2 MANIFOLD SOLID BOUNDARY REPRESENTATION

A manifold solid is an arcwise connected closed finite volume and the surface(s) of the solid is an arcwise connected orientable compact two-manifold. There is no restriction on the genus of the volume, nor on the number of voids within the volume.

The Boundary Representation (Brep) of a manifold solid utilises a graph of edges and vertices embedded in in a compact orientable two manifold surface. The embedded graph divides the surface into arcwise connected areas known as faces. The edges and vertices, therefore, form the boundaries of the faces and the extent of a face does not include its boundaries. The embedded graph may be disconnected and may be a pseudograph. The graph is labelled; each entity in the graph has a unique identity.

The geometric surface definition used to specify the geometry of a face must be two-manifold within the extent of the face. In other words, the geometric surface must neither self-intersect, overlap, nor have any holes or handles within the extent of the face.

The complete assemblage of the faces in the Brep forms a single arcwise connected closed volume and the surface(s) of the volume is two-manifold.

Faces do not intersect except along their boundaries. Each edge along the boundary of a face is shared by at most one other face in the assemblage. The assemblage of edges in the Brep do not intersect except at their boundaries (i.e vertices). The geometric curve definition used to specify the geometry of an edge must be arcwise connected and must not self intersect or overlap within the extent of the edge. The geometry of an edge must be conistent with the geometry of the faces of which it forms a (partial) bound.

The geometry used to define a vertex must be consistent with the geometry of the faces and edges of which it forms a (partial) bound.

A Brep is represented by one or more **closed shells** which must be disjoint. One shell, the outer, must completely enclose all the other shells and no other shell may enclose a shell.

The following version of the Euler formula must be satisfied

$$\chi_{ms} = \mathcal{V} - \mathcal{E} + 2\mathcal{F} - \mathcal{L}_l - 2(\mathcal{S} - \mathcal{G}^*) = 0 \qquad (10)$$

where $\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{L}_l$ and $\mathcal{S}$ are the numbers of unique vertices, edges, faces, loop uses and shells in the model and $\mathcal{G}^*$ is the sum of the genus of the shells.

More specifically, the topological entities must conform to the following constraints, where B denotes a manifold solid Brep:

- The shells must be unique

$$(B)[S] = (B)\{S\}$$

- Each face in the Brep is unique

$$((B)[S])[F] = ((B)[S])\{F\}$$

- Each loop is unique

$$(((B)[S])[F])[L] = (((B)[S])[F])\{L\}$$

- Each (edge + logical) pair is unique

$$((((B)[S])[F])[L])[E_l] = ((((B)[S])[F])[L])\{E_l\}$$

- Each edge in the Brep is either used by exactly two loops or twice by one loop

$$|((((B)[S])[F])[L])\{E_l\}| = 2|((((B)[S])[F])[L])[E_l]|$$

  That is, in the list $((((B)[S])[F])[L])[E]$ each edge appears exactly twice.

- Equation (10) must be satisfied

$$\begin{aligned} 2|(B)[S]| - 2\sum G^s \;=\; & |((((((B)[S])[F])\{L^e\})\{E\})\{V\}| + |((((B)[S])[F])\{L^v\})\{V\}| \\ & -|((((B)[S])[F])\{L\})\{E\}| + 2|((B)[S])[F]| - |(((B)[S])[F])[L]| \end{aligned}$$

```
*)
ENTITY manifold_solid_brep
  SUBTYPE OF (solid_model);
  outer     : shell_logical_structure;
  voids     : SET [0:?] OF shell_logical_structure;
WHERE
  dimensionality(manifold_solid_brep) = 3;
  {0 < extent(manifold_solid_brep) < infinity};
  arcwise_connected(manifold_solid_brep);
  closed(manifold_solid_brep);
  manifold(manifold_solid_brep);
  genus(manifold_solid_brep) >= 0;
  TYPEOF(brep_outer.shell_element) = closed_shell;
  (* the outer shell must be a closed shell *)
  NOT outer IN voids;
  constraints_topology_brep(manifold_solid_brep);
  constraints_geometry_brep(manifold_solid_brep);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

outer: A closed shell defining the exterior boundary of the solid.

voids: Set of closed shells defining voids within the solid. The set may be empty.

```
*)
RULE boundary_solid_and_loop FOR (manifold_solid_brep, loop);
  REPEAT FOR EACH loop IN manifold_solid_brep;
    IF TYPEOF(loop) = poly_loop THEN
      VIOLATION;
    (* The loops must not be POLY_LOOPs *)
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. Constraints topology brep returns TRUE where the function evaluates the topological constraints on the Brep.
2. Constraints geometry brep returns TRUE where the function evaluates the geometric constraints on the Brep.
3. The solid must have a defined outer boundary.
4. All shells in the solid must be closed.
5. Except for the outer shell, no shell can contain another.
6. The shells of the solid must be disconnected.
7. The faces of the shells of a Brep must not contain poly loops.

4.7.2.3 FACETTED MANIFOLD BOUNDARY SOLID MODEL

The facetted brep has been introduced in order to support the large number of systems that allow boundary type solid representations with planar surfaces only. Facetted models may be represented by MANIFOLD SOLID BREP but their representation as a facetted brep will be more compact. Unlike the Brep model, edges and vertices are not represented explicitly in the model but are implicitly available through the poly loop entity. A facetted brep has to meet the same topological constraints as the manifold solid brep.

```
*)
ENTITY facetted_brep
  SUBTYPE OF (solid_model);
  outer     : shell_logical_structure;
  voids     : SET [0:?] OF shell_logical_structure;
WHERE
  dimensionality(facetted_brep) = 3;
  (0 < extent(facetted_brep) < infinity);
  arcwise_connected(facetted_brep);
  closed(facetted_brep);
  manifold(facetted_brep);
```

```
  genus(facetted_brep) >= 0;
  TYPEOF(facetted_outer.shell_element) = closed_shell;
  (* the outer shell must be a closed shell *)
  NOT outer IN voids;
  constraints_topology_brep(facetted_brep);
  constraints_geometry_brep(facetted_brep);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

outer: A closed shell defining the exterior boundary of the solid.

voids: Set of closed shells defining voids within the solid. The set may be empty.

```
*)
RULE facetted_brep_and_loop FOR (facetted_brep, loop);
  REPEAT FOR EACH loop IN facetted_brep;
    IF TYPEOF(loop) <> poly_loop THEN
      VIOLATION;
    (* The loops must be POLY_LOOPs *)
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. Constraints topology brep returns TRUE where the function evaluates the topological constraints on the Brep.
2. Constraints geometry brep returns TRUE where the function evaluates the topological constraints on the Brep.
3. The solid must have a defined outer boundary.
4. All shells in the solid must be closed.
5. Except for the outer shell, no shell can contain another.
6. The shells of the solid must be disconnected.
7. The faces of the shells of a facetted Brep must only contain poly loops.

4.7.2.4 Constructive Solid Geometry Model

4.7.2.5 Introduction

This Section provides the CSG related portion of the Integrated Product Information Model.

The constructive solid geometry (CSG) section contains standard formats for one of the two most widely used solid model representations — CSG. The other representation is boundary representation.

The entities in this section can be thought of as one to two types — geometric or structural. The geometric entities are SOLIDs or volumetric primitives. These primitives include a block, wedge,

cylinder, cone, sphere, torus, and "swept solids". The model information for a primitive contains dimensions that define the shape of the primitive and point and direction coordinates that define a local coordinate system for the primitive.

The structural entities are the **csg solid** and **boolean expression**. The **csg solid** entity consists of references to the elements of the tree and operations such as union, difference, and interesection to be performed on these elements. Elements may be any type of **solid**.

The description of a STEP CSG solid model is an acyclic directed graph. The nodes in the graph are the various geometric and structural entities. This type of graph is like a tree structure, except that the branches of this graph may reconvene as a move is made down the graph, where down means in the general direction from root to terminal node. There may be any number of root nodes, which represent the actual solid models. A root may even be within the branches of anothers root's graph.

The terminal nodes are the primitives — the geometric entities — while all the other nodes are structural entitries. The structural entities are all able to point to each of the other structural entities as well as to primitives. In this way, by appropriately combining the geometric entities with the structural entities to create a graph structure, a CSG solid model is represented in STEP.

4.7.2.6 CSG SOLID

A **csg solid** is a solid made by combining simpler solids using regularized Boolean operations. The allowed operations are intersection, union, and difference. A regularized operation is defined as the closure of the interior of the result of the Boolean operation.

The **csg solid** entity serves two functions. First it identifies the root Boolean operation of a Boolean expression. Second it is the means to differentiate intermediate results and significant results. The result of the Boolean operation pointed to by a **csg solid** entity is significant and should be kept; otherwise, the result is only an intermediate result.

```
*)
ENTITY csg_solid
  SUBTYPE OF (solid_model);
  tree_expression : boolean_expression;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

tree_expression: Boolean expression of primitives and regularized operators describing the solid.

4.7.2.7 BOOLEAN EXPRESSION

A **boolean expression** expresses the operands and their operations which, together, serve to define a CSG **solid**. An operand may be either a **boolean term** or another **solid**.

```
*)
ENTITY boolean_expression
  SUPERTYPE OF (boolean_term)
  SUBTYPE OF (solid_model);
END_ENTITY;
(*
```

4.7.2.8 BOOLEAN-TERM

A boolean term is a regularized operation on two solids to create a new solid. Valid operations are regularized union, regularized intersection, and regularized difference. For purposes of Boolean operations, a solid is a regularized set of points.

```
*)
ENTITY boolean_term
  SUPERTYPE OF (union XOR
                intersection XOR
                difference)
  SUBTYPE OF (boolean_expression);
END_ENTITY;
(*
```

4.7.2.9 BOOLEAN OPERAND

```
*)
TYPE boolean_operand = SELECT
     (solid_model,
      shape_instance);
END_TYPE;
(*

*)
RULE boolean_operation_shape_instance FOR (boolean_operation);
  IF union IN TYPEOF(boolean_operation) OR
     intersection IN TYPEOF(boolean_operation) THEN
    IF NOT solid_model IN TYPEOF(first_operand) THEN
      VIOLATION;
    END_IF;
    IF NOT solid_model IN TYPEOF(second_operand) THEN
      VIOLATION;
    END_IF;
  END_IF;
  IF difference IN TYPEOF(boolean_operation) THEN
    IF NOT solid_model IN TYPEOF(work_piece) THEN
      VIOLATION;
    END_IF;
    IF NOT solid_model IN TYPEOF(to_be_removed) THEN
      VIOLATION;
    END_IF;
  END_IF;
END_RULE;
(*
```

4.7.2.10 UNION

Union on two solids is the new solid that contains all the points that are in either the first operand or the second operand or both.

```
*)
ENTITY union
  SUBTYPE OF (boolean_term);
  first_operand  : boolean_operand;
  second_operand : boolean_operand;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

first_operand: The first of a pair of operand solid models.

second_operand: The second of a pair of operands solid models.

4.7.2.11 INTERSECTION

Intersection on two solids is the new solid that is the regularization of the set of all points that are in both the first operand and the second operand.

```
*)
ENTITY intersection
  SUBTYPE OF (boolean_term);
  first_operand  : boolean_operand;
  second_operand : boolean_operand;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

first_operand: The first of a pair of solid model operands.

second_operand: The second of a pair of solid model operands.

4.7.2.12 DIFFERENCE

Difference on two solids is the regularization of the set of all the points in work piece, but not in the to be removed.

```
*)
ENTITY difference
  SUBTYPE OF (boolean_term);
  work_piece     : boolean_operand;
  to_be_removed  : boolean_operand;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

work_piece: Solid model from which material is to be removed.

to_be_removed: Solid model defining material to be removed.

4.7.2.13 SOLID INSTANCE

A solid instance is a copy of another solid at a new location.

```
*)
ENTITY solid_instance
  SUBTYPE OF (solid_model);
  solid_to_be_copied : solid_model;
  location           : axis2_placement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

solid_to_be_copied: Solid model which is the object of the **solid instance.**

location: Location and orientation at which the **solid instance** takes place.

PROPOSITIONS:

1. A solid may be instanced zero or more times

4.7.2.14 CSG PRIMITIVE

A **csg primitive** is a collecter for all CSG primitives. A primitive is a simple solid used in the construction of more complex solids using regularized operations. The **csg primitive** contains a location attribute which is common to and inherited by each of its subtypes.

```
*)
ENTITY csg_primitive
  SUPERTYPE OF (sphere XOR
                primitive_with_one_axis XOR
                primitive_with_axes)
  SUBTYPE OF (solid_model);
END_ENTITY;
(*
```

4.7.2.15 SPHERE

A sphere is defined by a center and a radius.

```
*)
ENTITY sphere
  SUBTYPE OF (csg_primitive);
  radius : real;
  center : point;
WHERE
    radius > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

radius: The radius of the sphere.

center: The location of the center of the sphere.

PROPOSITIONS:

1. Radius must be positive.

4.7.2.16 PRIMITIVE WITH ONE AXIS

A primitive with one axis entity collects all primitives with a single axis.

```
*)
ENTITY primitive_with_one_axis
  SUPERTYPE OF (right_circular_cone XOR
                right_circular_cylinder XOR
                torus)
  SUBTYPE OF (csg_primitive);
END_ENTITY;
(*
```

4.7.2.17 RIGHT CIRCULAR CONE

A right circular cone is defined by an axis, a point on the axis, the semi-angle of the cone, and a distance giving the location in the negative direction along the axis from the point to the base of the cone. In addition, a radius is given, which, if non-zero, gives the size and location of a truncated face of the cone.

```
*)
ENTITY right_circular_cone
  SUBTYPE OF (primitive_with_one_axis);
  semi_angle : real;
  radius     : real;
  position   : axis1_placement;
  height     : real;
WHERE
  {0 < semi_angle < 90};
  radius >= 0;
  height > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

semi_angle: One half the angle of the cone.

radius: The radius of the cone at the axis point. If the radius is zero, the cone has an apex. If the radius is greater than zero, the cone is truncated.

position: The location of a point on the axis and the direction of the axis.

height: The distance between the base of the cone and the point at which the radius is measured.

PROPOSITIONS:

1. The semi angle must be between 0° and 90°.
2. The radius must be non-negative.
3. The height must be positive.

4.7.2.18 RIGHT CIRCULAR CYLINDER

A right circular cylinder is defined by an axis point at the center of one circular cylinder face, an axis, a height, and a radius. The faces are perpendicular to the axis and are circular discs with the specified radius. The height is the distance from the first circular face center in the positive direction of the axis to the second circular face center.

```
*)
ENTITY right_circular_cylinder
  SUBTYPE OF (primitive_with_one_axis);
  radius      : real;
  position    : axis1_placement;
  height      : real;
WHERE
  radius > 0;
  height > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

radius: The radius of the cylinder.

position: The location of a point on the axis and the direction of the axis.

height: The distance between bases of the cylinder.

PROPOSITIONS:

1. The radius must be positive.
2. The height must be positive.

4.7.2.19 TORUS

A torus is a solid primitive defined by sweeping the center of a circle (the generatrix) about a larger circle (the directrix). The directrix is defined by a location and direction.

```
*)
ENTITY torus
  SUBTYPE OF (primitive_with_one_axis);
  major_radius : real;
```

```
  minor_radius : real;
  position     : axis1_placement;
WHERE
  minor_radius > 0;
  major_radius > minor_radius;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

major_radius: The radius of the directrix.

minor_radius: The radius of the generatrix.

position: The location of a point on the axis and the direction of the axis.

PROPOSITIONS:

1. The minor radius must be positive.
2. The major radius must be greater than the minor radius.

4.7.2.20 PRIMITIVE WITH AXES

The primitive with axes entity collects those entities which require more than one axis to describe their orientation in space.

```
*)
ENTITY primitive_with_axes
  SUPERTYPE OF (right_angular_wedge XOR
                block)
  SUBTYPE OF (csg_primitive);
END_ENTITY;
(*
```

4.7.2.21 BLOCK

A block is a rectangular parallelepiped, defined with a location and local coordinate system. The block is specified by the positive lengths X, Y, and Z along the axes of the local coordinate system.

```
*)
ENTITY block
  SUBTYPE OF (primitive_with_axes);
  x        : REAL;
  y        : REAL;
  z        : REAL;
  position : axis2_placement;
WHERE
    x > 0;
    y > 0;
    z > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

x: The size of the block along the X axis.

y: The size of the block along the Y axis.

z: The size of the block along the Z axis.

position: The location and orientation of the axis system for the primitive.

PROPOSITIONS:

1. x must be positive.
2. y must be positive.
3. z must be positive.

4.7.2.22 RIGHT ANGULAR WEDGE

A **right angular wedge** is defined with a location and local coordinate system. A triangular/trapezoidal face lies in the plane defined by the local X and Y axes. This face is defined by positive lengths X and Y along the local X and Y axes, by the length LTX (if non-zero) parallel to the X axis at a distance Y from the origin, and by the line connecting the ends of the X and LTX segments. The remainder of the wedge is specified by the positive length Z along the Z axis which defines a distance through which the trapezoid or triangle is extruded. If LTX = 0, the wedge has five faces; otherwise, it has six faces.

```
*)
ENTITY right_angular_wedge
  SUBTYPE OF (primitive_with_axes);
  x          : REAL;
  y          : REAL;
  z          : REAL;
  ltx        : REAL;
  position   : axis2_placement;
WHERE
    x > 0;
    y > 0;
    z > 0;
    ltx >= 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

x: The size of the wedge along the X axis.

y: The size of the wedge along the Y axis.

z: The size of the wedge along the Z axis.

ltx: The distance in the positive X direction of the smaller surface of the wedge.

position: The location and orientation of the axis system for the primitive.

PROPOSITIONS:

1. x, y, and z must be positive.
2. ltx must be non-negative.

4.7.2.23 SWEPT AREA SOLID

The **swept area solid** entity collects the entities which are defined by a sweeping action on plane figures.

```
*)
ENTITY swept_area_solid
  SUPERTYPE OF (solid_of_revolution XOR
                solid_of_linear_extrusion)
  SUBTYPE OF (solid_model);
END_ENTITY;
(*
```

4.7.2.24 SOLID OF LINEAR EXTRUSION

A **solid of linear extrusion** is a solid defined by sweeping a planar **face**. The direction of translation is defined by a **direction** vector, and the length of the translation is defined by a distance **depth**. The planar face may have holes which will sweep into holes in the solid.

```
*)
ENTITY solid_of_linear_extrusion
  SUBTYPE OF (swept_area_solid);
  extruded_face      : face;
  extruded_direction : direction;
  depth              : real;
WHERE
  depth > 0;
  cross_product(perpendicular(extruded_face),
                extruded_direction) = NULL;
  planar(extruded_face);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

extruded_face: The **face** defining the are to be swept.

extruded_direction: The **direction** in which the face is to be swept.

depth: The distance the face is to be swept.

PROPOSITIONS:

1. **Depth** must not be greater than zero.
2. **Extruded direction** must be perpendicular to **extruded face**.
3. **Extruded face** must be planar.

4.7.2.25 SOLID OF REVOLUTION

A solid of revolution is a solid formed by revolving a planar face about an axis. The axis must be in the plane of the face and the axes must not intersect the interior of the face. The planar face may have holes which will sweep into holes in the solid.

```
*)
ENTITY solid_of_revolution
  SUBTYPE OF (swept_area_solid);
  axis              : axis1_placement;
  extruded_face     : face;
  angle             : REAL;
WHERE
  (0 < angle <= 360);
  co_planar(axis, extruded_face);
  NOT intersect(domain(extruded_face),axis);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

axis: Axis about which sweep will be made.

extruded_face: Face to be swept.

angle: Angle through which the sweep will be made.

PROPOSITIONS:

1. Angle must be between 0° and 360°.
2. Axis must lie in plane of extruded face.
3. The axis must not intersect the interior of the face.

4.7.2.26 HALF SPACE SOLID

A half space solid is defined by the half space which is the regular subset of the domain which lies on one side of an unbounded surface. The domain is an orthogonal box or all space (all space is the default). See box domain for domain definition. Which side of the surface is determined by the surface normals and the complement flag. If the complement flag is FALSE, then the subset is the one the normals point away from. If the COMPLEMENT FLAG is TRUE, then the subset is the one the normals point into.

For a valid half space solid, the surface must divide the domain into exactly two subsets. Also, within the domain the surface must be manifold and all the surface normals must point into the same subset.

```
*)
ENTITY half_space
  SUBTYPE OF (solid_model);
  base_surface      : surface;
  enclosure         : OPTIONAL box_domain;
```

```
  complement_flag : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

base_surface: Surface defining side of half space.

enclosure: Domain defining remainder of half space. Default is all space.

complement_flag: Flag defining side of surface on which material lies.

PROPOSITIONS:

4.7.2.27 BOX DOMAIN

A **box domain** is an orthogonal box which may be used to limit the domain of a **half space**. The **box domain** is specified by the coordinates of two diagonal corners, (X-MIN, Y-MIN, Z-MIN) and (X-MAX, Y-MAX, Z-MAX).

```
*)
ENTITY box_domain;
  x_min : real;
  y_min : real;
  z_min : real;
  x_max : real;
  y_max : real;
  z_max : real;
WHERE
  x_max > x_min;
  y_max > y_min;
  z_max > z_min;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

x_min, y_min, z_min: Coordinates of most negative corner of box.

x_max, y_max, z_max: Coordinates of most positive corner of box.

PROPOSITIONS:

1. The Maximum coordinate values must be greater than the respective Minimum values

4.7.2.28 Solids FUNCTION Definitions

4.7.2.28.1 CONSTRAINTS GEOMETRY BREP

This function evaluates the geometric constraints on a **facetted** or **manifold solid brep** and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_brep(brep: solid_model): LOGICAL;
  LOCAL
    result    : LOGICAL := TRUE;
    outer     : shell := brep.outer.shell_element;
    void_list : LIST [0 : ?] OF shell;
    void_set  : SET OF shell;
  END_LOCAL;
  IF (TYPEOF(brep) <> manifold_solid_brep) OR
     (TYPEOF(brep) <> facetted_brep) THEN
    result := FALSE;
  END_IF;
  IF brep.outer.flag <> agreement(outer, brep) THEN
    result := FALSE;
    (* Outer shell normal must agree with Brep normal *)
  END_IF;
  (* get list and set of void shells in Brep *)
  REPEAT FOR EACH shell_logical_structure IN brep.voids;
    void_list := void_list + shell_logical_structure.shell_element;
    IF (shell_logical_structure.flag <>
        agreement(shell_logical_structure.shell_element, brep)) THEN
      result := FALSE;
      (* void normals must also point away from solid *)
    END_IF;
  END_REPEAT;
  void_set := void_set + void_list;
  REPEAT FOR EACH shell IN void_set;
    IF NOT disjoint(shell, outer) THEN
      result := FALSE;
      (* voids must be disjoint from outer shell *)
    END_IF;
    IF NOT inside(outer, shell, brep) THEN
      result := FALSE;
      (* voids must be within the outer shell *)
    END_IF;
    REPEAT i := 1 to SIZEOF(void_list);
    IF shell <> void_list[i] THEN
      BEGIN
        IF NOT disjoint(shell, void_list[i]) THEN
          result := FALSE;
          (* voids must be disjoint *)
        END_IF;
        IF inside(shell, void_list[i], brep) THEN
          result := FALSE;
          (* a void shell cannot enclose another void shell *)
        END_IF;
      END;
    END_REPEAT;
  END_REPEAT;
```

```
  RETURN(result);
END_FUNCTION;
(*
```

4.7.2.28.2 CONSTRAINTS TOPOLOGY BREP

This function evaluates the topological constraints on a facetted or manifold solid brep and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_topology_brep(brep: solid_model): LOGICAL;
  LOCAL
    shell_list     : LIST [0 : #] OF closed_shell;
    shell_set      : SET OF closed_shell;
    face_list      : list_of_face;
    face_set       : set_of_face;
    loop_list      : list_of_loop;
    loop_set       : set_of_loop;
    edge_list      : list_of_edge;
    edge_set       : set_of_edge;
    edge_log_list  : list_of_edge_logical_structure;
    edge_log_set   : SET OF edge_logical_structure;
    vtx_set        : set_of_vertex;
    shell_genus    : INTEGER := 0;
    result         : LOGICAL := TRUE;
  END_LOCAL;
  IF (TYPEOF(brep) <> manifold_solid_brep) OR
     (TYPEOF(brep) <> facetted_brep) THEN
    result := FALSE;
  END_IF;
  shell_genus := 0;
  (* calculate set and list of shells and
     sum of shell genus of Brep *)
  shell_list := shell_list + brep.outer.shell_element;
  shell_genus := shell_genus + genus(brep.outer.shell_element);
  REPEAT FOR EACH shell_logical_structure IN brep.voids;
    shell_list := shell_list +
                  shell_logical_structure.shell_element;
    shell_genus := shell_genus +
                   genus(shell_logical_structure.shell_element);
  END_REPEAT;
  shell_set := shell_set + shell_list;
  (* check shell validity *)
  REPEAT FOR EACH shell IN shell_set;
    IF NOT constraints_topology_closed_shell(shell) THEN
      result := FALSE;
    END_IF;
  END_REPEAT;
  (* check shell uniqueness *)
```

```
  IF (shell_list <> shell_set) THEN
    result := FALSE;
  END_IF;
  (* calculate set and list of faces, loops etc. of Brep *)
  REPEAT FOR EACH shell IN shell_set;
    face_list := face_list + list_shell_faces(shell);
    loop_list := loop_list + list_shell_loops(shell);
    edge_list := edge_list + list_shell_edges(shell);
    vtx_set := vtx_set + set_shell_vertices(shell);
  END_REPEAT;
  face_set := face_set + face_list;
  loop_set := loop_set + loop_list;
  edge_set := edge_set + edge_list;
  (* check face uniqueness *)
  IF (face_list <> face_set) THEN
    result := FALSE;
  END_IF;
  (* check loop uniqueness *)
  IF loop_list <> loop_set THEN
    result := FALSE;
  END_IF;
  (* calculate set and list of edge/logicals in Brep *)
  REPEAT FOR EACH loop IN loop_set;
    edge_log_list := edge_log_list + list_loop_edge_logicals(loop);
  END_REPEAT;
  edge_log_set := edge_log_set + edge_log_list;
  (* check edge/logical uniqueness *)
  IF (edge_log_list <> edge_log_set) THEN
    result := FALSE;
  END_IF;
  (* Check if each edge is used twice by the loops of the Brep *)
  REPEAT FOR EACH edge IN edge_set;
    IF (occurs(edge, edge_list) <> 2) THEN
      result := FALSE;
    END_IF;
  END_REPEAT;
  (* check genus *)
  IF (genus(brep) < 0) THEN
    result := FALSE;
  END_IF;
  (* check Euler formula *)
  IF (SIZEOF(vtx_set) - SIZEOF(edge_set) + 2*SIZEOF(face_set) -
      SIZEOF(loop_list) - 2*(SIZEOF(shell_set) - shell_genus)
      <> 0) THEN
        result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.7.2.29 Solids Classification Structure

The following indented listing provides the classification structure for the Solids model.

```
BOX DOMAIN
SOLID MODEL
  BOOLEAN EXPRESSION
    BOOLEAN TERM
      DIFFERENCE
      INTERSECTION
      UNION
  CSG PRIMITIVE
    PRIMITIVE WITH AXES
      BLOCK
      RIGHT ANGULAR WEDGE
    PRIMITIVE WITH ONE AXIS
      RIGHT CIRCULAR CONE
      RIGHT CIRCULAR CYLINDER
      TORUS
    SPHERE
  CSG SOLID
  FACETTED BREP
  HALF-SPACE
  MANIFOLD SOLID BREP
  SOLID INSTANCE
  SWEPT AREA SOLID
    SOLID OF LINEAR EXTRUSION
    SOLID OF REVOLUTION

*)
END_SCHEMA; -- end SOLIDS schema
(*
```

4.7.2.30 SURFACE MODEL

Some mechanical part representations consist of collections of surfaces which do not necessarily form the complete boundary of a solid. Such a model can be represented by a collection of faces.

```
*)
ENTITY surface_model
  SUPERTYPE OF (shell_based_surface_model,
                face_based_surface_model)
  SUBTYPE OF (shape_model);
END_ENTITY;
(*
```

4.7.2.31 SHELL BASED SURFACE MODEL

A shell based surface model is described by a set of open shells of dimensionality 2. The shells must be disjoint except at edges and vertices but a face in one shell may overlap a face in another

shell, provided the face boundaries are identical. Coincident portions of shells must both reference the same faces, edges and vertices defining the coincident region.

```
*)
ENTITY shell_based_surface_model
  SUBTYPE OF (surface_model);
  sbsm_boundary  : SET [1:#] OF shell_logical_structure;
WHERE
  dimensionality(surface_model) = 2;
  (0 < extent(surface_model) < infinity);
  constraints_geometry_sb_surface_model
                                (shell_based_surface_model);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sbsm_boundary: The set of **open_shell** and logical flag combinations comprising the **surface model.**

PROPOSITIONS:

1. There must be at least one **open shell.**
2. An **open shell** may exist independently of a **surface model.**
3. The **shells** must not overlap or intersect except at common **faces, edges** or **vertices.**
4. The function **constraints geometry sb surface model** returns TRUE, where the function evaluates the geometric constraints.

4.7.2.32 FACE BASED SURFACE MODEL

A **face based surface model** is described by a set of **connected face sets** of dimensionality 2.
The **connected face sets** must be disjoint except at **edges** and **vertices** but a **face** in one set may overlap a **face** in another set, provided the face boundaries are identical. Coincident portions of the **connected face sets** must both reference the same **faces, edges** and **vertices** defining the coincident region.

```
*)
ENTITY face_based_surface_model
  SUBTYPE OF (surface_model);
  fbsm_faces : SET [1:#] OF connected_face_set;
WHERE
  constraints_geometry_fb_surface_model
                        (face_based_surface_model);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

fbsm_faces: The set of **connected face sets** comprising the **face based surface model.**

PROPOSITIONS: -

1. There must be at least one **connected face set.**
2. A **connected face set** may exist independently of a **surface model.**
3. The **connected face sets** must not overlap or intersect except at common **faces, edges** or **vertices.**
4. The function **constraints geometry fb surface model** returns TRUE where the function evaluates the geometric constraints.

4.7.2.33 WIREFRAME MODEL

A wireframe representation of a mechanical piecepart contains information only about the intersections of the surfaces forming the boundary of the part but does not contain information about the surfaces themselves.

```
*)
ENTITY wireframe_model
  SUPERTYPE OF (shell_based_wireframe_model OR
                edge_based_wireframe_model)
  SUBTYPE OF (shape_model);
END_ENTITY;
(*
```

4.7.2.34 SHELL BASED WIREFRAME MODEL

A **shell based wireframe model** is described by a graph of edges and vertices embedded in R^3. The graph may be disconnected. Within the graph the edges do not intersect except at their boundaries (i.e vertices).

The geometry associated with a vertex must be consistent with the geometry associated with any of the edges of which the vertex forms a boundary.

A **shell based wireframe model** is represented by one or more **shells** of dimensionality 0 or 1.

```
*)
ENTITY shell_based_wireframe_model
  SUBTYPE OF (wireframe_model);
  sbwm_boundary  : SET [1:#] OF shell;
WHERE
  dimensionality(sbwm_boundary) <= 1;
  (0 < extent(shell_based_wireframe_model) < infinity);
  constraints_geometry_sb_wf_model
                        (shell_based_wireframe_model);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sbwm_boundary: The set of vertex **shells** and/or wire **shells** comprising the **wireframe model.**

PROPOSITIONS:

1. There must be at least one **shell**.
2. A **shell** may exist independently of a **wireframe model**.
3. The **shells** must not overlap or intersect except at common **edges** or **vertices**.
4. Each **shell** must be either of type **vertex shell** or **wire shell**.
5. The function **constraints geometry sb wf model** returns TRUE, where the function evaluates the geometric constraints

4.7.2.35 EDGE BASED WIREFRAME MODEL

A **edge based wireframe model** is described by a graph of edges and vertices embedded in R^3. The graph may be disconnected. Within the graph the edges do not intersect except at their boundaries (i.e vertices).

The geometry associated with a vertex must be consistent with the geometry associated with any of the edges of which the vertex forms a boundary.

A **edge based wireframe model** is represented by one or more **connected edge sets** of dimensionality 1.

```
*)
ENTITY shell_based_wireframe_model
  SUBTYPE OF (wireframe_model);
  ebwm_boundary  : SET [1:#] OF connected_edge_set;
WHERE
  dimensionality(ebwm_boundary) = 1;
  {0 < extent(edge_based_wireframe_model) < infinity};
  constraints_geometry_eb_wf_model
                        (edge_based_wireframe_model);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

ebwm_boundary: The set of **connected edge sets** comprising the **edge based wireframe model**.

PROPOSITIONS:

1. There must be at least one **connected edge set**.
2. A **connected edge set** may exist independently of a **wireframe model**.
3. The **connected edge sets** must not overlap or intersect except at common **edges** or **vertices**.
4. The function **constraints geometry eb wf model** returns TRUE, where the function evaluates the geometric constraints

4.7.2.36 GEOMETRIC SET

This entity is intended for the transfer of models when a topological structure is not available.

```
*)
ENTITY geometric_set
  SUPERTYPE OF (geometric_2d_set XOR
                geometric_projective_set XOR
                geometric_3d_curve_set XOR
                geometric_3d_surface_set)
  SUBTYPE OF (shape_model);
END_ENTITY;
(*
```

4.7.2.37 GEOMETRIC 2D SET

A geometric 2d set is a collection of two dimensional geometry.

```
*)
ENTITY geometric_2d_set
  SUBTYPE OF (geometric_set);
  points : SET [0:#] OF point;
  curves : SET [0:#] OF curve;
WHERE
  coordinate_space(points) = 2;
  coordinate_space(curves) = 2;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

points:

curves:

PROPOSITIONS:

4.7.2.38 GEOMETRIC PROJECTIVE SET

A geometric projective set is a collection of projective views of an object.

```
*)
ENTITY geometric_projective_set
  SUBTYPE OF (geometric_set);
  projective_views : SET [1:#] OF projective_view;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

projective_views: The

4.7.2.39 GEOMETRIC 3D CURVE SET

A **geometric 3d curve set** is a collection of three dimensional **points** and **curves**.

```
*)
ENTITY geometric_3d_curve_set
  SUBTYPE OF (geometric_set);
  points : SET [0:#] OF point;
  curves : SET [0:#] OF curve;
WHERE
  coordinate_space(points) = 3;
  coordinate_space(curves) = 3;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

points:

curves:

PROPOSITIONS:

4.7.2.40 GEOMETRIC 3D SURFACE SET

A **geometric 3d surface set** is a collection of three dimensional **points**, **curves** and **surfaces**.

```
*)
ENTITY geometric_3d_surface_set
  SUBTYPE OF (geometric_set);
  points   : SET [0:#] OF point;
  curves   : SET [0:#] OF curve;
  surfaces : SET [0:#] OF surface;
WHERE
  coordinate_space(points) = 3;
  coordinate_space(curves) = 3;
  coordinate_space(surfaces) = 3;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

points:

curves:

surfaces:

PROPOSITIONS:

4.7.2.41 PROJECTIVE VIEW

A projective view defines the content, location and orientation of a 2D view of a 3D object.

```
*)
ENTITY projective_view;
  view             : geometric_2d_set;
  projection_plane : axis2_placement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

view:

projection_plane:

4.7.2.42 Nominal Model FUNCTION Definitions

4.7.2.42.1 CONSTRAINTS GEOMETRY SB SURFACE MODEL

This function evaluates the geometric constraints on a shell based surface model and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_sb_surface_model
         (m: shell_based_surface_model): LOGICAL;
  LOCAL
    shell_list : LIST [1 : #] OF shell;
    shell_set  : SET OF shell;
    ts         : shell;
    tf         : LOGICAL;
    result     : LOGICAL := TRUE;
  END_LOCAL;
  REPEAT FOR EACH shell_logical_structure IN m.sm_boundary;
    ts := shell_logical_structure.shell_element;
    IF TYPEOF(ts) <> open_shell THEN
      result := FALSE;
    (* a surface model is composed of OPEN SHELLs *)
    END_IF;
    tf := shell_logical_structure.flag;
    IF tf <> agreement(ts, m) THEN
      result := FALSE;
    (* shell normals must agree with model surface normal *)
    END_IF;
    shell_list := shell_list + ts;
  END_REPEAT;
  shell_set := shell_set + shell_list;
  (* check if shells unique *)
  IF shell_set <> shell_list THEN
    result := FALSE;
```

```
  END_IF;
  REPEAT FOR EACH shell IN shell_set;
    REPEAT i := 1 TO SIZEOF(shell_list);
      IF shell <> shell_list[i] THEN
        BEGIN
          IF (intersect(shell, shell_list[i]) <> NULL) THEN
            result := FALSE;
          (* the shells must not intersect *)
          END_IF;
          IF overlap(shell, shell_list[i]) THEN
          (* but they may overlap at common faces,
             edges and vertices *);
          END_IF;
        END;
      END_IF;
    END_REPEAT;
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.7.2.42.2 CONSTRAINTS GEOMETRY FB SURFACE MODEL

```
*)
FUNCTION constraints_geometry_fb_surface_model
         (m : face_based_surface_model) : LOGICAL;
  LOCAL
    face_list : LIST [1:#] OF face := list_fbsm_faces(m);
    face_set  : SET [1:#] OF face := set_fbsm_faces(m);
    result    : LOGICAL := TRUE;
  END_LOCAL;
  (* check face uniqueness *)
  IF (face_set <> face_list) THEN
    result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.7.2.42.3 CONSTRAINTS GEOMETRY SB WF MODEL

This function evaluates the geometric constraints on a shell based wireframe model and returns TRUE if they are satisfied.

```
*)
FUNCTION constraints_geometry_sb_wf_model
         (m: shell_based_wireframe_model): LOGICAL;
  LOCAL
    shell_list : LIST [1 : #] OF shell;
```

```
    shell_set  : SET OF shell;
    result     : LOGICAL := TRUE;
  END_LOCAL;
  REPEAT FOR EACH shell IN m.wf_boundary;
    IF (TYPEOF(shell) <> wire_shell) OR
       (TYPEOF(shell) <> vertex_shell) THEN
      result := FALSE;
    (* a wireframe model is composed of WIRE and VERTEX SHELLs *)
    END_IF;
    shell_list := shell_list + shell;
  END_REPEAT;
  shell_set := shell_set + shell_list;
  REPEAT FOR EACH shell IN shell_set;
    REPEAT i := 1 TO SIZEOF(shell_list);
      IF shell <> shell_list[i] THEN
        BEGIN
          IF (intersect(shell, shell_list[i]) <> NULL) THEN
            result := FALSE;
          (* the shells must not intersect *)
          END_IF;
          IF overlap(shell, shell_list[i]) THEN
          (* but they may overlap at common edges and vertices *);
          END_IF;
        END;
      END_IF;
    END_REPEAT;
  END_REPEAT;
  RETURN(result);
END_FUNCTION;
(*
```

4.7.2.43 Nominal Shape Classification Structure

The following indented list provides the classification structure for the Nominal Shape Representation Information Model.

```
PROJECTIVE VIEW
SHAPE MODEL
  GEOMETRIC SET
    GEOMETRIC 2D SET
    GEOMETRIC 3D CURVE SET
    GEOMETRIC 3D SURFACE SET
    GEOMETRIC PROJECTIVE SET
  SOLID MODEL
  SURFACE MODEL
    FACE BASED SURFACE MODEL
    SHELL BASED SURFACE MODEL
  WIREFRAME MODEL
    EDGE BASED WIREFRAME MODEL
    SHELL BASED WIREFRAME MODEL
```

```
*)
END_SCHEMA; -- end NOMINAL SHAPE schema
(*
```

4.8 Features

4.8.1 Form Features

4.8.1.1 Introduction

This section presents the conceptual model of form feature information for the Integrated Product Information Model (IPIM). It is titled the "Form Features Information Model" (FFIM).

A form feature is a stereotypical portion of a shape. Commonly referred to shapes are holes, pockets, walls, and ribs. Form features are desirable for simplifying the man/machine interface and for standardizing data structures to permit automated decision systems to interface directly with the part definition.

4.8.1.2 Purpose

The primary purpose of the FFIM is to develop a form feature representation scheme that will be useful for storing and using feature data. The FFIM is intended to be compatible with the other topical models. That is, it should be possible to merge the FFIM and any other topical model(s) into a single model that is correct in syntax, semantics, and substance. In particular, the FFIM combined with other shape-oriented models should form an integrated information model for toleranced nominal shape.

The FFIM is intended to be independent of product classes and user applications. The model does not exhaustively list the alternate groupings of the form feature parameters, but attempts to establish a base that allows as many viewpoints as there are applications. For any particular user application it is possible to define a group of standard form features that map to the FFIM. These mapping concepts are described further in the Viewpoint Section. The relationship between the FFIM and Standard Features (such as threads described by Standards-making organizations like ISO) is addressed in the Future Development section. Intentionally, the FFIM does not force user applications to use specific form features nor application-independent nomenclature.

There are two types of Form Feature representations:

1. Explicit Form Features are groupings of elements of the geometric model. In Explicit Form Features, the shape elements that are necessary to define the feature are listed explicitly. For example, an explicit pocket feature might be the identification of five constituent faces.

2. Implicit Form Features model shape information parametrically rather than geometrically. For example, a hole might be described by giving a diameter and center line rather than a cylindrical surface.

Explicit and implicit form features provide powerful data representations for applications. Explicit form features allow referencing of feature constituents, but if feature parameters such as hole depth are needed, then they must be derived. On the other hand, implicit form features provide direct access to such parameters. However, implicit form features do not presently allow constituents to be individually referenced.

4.8.1.3 Scope

The FFIM presently treats a form feature as a portion of the skin of a shape that:

1. conforms to some stereotypical pattern and

2. is considered a unit for some purpose.

The FFIM is concerned with shape. The FFIM does not attempt to limit the portions of skin that might be treated as features, the nomenclature that is applied to these, or the purposes for which they are created.

It is understood that a variety of nonshape information will be associated with form features — functionality, processes, surface finish, etc. However, the FFIM's role is not to provide for the non-shape data, but to provide shape representation information to which such non-shape data might be "attached".

Other scoping considerations are:

- In the shape model representation, the use of form features is optional. For example, a cylindrical passage in a shape would generally be considered a "hole". This does not imply any obligation to have a feature representation of the "hole".
- Information for explicit representations is treated quite generally in the FFIM. The majority of the FFIM is devoted to information for implicit (constructive, descriptive) representations of form features.
- Tolerance information is not covered in the FFIM since it is covered in the Shape Variation Tolerances Model. There has been a deliberate effort to make the FFIM compatible with the tolerance model.
- The model is limited to form features of individual, rigid objects. Flexibility, assembly interfaces, mechanism joints, and the like are not currently addressed.
- In the FFIM, a form feature is limited to being a portion of shape, rather than the entire shape of an object. It is not presently intended that "standard" parts such as screws or bars be regarded as form features.
- The FFIM is concerned with "macro shape" of features. This excludes surface finish, the wave pattern produced by some processes, burrs, thin coatings, etc.

4.8.1.4 Viewpoint

The FFIM represents form features in general terms, such as "along part sweep of a U-shaped profile". On the other hand, applications identify form features using application-specific nomenclature. For example, different applications might interpret the same U-shaped feature as a "slot", "groove", "channel", or "dado". There is a need for a bridge between the form features of the FFIM and application specific form features. Mechanisms for this bridge have not been developed but are under consideration.

4.8.1.5 Future Developments

Items to be addressed in the future are divided into two groups: near-term and long-term.

4.8.1.5.1 Near-Term Topics

1. Enumeration of Standard Implicit Features

   Based on entities which are already in the FFIM, there will be a development of an initial set of more specific form features. These specific form features are both standardized shapes such

as the threads on screws, pins on electrical plugs, and other common shapes. Standard feature shapes will be modeled using existing international standards and practices as guidelines.

2. Dimensionality Generalization

   The FFIM will be extended to handle form features that are defined as Dimensionality-3 (DIM-3) shape volume elements. This will allow easier coverage of volumetric features and will facilitate modelling internal voids. DIM-2 (skin) will be retained in the model; DIM-1 (curve) and DIM-0 (point) shape elements may be added.

3. Components of Implicit Form Features

   It is currently impossible to attach components to attributes of implicit form features. This is an obstacle to full use of implicit form features. A method of referring to the components is desireable.

4. Standard Elements of Commonly Used Form Features

   A structured grouping of commonly used form features and their elements will be developed. These structured entities will allow standard referencing of components of form features such as the bottom of a pocket. These developments are necessary to support applications such as process planning, generative N/C, inspection and others.

5. Gears and Splines

   Gears and Splines will be added as technical review is provided.

4.8.1.5.2 Long Term Developments

1. Standard Parts

   The inclusion of standard parts such as nuts and washers in this Standard is considered to be of great importance. It has not been decided which committee should address this issue.

2. Assembly Features

   Form feature relationships resulting from joints in assembled parts such as: glued joints, riveted joints, or welded joints will be considered by the Form Feature Committee.

3. Layers and Coating Features

   The idea of layer features such as those used for composites and electrical products will be considered by the Form Features Committee.

4.8.1.6 FORM FEATURES SCHEMA

```
*)
SCHEMA ipim_features_schema;

EXPORT EVERYTHING;

ASSUME (ipim_shape_interface_schema,
        ipim_nominal_shape_schema,
        ipim_geometry_schema,
        ipim_solids_schema,
        ipim_tolerances_schema);
```

```
(*
```

4.8.1.7 Form Features TYPE Definitions

4.8.1.7.1 BOUND TYPES

```
*)
TYPE bound_types = ENUMERATION OF
     (start_bound,
      end_bound,
      volumetric_bound);
END_TYPE;
(*
```

4.8.1.7.2 COORDINATE ENUMERATION

```
*)
TYPE coordinate_enumeration = ENUMERATION OF
     (x_coordinate,
      y_coordinate,
      z_coordinate);
END_TYPE;
(*
```

4.8.1.7.3 NECK DIRECTION TYPES

```
*)
TYPE neck_direction_types = ENUMERATION OF
     (neck_positive,
      neck_negative);
END_TYPE;
(*
```

4.8.1.7.4 ELL ORIENTATION TYPES

```
*)
TYPE ell_orientation_types = ENUMERATION OF
     (ell_positive,
      ell_negative);
END_TYPE;
(*
```

4.8.1.7.5 TAPER TYPES

```
*)
TYPE taper_types = ENUMERATION OF
     (taper_increasing,
      taper_decreasing);
```

```
END_TYPE;
(*
```

4.8.1.7.6 FEATURE END TYPES

```
*)
TYPE feature_end_types = ENUMERATION OF
     (initial_end,
      terminal_end);
END_TYPE;
(*
```

4.8.1.7.7 HANDS

```
*)
TYPE hands = ENUMERATION OF
     (left_hand,
      right_hand);
END_TYPE;
(*
```

4.8.1.7.8 THREAD FIT CLASSES

```
*)
TYPE thread_fit_classes = ENUMERATION OF
     (one_a,
      two_a,
      three_a,
      one_b,
      two_b,
      three_b);
END_TYPE;
(*
```

4.8.1.7.9 THREAD FORMS

```
*)
TYPE thread_forms = ENUMERATION OF
     (sharpv,
      unified,
      sixty_degree_stub,
      acme,
      square,
      stub_acme,
      buttress,
      knuckle,
      brit_std);
END_TYPE;
(*
```

4.8.1.7.10 THREADING DIRECTION TYPES

```
*)
TYPE threading_direction_types = ENUMERATION OF
     (thread_upto,
      thread_beyond);
END_TYPE;
(*
```

4.8.1.7.11 COUPLING SHAPE TYPES

```
*)
TYPE coupling_shape_types = ENUMERATION OF
     (concave_coupling,
      convex_coupling);
END_TYPE;
(*
```

4.8.1.7.12 BEND MEASUREMENT TYPES

```
*)
TYPE bend_measurement_types = ENUMERATION OF
     (concave_measurement,
      convex_measurement,
      center_measurement);
END_TYPE;
(*
```

4.8.1.7.13 TUBE BEND MOVED END TYPES

```
*)
TYPE tube_bend_moved_end_types = ENUMERATION OF
     (tube_forward_end,
      tube_rearward_end);
END_TYPE;
(*
```

4.8.1.7.14 SET OF PROFILE PAIRS

```
*)
TYPE set_of_profile_pairs = SET [0:?] OF profile_pair;
END_TYPE;
(*
```

4.8.1.8 Feature FUNCTION Definitions

4.8.1.8.1 CYLINDRICAL

```
*)
FUNCTION cylindrical(area:dimensionality_2_shape_element):LOGICAL;
```

```
-- determines if an area is cylindrical
END_FUNCTION;
(*
```

4.8.1.8.2 CONICAL

```
*)
FUNCTION conical(area:dimensionality_2_shape_element):LOGICAL;
-- determines if an area is conical
END_FUNCTION;
(*
```

4.8.1.8.3 PLANAR

```
*)
FUNCTION planar(pairs:set_of_profile_pairs):LOGICAL;
-- determines if a curve string is planar
END_FUNCTION;
(*
```

4.8.1.8.4 SEQUENCED

```
*)
FUNCTION sequenced(initial:curve;
                   subsequent:set_of_profile_pairs):LOGICAL;
-- determines if a GENERAL PROFILE is made up of correctly ordered
-- initial curve and subsequent curve pairs
END_FUNCTION;
(*
```

4.8.1.8.5 CLOSED

```
*)
FUNCTION closed(pairs:set_of_profile_pairs):LOGICAL;
-- determines if a curve string is closed
END_FUNCTION;
(*
```

4.8.1.9 FORM FEATURE

A portion of a shape that fits a pattern or stereotype. That is, it is an occurrence of some recognized shape configuration. The stereotype may be precise (constant diameter thru hole) or vague (web).

```
*)
ENTITY form_feature;
  feature_type  : STRING;
  implicit_reps : SET [0:#] OF implicit_form_feature;
  pattern_rep   : OPTIONAL implicit_form_feature_pattern;
  replicate_rep : OPTIONAL replicate_form_feature;
```

```
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

feature_type: A specification of the type of feature (e.g countersink)

implicit_reps: A collection of implicit representations of the feature

pattern_rep: An implicit form feature pattern representation of the feature

replicate_rep: A replicate form feature representation of the feature

4.8.1.10 IMPLICIT FORM FEATURE

A description of a form feature by parameters and/or geometric data. The description must be sufficient to "realize" the feature; i.e., to compute its shape and location and integrate these into a geometric model of the shape which contains the form feature. (For a surfaced wireframe model, for example, this means to determine the faces, loops, edges, and vertices that must be added to/subtracted from the model in order that the geometric model "include" the feature. An implicit representation might also be called "constructive" or "descriptive".

```
*)
ENTITY implicit_form_feature
SUPERTYPE OF (implicit_area_feature XOR
              implicit_deformation XOR
              implicit_depression XOR
              implicit_passage XOR
              implicit_protrusion XOR
              implicit_transition);
END_ENTITY;
(*
```

4.8.1.11 GEOMETRIC MODEL IMPLICIT FORM FEATURE ASSN

An indication that a representation of a form feature as an implicit form feature applies to a particular geometric model.

```
*)
ENTITY geometric_model_implicit_form_feature_assn;
  augmented_geometric_model         : geometric_model;
  augmenting_implicit_form_feature  : implicit_form_feature;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

augmented_geometric_model: A reference to the geometric model that is augmented by the application of an implicit form feature.

augmenting_implicit_form_feature: The implicit form feature that adds shape information to the shape model.

4.8.1.12 IMPLICIT-FEATURE PRECEDENCE

An indication of existence precedence between two implicit feature representations. This entity gives a predecessor/successor relation between implicitly represented form features. It implies that the predecessor feature is a part of the pre-existing shape to which the successor is an increment. Note that, functionally, the **implicit feature bound** mechanism partially overlaps this entity in the sense that a "bounder"/"boundee" relationship implies a predecessor/successor relation. Only one of the two modeling tactics should be used for a given case.

```
*)
ENTITY implicit_feature_precedence;
  predecessor_feature : implicit_form_feature;
  successor_feature   : implicit_form_feature;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

predecessor_feature: The **implicit form feature** that has a higher precedence than the **successor**, i.e., must be installed or constructed first.

successor_feature: The **implicit form feature** that has a lower precedence than the **predecessor**, i.e., must be installed or constructed second.

4.8.1.13 IMPLICIT FEATURE BOUND

A bound or limit to the extent of an **implicit form feature**. These are used where implicit feature representations define volumes added to or subtracted from preexisting material (passages, depressions, protrusions). They indicate where the preexisting material's volume and the volume specified by the implicit feature representation intersect and thus serve to reduce the material to be added or subtracted to "realize" the feature. For example, consider a thru hole modeled as an implicit passage. The implicit representation of the hole might be by centerline and diameter, effectively defining an infinite cylindrical volume. **Implicit feature bounds** could be used to specify entry and exit faces, thereby specifying the limits of the volume to be removed.

```
*)
ENTITY implicit_feature_bound;
  elements : SET [1:#] OF dimensionality_2_shape_element;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

elements: The bounding elements

4.8.1.14 IMPLICIT PASSAGE

An **implicit form feature** that is viewed as being "subtracted" from preexisting shape and which intersects preexisting shape in two places; i.e., goes through the shape. The feature increases the genus of the shape by 1 or more. Its surface elements generally intersect those of the preexisting shape at convex solid angles (0 to 180 degrees).

```
*)
ENTITY implicit_passage
  SUBTYPE OF (implicit_form_feature);
  definition       : feature_volume;
  end_bounds       : SET [0:2] OF passage_bound;
  boundary_blends  : SET [0:2] OF passage_boundary_blend;
  interruptions    : SET [0:#] OF passage_intermediate_bound;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The defining feature sweep or ruling.

end_bounds: The description of the desired end bounds for the feature.

boundary_blends: A description of the implicit blends between the feature and its ends.

interruptions: A description of any bounds that define discontinuities over the feature length.

4.8.1.15 PASSAGE BOUND

An indication that an **implicit feature bound** describes the preexisting area where an **implicit passage** enters/exits material.

```
*)
ENTITY passage_bound;
  specification : implicit_feature_bound;
  passage_end   : feature_end_types;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

specification: The definition of the feature bound.

passage_end: The indication of which end is being bounded.

4.8.1.16 PASSAGE BOUNDARY BLEND

A blend at one of the openings of an **implicit passage.**

For a swept passage, the defining **feature sweep** provides the directionality needed for **blended end** to make sense. For a sweep with a **linear feature sweep path**, the **entry** end of the feature is at $Z = 0$. The same holds for an **axisymmetric feature sweep**. For a sweep with a **partial circular feature sweep path**, the **entry** end of the feature is in the positive XZ quadrant.

```
*)
ENTITY passage_boundary_blend;
  blended_end : feature_end_types;
  end_blend   : implicit_edge_blend;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

blended_end: An indication of the end that is blended, initial or terminal.

end_blend: The definition of the blend desired.

4.8.1.17 PASSAGE INTERMEDIATE BOUND

A bound or limit to the extent of an **implicit passage**, not occurring at the passage ends. This indicates pre-existing air (or a preexisting air-material interface) contained within the volume implied by the implicit specification of the passage. For example, a hole through both flanges of an I-beam is interrupted by the void between them. Depending on the context, the interruption may be viewed as resulting from an intervening pre-existing volume or an intervening surface. In the I-beam example, if the void is modeled as a swept form feature then the hole would be viewed as being interrupted by a volume. On the other hand, if the flanges were explicitly modeled, then the hole would be seen as having two planar intermediate bounds.

```
*)
ENTITY passage_intermediate_bound;
  passage_bound_type     : bound_types;
  interruption_complete  : LOGICAL;
  specification          : implicit_feature_bound;
  bound_blend            : OPTIONAL implicit_edge_blend;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

passage_bound_type: An indication of whether the bound is seen as two or three dimensional. Values are **start bound** or **end bound** for two dimensional bounds, and **volumetric bound** for three dimensional bounds. For a two dimensional bound, the direction of the feature as determined by its path definition is used to label the bound as a start or end.

interruption_complete: An indication of whether the interruption is complete or incomplete. A complete interruption effects the entire profile, rather than only part of it.

specification: The definition of the bound.

bound_blend: The definition of a bound blend.

4.8.1.18 IMPLICIT PROTRUSION

An **implicit form feature** that extends outward from "the rest of the shape"; i.e., is viewed as being "added" to preexisting shape. The feature does not change the genus of the shape. Its surface elements generally intersect those of the preexisting shape at concave solid angles (180 to 360 degrees).

```
*)
ENTITY implicit_protrusion
  SUBTYPE OF (implicit_form_feature);
  definition : feature_volume;
  end_bound  : OPTIONAL implicit_feature_bound;
```

```
  end_blend  : OPTIONAL implicit_edge_round;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The defining feature sweep or ruling.

end_bound: A description of the desired bound or end for the feature.

end_blend: A blend around the perimeter of the protrusion.

PROPOSITIONS:

1. The addition of a protrusion to a shape does not change the genus of the shape

4.8.1.19 IMPLICIT DEPRESSION

An **implicit form feature** that extends inward from "the rest of the shape"; i.e., is viewed as being "subtracted" from preexisting shape. The feature does not change the genus of the shape; i.e., does not go through the shape. Its surface elements generally intersect those of the preexisting shape at convex solid angles (0 to 180 degrees).

```
*)
ENTITY implicit_depression
  SUBTYPE OF (implicit_form_feature);
  definition    : feature_volume;
  end_bound     : OPTIONAL implicit_feature_bound;
  end_blend     : OPTIONAL implicit_edge_blend;
  interruptions : SET [0:#] OF depression_intermediate_bound;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The defining feature sweep or ruling.

end_bound: A definition of the desired ends for the feature.

end_blend: A blend around the perimeter of the depression and a pre-existing shape.

interruptions: An indication of whether the interruption is complete or incomplete. A complete interruption effects the entire profile, rather than only part of it.

PROPOSITIONS:

1. The addition of the feature to the part leaves the genus of the part unaltered.

4.8.1.20 DEPRESSION INTERMEDIATE BOUND

A bound or limit to the extent of an **implicit depression**, not occurring at the open end of the depression. **Depression intermediate bounds** are analogous to **passage intermediate bounds**.

```
*)
ENTITY depression_intermediate_bound;
  depression_bound_type : bound_types;
  interruption_complete : LOGICAL;
  specification         : implicit_feature_bound;
  bound_blend           : OPTIONAL implicit_edge_blend;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

depression_bound_type: An indication of whether the bound is seen as two or three dimensional. Values of **start bound** and **end bound** are for two dimensional bounds, and **volumetric bound** for three dimensional bounds. For a two dimensional bound, the direction of the feature as determined by its path definition is used to label the bound as a start or end.

interruption_complete: An indication of whether the interruption is complete or incomplete. A complete interruption effects the entire profile, rather than only part of it.

specification: The specification of the bound.

bound_blend: The definition of the blend between the feature and its bounds.

4.8.1.21 IMPLICIT TRANSITION

An implicit representation of a **form feature** which connects two elements of preexisting shape. As a rule, the transition is regarded as less important than the elements it connects. Fillets and chamfers steps are examples.

```
*)
ENTITY implicit_transition
  SUPERTYPE OF (implicit_corner_blend XOR
                implicit_edge_blend)
  SUBTYPE OF (implicit_form_feature);
END_ENTITY;
(*
```

PROPOSITIONS:

1. The addition of the feature to the part leaves the part genus unchanged

4.8.1.22 IMPLICIT EDGE BLEND

An implicit representation of a blend or transition between two adjacent surface areas of a shape.

```
*)
ENTITY implicit_edge_blend
  SUPERTYPE OF (implicit_edge_flat XOR
                implicit_edge_round)
  SUBTYPE OF (implicit_transition);
END_ENTITY;
(*
```

4.8.1.23 EDGE BLENDED INTERSECTION

An indication that an **implicit edge blend** applies to the intersection of two **dimensionality 2 shape elements**.

```
*)
ENTITY edge_blended_intersection;
  blend        : implicit_edge_blend;
  first_shape  : dimensionality_2_shape_element;
  second_shape : dimensionality_2_shape_element;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

blend: With the following two attributes, identifies the **implicit edge blend**.

first_shape: With the following attribute, identifies the first of the two **dimensionality 2 shape elements** whose intersection is blended.

second_shape: Identifies the second of the two **dimensionality 2 shape elements** whose intersection is blended.

4.8.1.24 IMPLICIT EDGE FLAT

An implicit representation of a ruled surface blend of two surface areas of a shape. The blend surface is set back a constant distance along one of the areas from the intersection of the two surface areas and makes a constant angle with that area. Most often, the blend surface is planar (blending planar surfaces) or conical (blending rotational surfaces).

Angle and **setback** are measured relative to one of the two elements blended by the chamfer. The following rules tell which, depending on the context of the feature:

1. Related to **edge blended intersection**:

   angle and **setback** are relative to the first of the two elements associated via the **edge blended intersection**.

2. Related to **passage boundary blend** or **depression boundary blend**:

   angle and **setback** are relative to the boundary element of the passage or depression; i.e., the preexisting shape.

3. For any **feature sweep profile** not utilizing **general profile**:

   Consider the segments of the profile to be clockwise ordered when looking in the direction of sweep **angle** and **setback** are relative to the first of the two profile segments blended.

4. Related to **profile sequencer**; i.e., a blend between segments of a **general profile**:
   angle and **setback** are relative to the **predecessor** element in **profile pair**.

5. Used as a wall-end blend in an **in out feature sweep wall end**, as a flat end blend in an **along feature sweep flat end**, or a wall-end blend in an **axisymmetric feature sweep wall end**:
   angle and **setback** are relative to the swept walls of the feature.

```
*)
ENTITY implicit_edge_flat
  SUBTYPE OF (implicit_edge_blend);
  angle    : angle_parameter;
  setback  : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

angle: the constant angle between the flat and the blended surface used for nominal shape definition.

setback: The constant distance between (i) the intersection of the two blended surfaces and (ii) the intersection of the flat and the blended surface used for nominal shape definition.

4.8.1.25 IMPLICIT EDGE ROUND

A blend of two surface areas of a shape, having a circular cross section of constant radius. The blend surface is tangent to both of the adjacent surface areas. Most often, the blend surface is cylindrical (blending two planar surfaces) or toroidal (blending rotational surfaces).

```
*)
ENTITY implicit_edge_round
  SUBTYPE OF (implicit_edge_blend);
  round_dim : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

round_dim: The dimension (radius, preferably, or diameter) of the round.

4.8.1.26 IMPLICIT CORNER BLEND

An implicit representation of a form feature that smooths or "gradualizes" a corner of a shape.

```
*)
ENTITY implicit_corner_blend
  SUPERTYPE OF (implicit_corner_flat XOR
                implicit_outside_corner_round)
  SUBTYPE OF (implicit_transition);
END_ENTITY;
(*
```

4.8.1.27 IMPLICIT-CORNER FLAT

An implicit representation of a planar form feature that smooths or "gradualizes" a corner of a shape.

```
*)
ENTITY implicit_corner_flat
  SUBTYPE OF (implicit_corner_blend);
  setback_edge1 : edge_shape_element;
  setback_dim1  : size_parameter;
  setback_edge2 : edge_shape_element;
  setback_dim2  : size_parameter;
  angle_dim     : angle_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

setback_edge1: Identifies one of the edges.

setback_dim1: The setback (measured linearly) from the corner along the first edge. (While half dimension is possible, it seems unuseful.)

setback_edge2: Identifies the other edge.

setback_dim2: The setback (measured linearly) from the corner along the second edge. (While half dimension is possible, it seems unuseful.)

angle_dim: The angle between the planar area and the flat. (While half dimension is possible, it seems unuseful.)

PROPOSITIONS:

1. The two EDGE SHAPE ELEMENTs must be adjacent boundaries, meeting at the CORNER SHAPE ELEMENT associated with the parent IMPLICIT CORNER BLEND of the feature, of a planar AREA SHAPE ELEMENT which is one of the areas that meet at the CORNER SHAPE ELEMENT.

4.8.1.28 IMPLICIT OUTSIDE CORNER ROUND

An implicit representation of a form feature which is a spherical rounding of a corner of a shape.

```
*)
ENTITY implicit_outside_corner_round
  SUBTYPE OF (implicit_corner_blend);
  dimension : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

dimension: The size dimension (radius, preferably, or diameter) of the spherical feature.

4.8.1.29 IMPLICIT-DEFORMATION

An implicit representation of a feature characterized by stretching or bending material.

```
*)
ENTITY implicit_deformation
  SUPERTYPE OF (implicit_bend XOR
                implicit_emboss XOR
                implicit_partial_cutout XOR
                implicit_tube_deformation XOR
                implicit_twist)
  SUBTYPE OF (implicit_form_feature);
END_ENTITY;
(*
```

4.8.1.30 BEND DIMENSION

The dimension of a bend, plus an indication whether the dimension applies to the concave side, convex side, or center of the bend.

```
*)
ENTITY bend_dimension;
  bend_dim    : size_parameter;
  bend_msrmt  : bend_measurement_types;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

bend_dim: The independent size parameter which gives the dimension value.

bend_msrmt: Whether the dimension is specified for the concave side of the bend, the convex side, or center.

4.8.1.31 IMPLICIT EMBOSS

An imprinting (rib or recess) that is totally surrounded by part material. Embossing is distinguished from bends and flanges by the fact that embossing is totally surrounded by the part material.

```
*)
ENTITY implicit_emboss
  SUPERTYPE OF (implicit_corner_rib XOR
                implicit_round_bead XOR
                implicit_v_bead XOR
                implicit_spherical_emboss)
  SUBTYPE OF (implicit_deformation);
END_ENTITY;
(*
```

4.8.1.32 IMPLICIT_V BEAD

A deformation that has the cross sectional characteristics of a V.

```
*)
ENTITY implicit_v_bead
  SUBTYPE OF (implicit_emboss);
  location    : bounded_curve;
  angle       : angle_parameter;
  height      : size_parameter;
  width       : size_parameter;
  apex_round  : bend_dimension;
  base_bend   : bend_dimension;
WHERE
  angle.dimension < 180;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Identifies the axis curve of the feature. The curve lies on the surface that is displaced to become the inner (concave) portion of the bead, so that points on the curve are displaced to the inner apex of the bead. Full bead profile begins and ends at the extrema of the axis bounded curve.

angle: The angle of the "V" formed.

height: The maximum displacement of points on the outer surface of the feature.

width: The width of the deformation, measured between points on the centerline of the bend at its base.

apex_round: The corner radius of the bead at the inner apex of the "V".

base_bend: The bend radius between the outer portion of the bead and its undeformed neighborhood.

PROPOSITIONS:

1. The angle must be less than 180 degrees.

4.8.1.33 IMPLICIT ROUND BEAD

An emboss that has the cross sectional characteristics of an arc of a circle.

```
*)
ENTITY implicit_round_bead
  SUBTYPE OF (implicit_emboss);
  location     : bounded_curve;
  height       : size_parameter;
  bead_size    : size_parameter;
  base_bend    : bend_dimension;
WHERE
```

```
  height.dimension <= bead_size.dimension/2;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Identifies the axis curve of the feature. The curve lies on the surface that is displaced to become the inner (concave) portion of the bead, so that points on the curve are displaced to the inner apex of the bead. Full bead profile begins and ends at the extrema of the axis bounded curve.

height: The height dimension of the bead. (Though half measurement is possible, full height from undeformed base to top is preferred.)

bead_size: The size (diameter or radius) of the circular bead cross-section.

base_bend: The dimension (radius, preferably, or diameter) of the bend between the bead and its neighboring, undeformed material, measured on the outer side of the bead.

PROPOSITIONS:

1. The height must not be greater than half of the bead size.

4.8.1.34 IMPLICIT CORNER RIB

A stiffening rib that is created by inverting the material, creating a protruding region at a location along a bend. The feature has rectangular surface shape and is planar except for bends at its intersection with the two stiffened surfaces.

The rib is located via a Local Coordinate System. The Z-axis is the length centerline of the outer rectangular surface of the rib. The origin is placed at the intersection of the Z-axis with the inner surface of one of the legs of the stiffened bend (the "first leg"), with the X-axis lying in that surface. Thus, ignoring bend radii of the feature, the outer surface of the rib is a sweep of the X-interval [-width/2, width/2] in the +Z direction.

```
*)
ENTITY implicit_corner_rib
  SUBTYPE OF (implicit_emboss);
  stiffens : form_feature;
  location : axis_placement;
  width    : size_parameter;
  height1  : derivable_dimension;
  height2  : derivable_dimension;
  length   : derivable_dimension;
  bend1    : bend_dimension;
  bend2    : bend_dimension;
WHERE
  height1.dimension**2 + height2.dimension**2 = length.dimension**2;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

stiffens: The form feature that the rib applies to.

location: The location and orientation of the rib.

width: The width dimension of the rib.

height1: The derivable height of the rib along the first (per the rule below for placement of the feature's local coordinate system) leg of the stiffened bend. (It is possible to give half-height, but full value is recommended.)

height2: The derivable height of the rib along the second (per the rule below for placement of the feature's local coordinate system) leg of the stiffened bend. (It is possible to give half-height, but full value is recommended.)

length: The derivable length of the rib. (It is possible to give semi-length but full value is recommended.)

bend1: The dimension (radius, preferably, or diameter) of the bend between the feature and the first (per the rule below for placement of the feature's local coordinate system) leg of the stiffened bend.

bend2: The bend dimension (radius, preferably, or diameter) between the feature and the second (per the rule below for placement of the feature's local coordinate system) leg of the stiffened bend.

4.8.1.35 IMPLICIT SPHERICAL EMBOSS

An implicit representation of an emboss having spherical displacement of material.

```
*)
ENTITY implicit_spherical_emboss
  SUBTYPE OF (implicit_emboss);
  location     : point;
  height       : size_parameter;
  emboss_size  : size_parameter;
  bend_dim     : bend_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Identifies the point which locates the feature. The point is located on the inward-bent side of the sheetlike geometry, at the location that becomes the apex of the sphere. Material movement is normal to the sheetlike geometry at that location.

height: The height dimension of the emboss. May be half or, preferably, full height. Measured from undeformed base of feature.

emboss_size: The diameter or radius of the sphere.

bend_dim: The dimension (radius, preferably, or diameter) of the bend at the feature's intersection with undeformed material.

PROPOSITIONS:

1. The sphere radius must be greater or equal to the feature's height.

4.8.1.36 IMPLICIT-TWIST

A twist in the material about a centerline. An implicit twist is applied to a sheet or plate like portion of a shape.

The twist feature is located via a Local Coordinate System as follows:

1. The Z-axis must coincide with the twist axis. The origin must lie on the center of the twist; i.e., the twist region is the Z-interval [-length/2, length/2] (taking length to be full length). The XY-plane must be oriented so that the twist takes +X into +Y in the positive Z halfspace and +X into -Y in the negative Z halfspace.

2. The post-twist shape is calculated as follows. Values are given in local coordinates, but the transformations apply to the entire shape containing the twist.

   Put A = angle/2, L = length/2 and θ as the rotation at any position z.

   The formulas assume that angular displacement varies linearly with z in the interval $[-L, L]$, that the angular displacement at $z >= L$ is A, and that the angular displacement at $z <= -L$ is $-A$.

$$\theta = \begin{cases} zA/L & \text{for } z \text{ in } [-L, L] \\ A & \text{for } z > L \\ -A & \text{for } z < -L \end{cases}$$

$$x' = x\cos\theta - y\sin\theta$$

$$y' = x\sin\theta + y\cos\theta$$

$$z' = z$$

```
*)
ENTITY implicit_twist
  SUBTYPE OF (implicit_deformation);
  length   : size_parameter;
  angle    : angle_parameter;
  location : axis_placement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

length: The length of the twist region.

angle: The angle to the original which the twist results in.

location: The establishment of a local coordinate system for the twist.

4.8.1.37 IMPLICIT PARTIAL CUTOUT

An implicit deformation which involves shearing as well as deformation.

```
*)
ENTITY implicit_partial_cutout
  SUPERTYPE OF (implicit_louver XOR
                implicit_circular_knockout XOR
```

```
                - implicit_tab)
  SUBTYPE OF (implicit_deformation);
END_ENTITY;
(*
```

4.8.1.38 IMPLICIT LOUVER

An implicit partial cutout for which the shear is a straight line and the deformation to the material causes an opening to be created. Deformation at the shear line is greater than material thickness.

The feature is located via a Local Coordinate System. This is done as follows. The origin is placed at the center of the shear line on the sheet side that is bent inward. The shear line lies on the Z-axis. Deformation is in the +Y-direction, so that (0,height,0) is the maximum height point of the feature and is the displaced position of the local origin. Deformation occurs in the 0 < x < width portion of the XZ-plane.

```
*)
ENTITY implicit_louver
  SUBTYPE OF (implicit_partial_cutout);
  location    : axis_placement;
  length      : size_parameter;
  width       : size_parameter;
  height      : size_parameter;
  base_bend   : bend_dimension;
  inner_bend  : bend_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Identifies the axis placement which locates the feature.

length: Length dimension of louver.

width: Width dimension of louver.

height: Height dimension of louver

base_bend: Dimension (radius, preferably, or diameter) of bend at intersection with undeformed material.

inner_bend: Dimension (radius, preferably, or diameter) of bend within louver.

4.8.1.39 IMPLICIT CIRCULAR KNOCKOUT

A circular implicit partial cutout where the deformation to the material causes no opening to be created. Deformation at the shear line is less than material thickness. The shear line has equally spaced gaps of equal length.

The feature is located via a Local Coordinate System. This is done as follows. The origin is placed at the center of the shear circle, lying on the shear entry surface of the pre-shear shape. The Z-axis points in the direction of material movement. The gaps in the shear line are assumed to be equally spaced, with the X-axis bisecting one of the gaps.

```
*)
ENTITY implicit_circular_knockout
  SUBTYPE OF (implicit_partial_cutout);
  location    : axis_placement;
  size_dim    : size_parameter;
  height      : size_parameter;
  gap_length  : size_parameter;
  bend_dim    : bend_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: The location of the feature.

size_dim: The size (diameter or radius) of the shear circle.

height: The height dimension of the protrusion of the feature from the material. (Semi-height is possible, but full height is preferred.)

gap_length: The space between adjacent shears, measured chordally. (Semi-length is possible, but full length is preferred.)

bend_dim: The bend dimension (radius, preferably, or diameter) between the feature and its neighborhood.

PROPOSITIONS:

1. The full height must be less than the thickness of the sheet-like environment.

4.8.1.40 IMPLICIT TAB

An implicit representation of a partial cutout where the shear is an arc of a circle and the deformation to the material causes an opening to be created.

The feature is located via a Local Coordinate System. This is done as follows. The origin must be at the midpoint of the pre-deformation bend line. The bend line lies on the Z-axis. The Y-axis is normal to the sheet and points in the initial direction of material movement. Thus the shear line is the circular arc with end points (0,0,-**chord**) and (0,0,**chord**) and midpoint (**width**,0,0). The latter point is displaced to y = **height**.

```
*)
ENTITY implicit_tab
  SUBTYPE OF (implicit_partial_cutout);
  location : axis_placement;
  chord    : size_parameter;
  width    : size_parameter;
  height   : size_parameter;
  angle    : derivable_angle_dimension;
  bend_dim : bend_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Identifies the **axis placement** that locates the feature.

chord: The length or half-length dimension of the bend line of the cutout, which is a chord of the shear line circular arc.

width: The distance or half-distance dimension from the center of the bend line to the edge of the shear line.

height: The maximum distance the feature extends beyond the material. (Full, preferably, or half dimension.)

angle: The angle the tab makes with unaffected material. (Full, preferably, or semi-angle.)

bend_dim: the (radius, preferably, or diameter) dimension of the bend at the base of the tab.

4.8.1.41 IMPLICIT BEND

An **implicit deformation** in which deforming occurs along a curve (bend line) and is characterized by radius and angle, which may be constant or vary.
An **implicit bend** is applied to a sheet- or plate-like underlying shape.

```
*)
ENTITY implicit_bend
  SUPERTYPE OF (implicit_general_bend XOR
                implicit_cutout_flange XOR
                implicit_straight_bend)
  SUBTYPE OF (implicit_deformation);
END_ENTITY;
(*
```

4.8.1.42 IMPLICIT GENERAL BEND

The general case of **implicit bend**. The bend is specified by giving its bendline and the angle and radius at points on the bendline.
The bendline must lie on the pre-bend shape and be on the side of the sheet/plate that will be the convex (inner) side of the bend. After bending, it is the intersection curve of the two inner surfaces around the bend. (Due to bend radius, the bendline is off-part after bending.)

```
*)
ENTITY implicit_general_bend
  SUBTYPE OF (implicit_bend);
  bend_line      : curve;
  bend_points    : SET [1:#] OF bend_point;
  bend_direction : hands;
  moved_side     : OPTIONAL hands;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

bend_line: Identifies the bendline curve of the feature.

bend_points: The bend points that describe the bend angles required at certain points along the bendline.

moved_side: For a bend described using a bendline, an indication of the side of the bendline on which material is regarded (for computational purposes, at least) as to be moved (or having been moved) by the bend. Material on the other side of the bendline is regarded as static. The purpose of this indication is to constrain calculation of post-bend shape (if the bend is not realized in the geometric model) or pre-bend shape (if the bend is realized in the geometric model). Whether material on the left or right side of the bendline, with respect to the direction of that curve, is to be/was moved.

```
*)
RULE bend_point_location FOR (implicit_general_bend);
  LOCAL
    i : INTEGER;
  END_LOCAL;
  REPEAT i := 1 TO SIZEOF(implicit_general_bend.bend_points);
    IF distance(implicit_general_bend.blend_line,
                bend_points[i].point) <> 0.0 THEN
      VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. The bend points must lie on the blend line.

4.8.1.43 BEND POINT

A point on the bendline of an implicit general bend at which bend angle and radius are specified.

```
*)
ENTITY bend_point;
  location     : point;
  bend_dim     : bend_dimension;
  bend_angle   : angle_parameter;
WHERE
  bend_angle < 180;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Identifies the point on the bendline.

bend_dim: The bend size dimension (radius, preferably, or diameter) at the point.

bend_angle: The bend angle at the point. (While semi-angle may be used, whole angle is preferred.)

4.8.1.44 IMPLICIT-CUTOUT FLANGE

An implicit representation of a flange formed by a bend around the periphery of a passage (cutout) feature. The feature has constant setback, bend angle, and bend radius.

```
*)
ENTITY implicit_cutout_flange
  SUBTYPE OF (implicit_bend);
  stiffens     : implicit_passage;
  flange_side  : feature_end_types;
  setback      : size_parameter;
  bend_dim     : bend_dimension;
  bend_angle   : angle_parameter;
UNIQUE
  stiffens;
WHERE
  bend_angle.dimension < 180;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

stiffens: The implicit passage that is stiffened by this implicit cutout flange.

flange_side: The indication of the side to be flanged being initial or terminal end.

setback: The distance from the border of the cutout to the bend line of the flange. (Semi-distance is possible, but full is preferred.)

bend_dim: The dimension (radius, preferably, or diameter) of the bend.

bend_angle: The bend angle dimension. (Semi-angle is possible, but full is preferred.)

PROPOSITIONS:

1. Stiffens must be UNIQUE.
2. The bend angle must be less than 180 degrees.

4.8.1.45 IMPLICIT STRAIGHT BEND

An implicit representation of a simple bend having linear bendline, constant angle, and constant radius.

```
*)
ENTITY implicit_straight_bend
  SUBTYPE OF (implicit_bend);
  bend_line      : line;
  bend_dim       : bend_dimension;
  bend_angle     : angle_parameter;
  bend_direction : hands;
  moved_side     : OPTIONAL hands;
WHERE
```

```
  bend_angle.dimension < 180;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

bend_line: Identifies the bendline line. The bendline line must lie on the pre-bend shape and be on the side of the sheet/plate that will be the convex (inner) side of the bend. After bending, it is the intersection curve of the two inner surfaces around the bend. (Due to bend radius, the bendline is off-part after bending.)

bend_dim: The size dimension (radius, preferably, or diameter) of the bend.

bend_angle: The angle of the bend. (While semi-angle is possible, full measurement is preferred.)

bend_direction: Tells whether the bending direction is by the left of right hand rule with respect to the direction of the bendline.

moved_side: For a bend described using a bendline, an indication of the side of the bendline on which material is regarded (for computational purposes, at least) as to be moved (or having been moved) by the bend. Material on the other side of the bendline is regarded as static. The purpose of this indication is to constrain calculation of post-bend shape (if the bend is not realized in the geometric model) or pre-bend shape (if the bend is realized in the geometric model). Whether material on the left or right side of the bendline, with respect to the direction of that curve, is to be/was moved.

PROPOSITIONS:

1. The bend angle must be less than 180 degrees.

4.8.1.46 IMPLICIT TUBE DEFORMATION

An implicit representation of a deformation of a tube-like shape.

The shape with which an **implicit tube deformation** is associated must be tubular, except that an implicit tube bend may be associated with a solid round bar.

```
*)
ENTITY implicit_tube_deformation
  SUPERTYPE OF (implicit_tube_bend XOR
                implicit_tube_flare XOR
                implicit_tube_neck XOR
                implicit_tube_flattening XOR
                implicit_tube_roll)
  SUBTYPE OF (implicit_deformation);
END_ENTITY;
(*
```

4.8.1.47 IMPLICIT TUBE BEND

An implicit representation of a bend in a tube or circular bar.

```
*)
ENTITY implicit_tube_bend
  SUBTYPE OF (implicit_tube_deformation);
  location    : point;
  direction   : vector;
  bend_size   : bend_dimension;
  bend_angle  : derivable_angle_dimension;
  moved_end   : tube_bend_moved_end_types;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: The bend point of the feature. The bend locating point must lie on the pre-bend centerline of the tube. After bending, it is the intersection point of the centerlines of the two lengths into which the bend divides the tube. (Due to bend radius, the point is off-centerline after bending.)

direction: The vector that gives bend direction.

bend_size: The size (diameter or, preferably, radius) dimension of the bend.

bend_angle: The angle dimension of the bend. (Semi-angle is possible, but full angle is usually preferred.)

moved_end: The purpose of this indication is to constrain calculation of post-bend shape (if the bend is not realized in the geometric model) or pre-bend shape (if the bend is realized in the geometric model). Whether movement of material is on the forward or rearward side, with respect to the centerline of the tube, of the point that locates the bend.

4.8.1.48 IMPLICIT_TUBE_FLARE

An implicit representation of a tapered increase in the diameter of a tube, occurring at an end of the tube.

```
*)
ENTITY implicit_tube_flare
  SUBTYPE OF (implicit_tube_deformation);
  flared_end  : feature_end_types;
  end_od      : size_parameter;
  length      : size_parameter;
  angle       : derivable_angle_dimension;
  bend_dim    : bend_dimension;
  end_id      : derivable_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

flared_end: Whether the flare occurs at the initial or terminal end of the tube. (It is assumed that the tube has a centerline curve whose direction gives meaning to this attribute.)

end_od: The OD dimension (diameter or radius) at the end of the tube.

length: The length dimension of the flare. (it is possible to give semi-length, but full length is preferred.)

angle: The derivable angle of the flare. (It is possible to give semi-angle, but full angle is preferred.)

bend_dim: The bend dimension (radius, preferably, or diameter) at the boundary of the feature.

end_id: The derivable ID dimension (radius or diameter) at the end of the tube.

4.8.1.49 IMPLICIT TUBE NECK

An implicit representation of a tapered change in the diameter of a tube.

```
*)
ENTITY implicit_tube_neck
  SUBTYPE OF (implicit_tube_deformation);
  location        : point;
  neck_direction  : neck_direction_types;
  outside_dim     : size_parameter;
  inside_dim      : derivable_dimension;
  length          : size_parameter;
  angle           : derivable_angle_dimension;
  bend_dim        : bend_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Identifies the **point** that locates the feature. The **point** is located on the tube centerline at the place where diameter begins to increase; i.e., at the intersection of the smaller diameter section of tube and the tapered section (ignoring the bend radius).

neck_direction: Tells whether the increase in diameter occurs in the positive or negative direction from this **point**. (This assumes that there is a directed centerline curve associated with the tube.)

outside_dim: The OD dimension (diameter or radius) of the neck.

inside_dim: The derivable ID dimension (diameter or radius) of the neck.

length: The length of the tube over which the change in diameter occurs. (Half-measure can be given, but full is preferred.)

angle: The derivable angle dimension of the neck. (Half-measure can be given, but full is preferred.)

bend_dim: The bend dimension (radius, preferably, or diameter) at the boundary between the feature and the larger (by diameter) of the two constant-diameter sections of tube it connects.

4.8.1.50 IMPLICIT TUBE FLATTENING

An implicit representation of a deformation of the end of a tube to an oblong shape.
Original tube OD, ID, and wall thickness are presumed to be known from other model information. Other dimensions can be calculated from these, **end width**, and **length**, assuming

1. constant wall thickness and

2. constant circumference, and

3. linear variation of all dimensions from local z = 0 to z = length, and

4. constant tube length.

```
*)
ENTITY implicit_tube_flattening
  SUBTYPE OF (implicit_tube_deformation);
  location     : axis_placement;
  outer_major  : size_parameter;
  outer_minor  : size_parameter;
  length1      : size_parameter;
  length2      : derivable_dimension;
  bend_dim     : bend_dimension;
  inner_major  : derivable_dimension;
  inner_minor  : derivable_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Identifies the axis placement that locates the feature. The origin must lie on the centerline of the tube being flattened at the point at which flattening begins to occur. The z-axis points in the direction of changing shape. That is, the z-axis points toward the direction where the oblong shape is achieved. The x-axis is parallel to the minor axis of the oblong and the y-axis to its major axis.

outer_major: The larger dimension (half or full measure) of the oblong, measured on the outside.

outer_minor: The smaller dimension (half or full measure) of the oblong, measured on the outside.

length1: The length along which the transition from circular tube to oblong shape occurs.

length2: The derivable length along which the final oblong shape is maintained.

bend_dim: The dimension (radius, preferably, or diameter) of the bend between the transitional section of the feature and the fully flattened area.

inner_major: The derivable larger dimension (half or full measure) of the inside of the oblong.

inner_minor: The derivable smaller dimension (half or full measure) of the inside of the oblong.

PROPOSITIONS:

1. OUTER MINOR < original tube OD.

2. OUTER MAJOR > original tube OD.

3. LENGTH1 < distance, in the +z direction, from the origin to the end of the tube.

4.8.1.51 IMPLICIT-TUBE ROLL

An implicit representation of a tube deformation feature where a tube end is expanded in diameter and rolled (curled) back on itself. (This feature is usually made by forcing the tube end into a die with a conical center.)

```
*)
ENTITY implicit_tube_roll
  SUBTYPE OF (implicit_tube_deformation);
  rolled_end     : feature_end_types;
  before_length  : size_parameter;
  tube_size      : size_parameter;
  curl_length    : size_parameter;
  gap            : size_parameter;
  curl_size      : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

rolled_end: Whether the flare occurs at the initial or terminal end of the tube. (It is assumed that the tube has a centerline curve whose direction gives meaning to this attribute.)

before_length: The length of tubing to be deformed. (Half dimension is possible, but full is preferred.)

tube_size: The maximum size (diameter, preferably, or radius) at/near the rolled end.

curl_length: The doubled-over length after rolling. (Half dimension is possible, but full is preferred.)

gap: The space between the curled end and the uncurled section of the tube. (Half dimension is possible, but full is preferred.)

curl_size: The curvature dimension (radius, preferably, or diameter) of the curl.

PROPOSITIONS:

1. The rolled end of the tube may not pinch or intersect the tube.
2. Original tube OD, ID, and wall thickness are presumed to be known from other model information.

4.8.1.52 IMPLICIT AREA FEATURE

An implicit representation of one of a class of form features viewed as being installed upon areas of preexisting shape. Examples are gear teeth, knurls, and threads.

Most or all of the feature types in this class are invariably patterns and, in practice, the patterns are described as a whole. That practice is followed here — the entities give a unitary description of the patterns rather than using implicit form feature pattern.

```
*)
ENTITY implicit_area_feature
  SUPERTYPE OF (implicit_knurl XOR
```

```
                implicit_marking XOR
                implicit_coupling XOR
                implicit_thread)
  SUBTYPE OF (implicit_form_feature);
  installation_area : OPTIONAL dimensionality_2_shape_element;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

installation_area: The area on which the feature is installed

PROPOSITIONS:

1. The addition of the feature to a part does not change the genus of the part.

4.8.1.53 IMPLICIT KNURL

An implicit representation of a scoring pattern on a surface.

```
*)
ENTITY implicit_knurl
  SUPERTYPE OF (implicit_diagonal_knurl XOR
                implicit_diamond_knurl XOR
                implicit_straight_knurl)
  SUBTYPE OF (implicit_area_feature);
  number_of_teeth    : INTEGER;
  knurl_major_dim    : size_parameter;
  knurl_nominal_dim  : size_parameter;
  tooth_depth        : size_parameter;
  fillet_at_root     : size_parameter;
WHERE
  cylindrical(installation_area);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

number_of_teeth: The number of teeth on the knurling tool to be used.

knurl_major_dim: The dimension (diameter, preferably, or radius) of the major or outside diameter of the knurling tool.

knurl_nominal_dim: The dimension (diameter, preferably, or radius) of the nominal diameter of the knurling tool.

tooth_depth: The tooth depth of the knurling tool

fillet_at_root: The dimension (diameter or, preferably, radius) of the root fillet between teeth.

4.8.1.54 IMPLICIT-STRAIGHT KNURL

An implicit representation of a knurl whose scoring is parallel to the axis of the scored surface.

```
*)
ENTITY implicit_straight_knurl
  SUBTYPE OF (implicit_knurl);
END_ENTITY;
(*
```

4.8.1.55 IMPLICIT DIAGONAL KNURL

An implicit representation of a knurl whose scoring is spiral about the axis of the scored surface.

```
*)
ENTITY implicit_diagonal_knurl
  SUBTYPE OF (implicit_knurl);
  helix_angle : angle_parameter;
  helix_hand  : hands;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

helix_angle: The angle the knurl helix makes with the work axis.

helix_hand: Whether the helix is Left or Right Hand. The hand of the angle the knurl helix makes with the work axis

4.8.1.56 IMPLICIT DIAMOND KNURL

An implicit representation of a knurl whose scoring is doubly spiral (a left and a right hand spiral) about the axis of the scored surface, with equal spacing of the two.

```
*)
ENTITY implicit_diamond_knurl
  SUBTYPE OF (implicit_knurl);
  helix_angle : angle_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

helix_angle: The angle, in degrees, the knurl helices make with the work axis.

4.8.1.57 IMPLICIT THREAD

An implicit representation of a spiral groove installed on a cylindrical or conical surface.

```
*)
ENTITY implicit_thread
  SUBTYPE OF (implicit_area_feature);
  major_dim         : size_parameter;
  threads_per_unit  : REAL;
  thread_form       : thread_forms;
  thread_hand       : hands;
  pitch_dim         : OPTIONAL size_parameter;
  minor_dim         : OPTIONAL size_parameter;
  thread_fit_class  : OPTIONAL thread_fit_classes;
  thread_spec       : OPTIONAL full_threading_specification;
  thread_apex       : OPTIONAL pitch_apex;
  thread_timing     : OPTIONAL thread_timer;
WHERE
  cylindrical(installation_area) OR
  conical(installation_area);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

major_dim: The dimension (diameter, preferably, or radius) of the major circle.

threads_per_unit: The number of threads per the axial unit of length measure.

thread_form: One of the standard thread forms.

thread_hand: Whether the thread is right or left handed.

pitch_dim: The dimension (diameter, preferably, or radius) of the pitch circle.

minor_dim: The dimension (diameter, preferably, or radius) of the minor circle.

thread_fit_class: The enumeration value for the class of fit specification for the thread.

thread_spec: Information on the full threading requirements.

thread_apex: A description of the point of the thread.

thread_timing: The definition of radial timing or alignment.

4.8.1.58 FULL THREADING SPECIFICATION

A specification of the area of full threading of an **implicit thread**. This entity is used when the **dimensionality 2 shape element** of (the parent **implicit area feature** of) an **implicit thread** is not fully threaded along its entire area. The full threading area is specified via a **point** on the axis of the threaded **dimensionality 2 shape element**. The **dimensionality 2 shape element** must be fully threaded on the side of that point indicated by the attribute **full threading direction**. If **full threading direction** is **upto**, full threading occurs in the negative direction along the axis from the point; if **beyond**, positive.

```
*)
ENTITY full_threading_specification;
  reference_point           : point;
  full_threading_direction  : threading_direction_types;
  full_thread_length        : derivable_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

reference_point: The point that is used to establish the point of reference for the application of the other attributes.

full_threading_direction: Whether full threading occurs up to or beyond the area indicated by the point.

full_thread_length: The derivable length/depth dimension of full threading. While semi-depth can be given, full dimension seems best.)

4.8.1.59 THREAD TIMER

A specification of the rotational placement of an **implicit thread** on the threaded (cylindrical or conical) surface by identifying a pressure point (a point on the pitch spiral).

```
*)
ENTITY thread_timer;
  pressure_point : point;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

pressure_point: The point used for radial alignment of the thread.

4.8.1.60 PITCH APEX

The definition of a pitch cone for an **implicit area feature**. The specification assumes that a directed axis is known from the **dimensionality 2 shape element** on which the feature is installed.

```
*)
ENTITY pitch_apex;
  apex_point               : point;
  pitch_distance           : REAL;
  pitch_in_axis_direction  : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

apex_point: The point that is the apex of the cone.

pitch_distance: The distance along the axis from the apex to the axial point at which diametral dimensions (e.g., pitch diameter, major diameter, minor diameter) are given.

pitch_in_axis_direction: Whether the direction from the apex to the axial point is the same as or the opposite of the axis direction.

4.8.1.61 IMPLICIT MARKING

An implicit representation of the application of a character (or standard symbol) string to an area of a shape. This is a very weak specification: there is no ability to specify size, font, depressed/raised lettering, layout of the characters, etc. Only the characters and, optionally, area can be specified.

```
*)
ENTITY implicit_marking
  SUBTYPE OF (implicit_area_feature);
  marked_string : STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

marked_string: An alphanumeric string which is to be present.

4.8.1.62 IMPLICIT COUPLING

An implicit representation of a toothed feature installed at the end of a cylindrical shape whose purpose is to fix axially adjacent components with respect to each other. The feature's teeth have sides with an arc shape. One coupling in a pair will have convex (barrel shaped) teeth; the other, concave.

```
*)
ENTITY implicit_coupling
  SUBTYPE OF (implicit_area_feature);
  coupling_shape             : coupling_shape_types;
  number_of_teeth            : INTEGER;
  pressure_angle             : angle_parameter;
  whole_depth                : size_parameter;
  chamfer_depth              : size_parameter;
  root_fillet                : size_parameter;
  dim_to_point_of_tangency   : size_parameter;
  coupling_timing            : OPTIONAL coupling_timer;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

coupling_shape: Whether the coupling is convex or concave.

number_of_teeth: The number of teeth in the coupling.

pressure_angle: The angle between the tapered mating walls of teeth and the radial direction of the cylindrical shape.

whole_depth: The tooth depth dimension. (Half dimension is possible, but discouraged.)

chamfer_depth: The distance dimension from the top of a tooth to the tapered contact surface of the tooth. (Half dimension is possible, but discouraged.)

root_fillet: The dimension (radius, preferably, or diameter) of the fillet between tooth and slot bottom.

dim_to_point_of_tangency: The dimension (radius, preferably, or diameter) of the grinding wheel at the pitch line, which establishes the curvature of the teeth sides.

coupling_timing: A fixing of the rotational alignment of the coupling about its axis.

PROPOSITIONS:

1. The INSTALLATION AREA attribute on which an IMPLICIT COUPLING is installed must be an open planar face of axisymmetric geometry, adjacent to an internal and an external diameter.

4.8.1.63 COUPLING TIMER

The use of a **point** to control the rotational placement of an **implicit coupling**. The **point** lies on any radial (with respect to the coupling's axis of rotation) line passing through the middle of any tooth of the feature.

```
*)
ENTITY coupling_timer;
  timing_point : point;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

timing_point: The point that establishes the rotational alignment of the coupling on it's axis.

4.8.1.64 IMPLICIT FORM FEATURE PATTERN

A representation of a **form feature** as an arrangement of identical (except for location/orientation) form features according to some mathematical logic. The pattern is represented by the identification of one of its member features (the "base" feature) and the logic for arranging "copies" of the base feature.

```
*)
ENTITY implicit_form_feature_pattern
  SUPERTYPE OF (implicit_array_form_feature_pattern XOR
                implicit_circular_form_feature_pattern);
  base_form_feature : implicit_form_feature;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

base_form_feature: The implicit form feature to be copied.

4.8.1.65 IMPLICIT-CIRCULAR FORM FEATURE PATTERN

An **implicit form feature pattern** whose component features are arranged in a circular arc pattern; i.e., are equally spaced about a centerline.

The data "constructs" the pattern as follows. The construction provides a way of referring to individual members of the pattern.

1. Member #1 is the **base feature** (see **implicit form feature pattern.**)
2. Member #I is a "copy" of the **base feature**, rotated (I-1)***angular spacing** about the pattern centerline in the counterclockwise direction, as viewed when looking in the direction of the centerline.

```
*)
ENTITY implicit_circular_form_feature_pattern
  SUBTYPE OF (implicit_form_feature_pattern);
  centerline        : line;
  number_of_members : INTEGER;
  angular_spacing   : angle_parameter;
  omissions         : SET [0:#] OF circular_pattern_omission;
  offsets           : SET [0:#] OF circular_pattern_offset_member;
WHERE
  (number_of_members * angular_spacing.dimension) < 360;
  number_of_members - SIZEOF(omissions) >= 2;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

centerline: The definition of the axis of symmetry for the pattern.

number_of_members: The number of locations in the pattern.

angular_spacing: The angle, in degrees, between each pattern location.

omissions: Set of locations with no feature.

offsets: Set of feature locations that are offset from their normal position.

```
*)
RULE verify_circular_pattern_omissions_and_offsets FOR
  (implicit_circular_form_feature_pattern,
   circular_pattern_omission,
   circular_pattern_offset_member);
LOCAL
  i,j     : INTEGER;
  omit    : circular_pattern_omission;
  offset  : circular_pattern_offset_member;
  pattern : implicit_circular_form_feature_pattern;
END_LOCAL;
  pattern := implicit_circular_form_feature_pattern;
  REPEAT i := 1 TO SIZEOF(pattern.omissions);
```

```
    omit := pattern.omissions[i];
    IF (omit.omitted_member_number <= 1) THEN
      VIOLATION;
    END_IF;
    IF (omit.omitted_member_number >
        pattern.number_of_members) THEN
      VIOLATION;
    END_IF;
    REPEAT j := 1 TO SIZEOF(pattern.offsets);
      offset := pattern.offsets[j];
      IF (offset.offset_member_number >
          pattern.number_of_members) THEN
        VIOLATION;
      END_IF;
      IF (offset.offset_member_number =
          omit.omitted_member_number) THEN
        VIOLATION;
      END_IF;
    END_REPEAT;
 END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. There must be at least two used locations in the pattern.
2. A location cannot be both offset and omitted.
3. The first pattern location must not be omitted.
4. Both offset and omitted locations must be within the pattern extent.

4.8.1.66 CIRCULAR PATTERN OMISSION

An indication that one of the members of an **implicit circular form feature pattern** is absent.

```
*)
ENTITY circular_pattern_omission;
  omitted_member_number : INTEGER;
WHERE
  omitted_member_number > 1;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

omitted_member_number: The sequence number of the pattern member to be omitted.

PROPOSITIONS:

1. The **omitted member number** must be greater than 1.

4.8.1.67 CIRCULAR PATTERN OFFSET MEMBER

An indication that a member of a circular pattern feature is at a location other than that indicated by the pattern rule.

```
*)
ENTITY circular_pattern_offset_member;
  offset_member_number : INTEGER;
  offset_angle         : REAL;
WHERE
  offset_member_number > 1;
  offset_angle <> 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

offset_member_number: The number of the pattern location that is offset.

offset_angle: The signed offset angle.

PROPOSITIONS:

1. The offset member number must be greater than zero.
2. Only angular offsetting is permitted.
3. Angular offset cannot equal zero.

4.8.1.68 IMPLICIT ARRAY FORM FEATURE PATTERN

An implicit pattern feature whose component features are arranged in a pattern of rows and columns.

```
*)
ENTITY implicit_array_form_feature_pattern
  SUPERTYPE OF (parallel_equal_spacing_array_pattern XOR
                parametric_equal_spacing_array_pattern)
  SUBTYPE OF (implicit_form_feature_pattern);
  number_of_rows    : INTEGER;
  number_of_columns : INTEGER;
  omissions         : SET [0:?] OF array_pattern_omission;
WHERE
  number_of_rows > 0;
  number_of_columns > 0;
  number_of_rows + number_of_columns > 2;
  SIZEOF(omissions) < (number_of_rows)*(number_of_columns) - 1;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

number_of_rows: The number of rows in the pattern.

number_of_columns: The number of columns in the pattern.

omissions: Set of locations to be omitted from the pattern.

PROPOSITIONS:

1. There must be at least one row in the pattern
2. There must be at least one column in the pattern
3. There must be at least two locations in the pattern
4. There must be at least two used locations in the pattern
5. An omitted location must be within the pattern extent

4.8.1.69 ARRAY PATTERN OMISSION

An indication that one feature of a rows-and-columns array of features is absent.

```
*)
ENTITY array_pattern_omission;
  row    : INTEGER;
  column : INTEGER;
WHERE
  row >= 1;
  column >= 1;
  (row <> 1) or (column <> 1);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

row: the array pattern location that is missing a feature.

column: the array pattern location that is missing a feature.

PROPOSITIONS:

1. The row and column numbers must both be greater than zero.
2. The row and column numbers may not be greater than the total parent pattern number of members.

4.8.1.70 PARAMETRIC EQUAL SPACING ARRAY PATTERN

An array of features arranged on a parametrically represented surface in such a way that the spacing between adjacent features on the same row is given by a U-delta and the spacing between adjacent features on the same column is given by a V-delta. The orientation of each member feature with respect to the surface is the same as that of the base feature.
The features in the pattern are located as follows:

1. Denote the layout surface as $f(U, V)$.

2. Denote the offsets in the U and V directions as Δ_u and Δ_v respectively.

3. The base feature of the pattern must have an LCS; i.e., be a type of **implicit form feature** with an associated **axis placement**.

4. The **base feature** must be known to correspond to a point (U_1, V_1) in parameter space.

5. The parameter space location of the I, J member of the pattern is (U_{ij}, V_{ij}), where $U_{ij} = U_1(I-1)\Delta_u$ and $V_{ij} = V_1(J-1)\Delta_v$.

6. Let A_{11} be the **axis placement** defined by $f(U_1, V_1)$, the partial derivative f_u of f with respect to U, and the

   partial derivative f_v of f with respect to V.

7. Let T be the transformation that takes A_{11} into the LCS of the base feature; i.e., $A_{11}T = LCS_{11}$.

8. For the I, J member of the pattern, let A_{ij} be the **axis placement** defined by $f(U_{ij}, V_{ij})$, the partial derivative, $f_{u_{ij}}$, of f with respect to U at that point, and the partial derivative, $f_{v_{ij}}$, of f with respect to V at that point.

9. Then the location and orientation of the I, J feature of the pattern are established by positioning its LCS at $A_{ij}T$; i.e., $LCS_{ij} = A_{ij}T$.

```
*)
ENTITY parametric_equal_spacing_array_pattern
  SUBTYPE OF (implicit_array_form_feature_pattern);
  layout_reference : surface;
  udelta           : REAL;
  vdelta           : REAL;
  offsets          : SET [0:#] OF
                         parametric_array_pattern_offset_member;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

layout_reference: The surface on which the pattern is laid out.

udelta: The row spacing in the U-direction of the surface.

vdelta: The column spacing in the V-direction of the surface.

offsets: The description of the possible offset members.

```
*)
RULE verify_parametric_array_pattern_omissions_and_offsets FOR
     (parametric_equal_spacing_array_pattern,
      array_pattern_omission,
      parametric_array_pattern_offset_member);
LOCAL
  i,j    : INTEGER;
  omit   : array_pattern_omission;
  offset : parametric_array_pattern_offset_member;
```

```
  pattern : parametric_equal_spacing_array_pattern;
END_LOCAL;
  pattern := parametric_equal_spacing_array_pattern;
  REPEAT i := 1 TO SIZEOF(pattern.omissions);
    omit := pattern.omissions[i];
    IF (omit.row > pattern.member_of_rows) THEN
      VIOLATION;
    END_IF;
    IF (omit.column > pattern.member_of_columns) THEN
      VIOLATION;
    END_IF;
    REPEAT j := 1 TO SIZEOF(pattern.offsets);
      offset := pattern.offsets[j];
      IF (offset.row > pattern.member_of_rows) THEN
        VIOLATION;
      END_IF;
      IF (offset.columns > pattern.member_of_columns) THEN
        VIOLATION;
      END_IF;
      IF ((offset.row=omit.row) AND (offset.column=omit.column))
        THEN VIOLATION;
      END_IF;
    END_REPEAT;
 END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. An offset member must be within the extent of the pattern.

2. A location cannot be both omitted and offset.

4.8.1.71 PARAMETRIC ARRAY PATTERN OFFSET MEMBER

An indication that a member feature of a parametric equal spacing array pattern is located elsewhere than indicated by the pattern rule.

```
*)
ENTITY parametric_array_pattern_offset_member;
  row       : INTEGER;
  column    : INTEGER;
  u_offset  : REAL;
  v_offset  : REAL;
WHERE
  ((row >=1) OR (column >= 1));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

row: The row location of a pattern offset.

column: The column location of a pattern offset.

u_offset: The offset from the calculated U position.

v_offset: The offset from the calculated V position.

PROPOSITIONS:

1. The row and column numbers must both be greater than zero.

4.8.1.72 PARALLEL EQUAL SPACING ARRAY PATTERN

An array of features which have the same orientation and are equally spaced in the object space.

```
*)
ENTITY parallel_equal_spacing_array_pattern
  SUBTYPE OF (implicit_array_form_feature_pattern);
  row_translation     : vector_with_magnitude;
  column_translation  : vector_with_magnitude;
  row_spacing         : derivable_dimension;
  column_spacing      : derivable_dimension;
  offsets             : SET [0:#] OF
                          parallel_array_pattern_offset_member;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

row_translation : The translation that takes the I^{th} member of a row into the $(I+1)^{th}$ member of that row.

column_translation: The translation that takes the J^{th} member of a column into the $(J+1)^{th}$ member of that column.

row_spacing: The normal distance between rows.

column_spacing: The normal distance between columns.

offsets: Pattern members that are offset from their normal location.

```
*)
RULE verify_parallel_array_pattern_omissions_and_offsets FOR
     (parallel_equal_spacing_array_pattern,
      array_pattern_omission,
      parallel_array_pattern_offset_member);
LOCAL
  i,j     : INTEGER;
  omit    : array_pattern_omission;
  offset  : parallel_array_pattern_offset_member;
  pattern : parallel_equal_spacing_array_pattern;
```

```
END_LOCAL;
  pattern := parallel_equal_spacing_array_pattern;
  REPEAT i := 1 TO SIZEOF(pattern.omissions);
    omit := pattern.omissions[i];
    IF (omit.row <= 1) THEN
      VIOLATION;
    END_IF;
    IF (omit.column <= 1) THEN
      VIOLATION;
    END_IF;
    IF (omit.row > pattern.number_of_rows) THEN
      VIOLATION;
    END_IF;
    IF (omit.column > pattern.number_of_columns) THEN
      VIOLATION;
    END_IF;
    REPEAT j := 1 TO SIZEOF(pattern.offsets);
      offset := pattern.offsets[j];
      IF (offset.row > pattern.number_of_rows) THEN
        VIOLATION;
      END_IF;
      IF (offset.columns > pattern.number_of_columns) THEN
        VIOLATION;
      END_IF;
      IF ((offset.row = omit.row) AND (offset.column = omit.column))
        THEN VIOLATION;
      END_IF;
    END_REPEAT;
 END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. Offsets must be within the extent of the pattern.

2. A location must not both be offset and omitted.

4.8.1.73 PARALLEL ARRAY PATTERN OFFSET MEMBER

An indication that a member feature of a parallel equal spacing array pattern is located elsewhere than indicated by the pattern rule.

```
*)
ENTITY parallel_array_pattern_offset_member;
  row    : INTEGER;
  column : INTEGER;
  offset : vector_with_magnitude;
WHERE
  ((row >= 1) OR (column >= 1));
```

```
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

row : The row location of a pattern offset.

column: The column location of a pattern offset.

offset: The vectored offset from the calculated pattern position.

PROPOSITIONS:

1. The row and column numbers must both be greater than zero.

4.8.1.74 REPLICATE FORM FEATURE

A representation of a **form feature** as a "copy" of another **form feature**, but in a different location and perhaps reflected (mirrored). the copied feature necessarily has an implicit representation. The replicate can be "realized" by applying a rigid transformation to its local coordinate system, if it has one, and to any definitional geometric data (points, curves, surfaces, etc.). Features dependent on non-transformable data (e.g., area features, transitions) cannot be replicated.

```
*)
ENTITY replicate_form_feature;
  copied_feature : implicit_form_feature;
  location       : axis_placement;
  mirror         : OPTIONAL coordinate_enumeration;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

copied_feature: The feature that is being replicated.

location: The location of the copy.

mirror: An indication that a **replicate form feature** is a reflection (mirror image) as well as a relocation of the feature of which it is a copy. The reflection is performed with respect to a specified coordinate plane of the local coordinate system. The coordinate (X, Y, or Z) that is negated in the reflection. For example, if this value is **z coordinate**, the reflection is with respect to the local XY-plane, taking Z into -Z.

PROPOSITIONS:

1. The copied feature must be an IMPLICIT FORM FEATURE that has an associated location.

4.8.1.75 FEATURE VOLUME

A volume added to or subtracted from pre-existing shape. Used to specify implicit definitions of form features modeled as increments/decrements of material.

```
*)
ENTITY feature_volume
  SUPERTYPE OF (feature_sweep XOR
                feature_ruling);
END_ENTITY;
(*
```

4.8.1.76 FEATURE RULING

An implicit representation of a form feature as a ruling of two curves. A **feature ruling** is used to specify an **implicit passage**, **implicit protrusion**, or **implicit depression**. A **feature ruling** may be viewed as a generalization of a **feature sweep**; it is used to define added or subtracted volumes over a finite range. The directionality implicit is sweeps is extended to rulings by regarding the first of the two defining curves as the initial end of the ruling and the second as the terminal end. The curves defining the ruled wall of a **feature ruling** are required to be **curve on surface**. The underlying **surfaces** of the **curve on surfaces** specify the ends of the added or subtracted volume. The curves defining the ruling may be open or closed. If open, it is the creator's responsibility to insure that the ruled surface extends sufficiently far that there is no ambiguity. That is, the ruling lines corresponding to the first and last points on the defining curves must lie in air or on the air/material boundary for a subtracted volume; in material or on the boundary for an added volume. A **feature ruling** has a "local" coordinate system. This may be specified by an **axis placement**. If there is no **axis placement**, the global coordinate system is by default the local system. The defining curves of the ruling are assumed to be specified in the local coordinate system.

```
*)
ENTITY feature_ruling
  SUBTYPE OF (feature_volume);
  defining_curve1 : curve_on_surface;
  defining_curve2 : curve_on_surface;
  blend           : SET [0:2] OF feature_ruling_wall_end_blend;
  location        : OPTIONAL axis_placement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

defining_curve1: Identifies the first of the two curves which defines the ruling.

defining_curve2: Identifies the second of the two curves which defines the ruling.

blend: Defines a blend between the walls and the ends formed by the feature ruling.

location: The identification of the possible **axis placement** for establishing the local coordinate system.

4.8.1.77 FEATURE RULING WALL END BLEND

A blend (round or chamfer) between the ruled surface wall of a **feature ruling** and one of its end surfaces. (The ruling curves are **curve on surface**. The end surfaces are the surfaces on which the ruling curves lie.)

```
*)
ENTITY feature_ruling_wall_end_blend;
  blend_end : feature_end_types;
  blend     : implicit_edge_blend;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

blend_end: Whether the blend is at the initial (**defining curve1**) or terminal (**defining curve2**) end of the ruling.

blend: Identifies the **implicit edge blend** that applies at the intersection of the wall and the end.

4.8.1.78 FEATURE SWEEP

A procedural definition of a 2 1/2 dimensional shape consisting of a planar profile, a path along which the profile is to be swept, and shape definitions for either or both ends. The profile is swept normal to the path. The orientation of the path with respect to the profile is achieved by:

1. Providing a pre-defined orientation for each path type relative to the feature's local coordinate system (LCS). For example, linear sweep paths begin at the LCS origin and extend along the positive Z-axis.
2. Defining profiles in their own 2-space called AB-space. Each profile type has a pre-defined location and orientation in its AB-space. For example, a rectangular profile has its center at the AB-origin with a specified pair of sides parallel to the A-axis.
3. Positioning the AB-axes of the profile in the feature's LCS according to conventions for each path type. For a linear path, the profile's A- and B-axes are mapped to the X- and Y-axes of the LCS, respectively.

The shape of **feature sweeps** may be refined via three types of blends: blends within the profile; blends within sweep end shapes, such as a radiused tip on a conical bottomed hole; and blends between the sides and ends of the swept shape. The sweep defines a subtracted volume for **swept** depressions and implicit passages, and an added volume for **implicit protrusions.**

```
*)
ENTITY feature_sweep
  SUPERTYPE OF (along_feature_sweep XOR
                axisymmetric_feature_sweep XOR
                in_out_feature_sweep)
  SUBTYPE OF (feature_volume);
  location : axis_placement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: The **axis placement** which locates the swept feature. It does so, in effect, by transforming the swept profile from the 2-space in which it is defined into its initial sweeping position in global space.

4.8.1.79 ALONG FEATURE SWEEP

A **feature sweep** whose path is roughly along the boundary (air/material interface) of preexisting shape. Some common features defined in this manner are grooves and channels.

```
*)
ENTITY along_feature_sweep
  SUBTYPE OF (feature_sweep);
  sweep_path    : feature_sweep_path;
  sweep_profile : open_feature_sweep_profile;
  sweep_ends    : SET [0:2] OF along_feature_sweep_end;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_path: The path that defines the sweep.

sweep_profile: The profile that is to be swept along the path.

sweep_ends: A description of the bounds of the sweep.

4.8.1.80 ALONG FEATURE SWEEP END

The end shape of an **implicit form feature** defined by an **along feature sweep**. It may only be used when the end of the sweep is not at the air/material boundary. As an example, this entity can be used to model the rounded end of a milled slot.

1. When the sweep does not end at the air/material boundary and no **along feature sweep end** is specified, the feature is assumed to have a flat end.
2. Flat ends must be modeled explicitly when the edge between the end and the rest of the feature is blended.

```
*)
ENTITY along_feature_sweep_end
  SUPERTYPE OF (along_feature_sweep_flat_end XOR
                along_feature_sweep_radiused_end);
  sweep_end : feature_end_types;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_end: Whether the end is the initial or terminal one.

4.8.1.81 ALONG FEATURE SWEEP FLAT END

An indication that an **along feature sweep end** is planar.

```
*)
ENTITY along_feature_sweep_flat_end
  SUBTYPE OF (along_feature_sweep_end);
  end_blend : OPTIONAL implicit_edge_blend;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

end_blend: A definition of the desired end blend.

4.8.1.82 ALONG FEATURE SWEEP RADIUSED END

An indication that an **along feature sweep end** is defined by a radius. The shape of the end depends on the type of **open feature sweep profile** used. For all profiles except the **ell feature sweep profile**, the **line plus radius feature sweep profile** and the **half obround feature sweep profile**, the shape can be thought of as the result of sweeping the portion of the profile in its positive A half-plane 180 degrees about the local B-axis. For example,

- the end shape for a **tee feature sweep profile** consists of two concentric half-cylinders.
- For an **ell feature sweep profile**, the end shape consists of

  a semi-infinite cylinder with a radius equal to half the stem width, and another cylinder with a radius equal to half the endbar width.
- For a **line plus radius feature sweep profile** and a **half obround feature sweep profile**, the end shape is obtained by sweeping the defining arcs 90 degrees about a line parallel to their local B-axes at the arc vertices, and sweeping the rest of the profile (i.e. the part above the defining line) a distance equal to the arc radius.

```
*)
ENTITY along_feature_sweep_radiused_end
  SUBTYPE OF (along_feature_sweep_end);
END_ENTITY;
(*
```

4.8.1.83 AXISYMMETRIC FEATURE SWEEP

A **feature sweep** that is realized by sweeping a planar curve 360 degrees about a coplanar axis. The swept curve may be explicitly or implicitly defined. Explicit definition is available via **other axisymmetric feature sweep**, which uses planar curve strings to define arbitrary profiles. Implicit curves are defined by **constant diameter axisymmetric feature sweep** and **tapered axisymmetric feature sweep**.

1. The sweep axis coincides with the Z-axis of the feature's local coordinate system.
2. The curve may be of any length. However, it is intended that only the portion of the curve defined between $Z = 0$ and $Z = LENGTH$ be operative. (If the curve is not defined throughout the range $0 <= Z <= LENGTH$, then it is assumed to extend to cover the range.)
3. All profiles for axisymmetric sweeps are defined in a local AB-space. The local A- and B-axes of the profile are equated with the X- and Z-axes, respectively, of the feature's LCS.

4. The curve must not intersect the Z axis in the range $0 < Z < LENGTH$.

5. The ends of the swept curve are assumed to be planar, unless an **axisymmetric feature sweep end** is specified. If an **axisymmetric feature sweep end** is given, it extends in the -Z direction from $Z = 0$.

```
*)
ENTITY axisymmetric_feature_sweep
  SUPERTYPE OF (constant_diameter_axisymmetric_feature_sweep XOR
                tapered_axisymmetric_feature_sweep XOR
                other_axisymmetric_feature_sweep)
  SUBTYPE OF (feature_sweep);
  sweep_length : size_parameter;
  sweep_end    : OPTIONAL axisymmetric_feature_sweep_end;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_length: The length dimension of the swept profile. (Semi-length is possible, but full is preferred.)

sweep_end: A definition of the desired end.

4.8.1.84 CONSTANT DIAMETER AXISYMMETRIC FEATURE SWEEP

An **axisymmetric feature sweep** whose profile consists of a line parallel to the local profile B-axis. After equating the local B-axis with the feature's Z-axis, the line is rotated about the Z-axis to form a cylinder.

```
*)
ENTITY constant_diameter_axisymmetric_feature_sweep
  SUBTYPE OF (axisymmetric_feature_sweep);
  sweep_size : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_size: The size dimension (diameter or radius) of the sweep.

4.8.1.85 TAPERED AXISYMMETRIC FEATURE SWEEP

An **axisymmetric feature sweep** whose profile consists of a half-line (the half in the +B half-plane) which is not parallel to the local B-axis.

```
*)
ENTITY tapered_axisymmetric_feature_sweep
  SUBTYPE OF (axisymmetric_feature_sweep);
  size_at_origin : size_parameter;
```

```
  angle_dim      : angle_parameter;
  taper_type     : taper_types;
WHERE
  size_at_origin.dimension >= 0.0;
  angle_dim.dimension < 90.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

size_at_origin: The distance (or twice the distance) from the AB-origin of the implicitly defined profile to the profile, measured along the positive A-axis. This value translates to the radius or diameter of the feature's cross-section in the local XY-plane.

angle_dim: The full- or semi-angle dimension of the cone formed by the sweep.

taper_type: An indication of whether the cone's diameter increases or decreases as local B increases.

4.8.1.86 OTHER AXISYMMETRIC FEATURE SWEEP

An axisymmetric feature sweep whose profile is defined by a general profile.

```
*)
ENTITY other_axisymmetric_feature_sweep
  SUBTYPE OF (axisymmetric_feature_sweep);
  sweep_profile : general_profile;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_profile: The profile to be rotated.

PROPOSITIONS:

1. The profile must not intersect the Z axis
2. The profile must be co-planar with the Z axis

4.8.1.87 AXISYMMETRIC FEATURE SWEEP END

The end shape of an implicit form feature defined by an axisymmetric feature sweep. As an example, this entity can be used to model the common end shapes for a hole: flat, conical, and spherical.

1. The end meets the swept body of the feature at $Z = 0$. Non-planar ends extend in the -Z direction.
2. When no axisymmetric feature sweep end is specified for an axisymmetric feature sweep, the feature is assumed to have a flat end. A flat end must be modeled explicitly in order to have an associated blend.

```
*)
ENTITY axisymmetric_feature_sweep_end
  SUPERTYPE OF (axisymmetric_feature_sweep_flat_end XOR
                axisymmetric_feature_sweep_spherical_end XOR
                axisymmetric_feature_sweep_conical_end);
  blend : OPTIONAL implicit_edge_blend;
END_ENTITY;
(*
```

4.8.1.88 AXISYMMETRIC FEATURE SWEEP FLAT END

An indication that an axisymmetric feature sweep end is planar.

The plane of the axisymmetric feature sweep flat end is local $Z = 0$.

```
*)
ENTITY axisymmetric_feature_sweep_flat_end
  SUBTYPE OF (axisymmetric_feature_sweep_end);
END_ENTITY;
(*
```

4.8.1.89 AXISYMMETRIC FEATURE SWEEP SPHERICAL END

An indication that an axisymmetric feature sweep end is spherical.

The radius of an axisymmetric feature sweep spherical end is equal to the radius of the parent axisymmetric feature sweep at local $Z = 0$.

```
*)
ENTITY axisymmetric_feature_sweep_spherical_end
  SUBTYPE OF (axisymmetric_feature_sweep_end);
WHERE
  blend = NULL;
END_ENTITY;
(*
```

PROPOSITIONS:

1. An axisymmetric feature sweep end which is an axisymmetric feature sweep spherical end must not have an associated blend.

4.8.1.90 AXISYMMETRIC FEATURE SWEEP CONICAL END

An indication that an axisymmetric feature sweep end is conical.

```
*)
ENTITY axisymmetric_feature_sweep_conical_end
  SUBTYPE OF (axisymmetric_feature_sweep_end);
  end_angle : angle_parameter;
  tip_blend : OPTIONAL implicit_edge_round;
WHERE
```

```
  {0.0 < end_angle.dimension < 180.0};
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

end_angle: The full angle or semi-angle, in degrees, of the implicit cone.

tip_blend: An optional cone tip blend.

PROPOSITIONS:

1. The cone angle must be greater than zero and less than 180 degrees.

4.8.1.91 IN OUT FEATURE SWEEP

A feature sweep whose path is roughly a plunge into or extrusion from the rest of the shape. Pockets and bosses are typical features defined with this type of sweep.

```
*)
ENTITY in_out_feature_sweep
  SUPERTYPE OF (constant_profile_in_out_sweep XOR
                tapered_profile_in_out_sweep XOR
                contoured_profile_in_out_sweep)
  SUBTYPE OF (feature_sweep);
  sweep_path      : linear_feature_sweep_path;
  sweep_profile   : closed_feature_sweep_profile;
  wall_end_blend : OPTIONAL implicit_edge_blend;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_path: The path along which the sweep profile is swept.

sweep_profile: The profile that is swept.

wall_end_blend: A blend between the sides and the end.

4.8.1.92 CONSTANT PROFILE IN OUT SWEEP

An in out feature sweep with a profile of constant size along its path.

```
*)
ENTITY constant_profile_in_out_sweep
  SUBTYPE OF (in_out_feature_sweep);
END_ENTITY;
(*
```

4.8.1.93 TAPERED_PROFILE IN OUT SWEEP

A **in out feature sweep** with sloped sides; i.e., profile size varies linearly over the length of the sweep. The dimensions given for the swept profile apply at the start of the sweep; i.e., at local $z = 0$. For $0 < z <$ (sweep length), all
dimensions are increased or decreased by $z \tan$(**semi angle**).

```
*)
ENTITY tapered_profile_in_out_sweep
  SUBTYPE OF (in_out_feature_sweep);
  semi_angle : angle_parameter;
  taper_type : taper_types;
WHERE
    (-90.0 <= semi_angle.dimension <= +90.0);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

semi_angle: The angle of inclination controlling the profile. This is the semi-angle between sweep centerline and the lines swept by points of the profile or the meeting angle of lines swept by opposing profile points.

taper_type: Whether the profile size increases or decreases as the profile is swept.

4.8.1.94 CONTOURED PROFILE IN OUT SWEEP

An **in out feature sweep** whose profile size varies according to a scaling curve. The curve $f(u)$ must be parameterized over $0 <= u <= 1$ with value $f(0) = 1$ and $f(u) > 0$ elsewhere. For any local z from 0 through the **length**, l, of the linear path of the **in out feature sweep**, the scaling factor is $f(z/l)$. The value indicates the size of a cross-section of the feature relative to its defined profile. In cases where the scaling curve is linear, it is recommended that a **tapered profile in out sweep** be used instead.

```
*)
ENTITY contoured_profile_in_out_sweep
  SUBTYPE OF (in_out_feature_sweep);
  scaling_curve : bounded_curve;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

scaling_curve: A curve that is used in the scaling function defined above.

4.8.1.95 FEATURE SWEEP PATH

A curve along which a **feature sweep profile** is swept to define the shape of an **implicit form feature**.

```
*)
ENTITY feature_sweep_path
  SUPERTYPE OF (circular_feature_sweep_path XOR
                spiral_feature_sweep_path XOR
                surface_conforming_feature_sweep_path XOR
                other_feature_sweep_path XOR
                linear_feature_sweep_path);
END_ENTITY;
(*
```

4.8.1.96 LINEAR FEATURE SWEEP PATH

A **feature sweep path** which is a straight line.

1. The sweep is along the +Z axis of the feature's LCS from $Z = 0$ to $Z = L$, where $L =$ **path length.**
2. The A- and B-axes of profiles used in linear sweeps are equated with the feature's LCS X- and Y-axes, respectively.
3. If the path is that of an **along feature sweep** which has **along feature sweep ends** defined, then the ends extend in the -Z direction from $Z = 0$ or the +Z direction from $Z = L$.

```
*)
ENTITY linear_feature_sweep_path
  SUBTYPE OF (feature_sweep_path);
  path_length : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

path_length: The length dimension of the sweep path. (Semi-length is possible, but full is preferred.)

4.8.1.97 CIRCULAR FEATURE SWEEP PATH

A **feature sweep path** which is circular.

1. The sweep is about the LCS Z-axis.
2. If the path is a **partial circular sweep path**, then the full profile interval begins at the XZ plane and runs counterclockwise as viewed from the +Z halfspace. If any **along feature sweep ends** are defined, they extend in the clockwise direction from the XZ-plane or the counterclockwise direction from the plane defined by the orientation angle attribute.
3. The AB-space of the profile is positioned and oriented in the LCS XZ-plane by:
   (a) the profile's origin is mapped onto the X-axis at $X = R$, where R is the radius, and
   (b) the angle from the Z-axis to the profile B-axis equals orientation angle, measured counterclockwise as viewed from the +Y halfspace. When the angle is 0, the profile A-axis extends in the direction of the positive X-axis.

```
*)
ENTITY circular_feature_sweep_path
  SUPERTYPE OF (complete_circular_feature_sweep_path XOR
                partial_circular_feature_sweep_path)
  SUBTYPE OF (feature_sweep_path);
  path_size          : size_parameter;
  orientation_angle : angle_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

path_size: The dimension (diameter or radius) of the circular path.

orientation_angle: An angular dimension (full- or semi-angle) which orients the AB-axes of the sweep's profile with the XZ-axes of the feature's LCS.

4.8.1.98 PARTIAL CIRCULAR FEATURE SWEEP PATH

A circular feature sweep path which is an arc of less than 360 degrees.

```
*)
ENTITY partial_circular_feature_sweep_path
  SUBTYPE OF (circular_feature_sweep_path);
  sweep_angle : angle_parameter;
WHERE
  (-360.0 < sweep_angle.dimension < 360.0);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_angle: The swept portion of the circle, given as a full angle or semi-angle.

PROPOSITIONS:

1. The sweep angle must be greater than zero and less than 360 degrees

4.8.1.99 COMPLETE CIRCULAR FEATURE SWEEP PATH

A 360 degree circular feature sweep path.

```
*)
ENTITY complete_circular_feature_sweep_path
  SUBTYPE OF (circular_feature_sweep_path);
END_ENTITY;
(*
```

4.8.1.100 SPIRAL FEATURE SWEEP PATH

A **feature sweep path** which is a spiral. The spiral may be of constant diameter or tapered. The latter is the case if and only if there is an associated **spiral taper** entity.

The swept profile is positioned as follows by the **axis placement** of the **feature sweep** which uses the spiral as sweep path:

1. AB-plane at initial position of profile; i.e., start of sweep,
2. AB-plane perpendicular to spiral,
3. B-axis intersecting centerline.

```
*)
ENTITY spiral_feature_sweep_path
  SUBTYPE OF (feature_sweep_path);
  spiral_path_length : size_parameter;
  spiral_path_size   : size_parameter;
  spiral_path_hand   : hands;
  turns_per_unit     : REAL;
  spiral_path_taper  : OPTIONAL spiral_taper;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

spiral_path_length: The length dimension, measured along its centerline, of the sweep path. (Half dimension can be specified, but full is preferred.)

spiral_path_size: The diameter or radius dimension of the spiral. In the case of a tapered spiral, this is measured at local $z = 0$.

spiral_path_hand: Whether the spiral is left or right handed, with respect to the z-axis of its LCS.

turns_per_unit: The number of times, per unit length along the centerline, that the spiral revolves about its centerline.

spiral_path_taper: The information that controls the tapering of the spiral taper.

4.8.1.101 SPIRAL TAPER

A specification of the taper of a **spiral feature sweep path** by giving the location of the spiral's apex. The apex point is located by giving its local z-coordinate. A **spiral feature sweep path** is tapered if and only if this entity is associated with it.

```
*)
ENTITY spiral_taper;
  apex_z_coord : REAL;
  angle_dim    : derivable_angle_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

apex_z_coord: The local z-coordinate of the spiral apex.

angle_dim: The taper angle dimension (full or half angle) of the sweep.

4.8.1.102 SURFACE CONFORMING FEATURE SWEEP PATH

A **feature sweep path** that is a **curve on a surface**. The swept profile maintains constant position and attitude with respect to the surface as it is swept along the path.
The **curve on surface** is used as follows:

1. Denote the surface on which the path lies as $f(U, V)$.
2. Let T be the transformation accomplished by the **axis placement** of the **feature sweep**; i.e., T transforms the swept profile from its AB-space into the desired position (in global space) relative to the initial point of the **curve on surface**. (Note: T must be such that the initial point of the **curve on surface** lies in the plane of the transformed profile.)
3. For any point $f(U, V)$ on the **curve on surface**, T transforms the profile into its position with respect to the **axis placement** defined by $f(U, V)$ the partial derivative of f with respect to U at $f(U, V)$, and the partial derivative of f with respect to V at $f(U, V)$.

```
*)
ENTITY surface_conforming_feature_sweep_path
  SUBTYPE OF (feature_sweep_path);
  reference_surface : curve_on_surface;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

reference_surface: A **curve on surface**.

4.8.1.103 OTHER FEATURE SWEEP PATH

The use of an arbitrary **curve** as a **feature sweep path**. The position and orientation of the swept profile are controlled by the path **curve** and an orienting **curve**.
The position and orientation of the swept profile are determined as follows:

1. Denote the path **curve** as $f(U)$ and the orienting **curve** as $g(U)$.
2. Let T be the transformation accomplished by the **axis placement** of the **feature sweep**; i.e., T transforms the swept profile from its AB-space into the desired position (in global space) relative to the initial point of the **curve**. (Note: T must be such that the initial point of the **curve** lies in the plane of the transformed profile.)
3. For any point $f(U)$ on the path **curve**, T transforms the profile into its position with respect to the **axis placement** defined by $f(U)$, the tangent vector of f at U, and the tangent vector of g at U.

```
*)
ENTITY other_feature_sweep_path
```

```
  SUBTYPE OF (feature_sweep_path);
  path_def     : curve;
  orientation : curve;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

path_def: The curve that serves as the path.

orientation: The curve that controls orientation of the swept profile along the path.

4.8.1.104 FEATURE SWEEP PROFILE

A planar curve or connected set of curves which is swept through space along a feature sweep path to define an implicit form feature. Profiles are defined in a local AB-space which is mapped into the feature's LCS according to conventions based on the type of path. Conventions for the orientation of the profile within the AB-space are specified for the type of profile.

```
*)
ENTITY feature_sweep_profile
  SUPERTYPE OF (closed_feature_sweep_profile XOR
                open_feature_sweep_profile);
END_ENTITY;
(*
```

4.8.1.105 CLOSED FEATURE SWEEP PROFILE

A feature sweep profile that is a closed curve or a closed composite curve.

```
*)
ENTITY closed_feature_sweep_profile
  SUPERTYPE OF (other_closed_feature_sweep_profile XOR
                standard_closed_feature_sweep_profile)
  SUBTYPE OF (feature_sweep_profile);
END_ENTITY;
(*
```

4.8.1.106 STANDARD CLOSED FEATURE SWEEP PROFILE

A common, "standard" closed profile used to define swept features.

```
*)
ENTITY standard_closed_feature_sweep_profile
  SUPERTYPE OF (rectangular_feature_sweep_profile XOR
                ngon_feature_sweep_profile)
  SUBTYPE OF (closed_feature_sweep_profile);
  blend : OPTIONAL implicit_edge_blend;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

blend: The definition of the desired blend.

4.8.1.107 RECTANGULAR FEATURE SWEEP PROFILE

A rectangle used as a closed feature sweep profile.
The rectangle is oriented with its center at the origin of the local AB-plane. The sides given by the LENGTH attribute are parallel to the A-axis.

```
*)
ENTITY rectangular_feature_sweep_profile
  SUBTYPE OF (standard_closed_feature_sweep_profile);
  sweep_length : size_parameter;
  sweep_width  : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_length: The full length or semi-length dimension of the rectangle.

sweep_width: The full width or semi-width dimension of the rectangle.

4.8.1.108 NGON FEATURE SWEEP PROFILE

A regular n-gon used as a standard closed feature sweep profile.
The N-GON is defined with its center at the local AB-origin. One of its sides is parallel to the A-axis.

```
*)
ENTITY ngon_feature_sweep_profile
  SUBTYPE OF (standard_closed_feature_sweep_profile);
  number_of_sides    : INTEGER;
  inscribed_dim      : size_parameter;
  circumscribed_dim  : derivable_dimension;
  side_length        : derivable_dimension;
  interior_angle     : derivable_angle_dimension;
  exterior_angle     : derivable_angle_dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

number_of_sides: The number of sides the n-gon is to contain.

inscribed_dim: The dimension (radius or diameter) of a circle inscribed in the n-gon.

circumscribed_dim: The derivable dimension (radius or diameter) of a circle circumscribed about the n-gon.

side_length: The derivable length (half or full dimension) of a side of the n-gon.

interior_angle: The derivable interior angle or semi-angle between sides.

exterior_angle: The derivable exterior angle or semi-angle between sides. This is the angle between one side and the extension of an adjacent side; i.e., the supplement of the interior angle.

4.8.1.109 OTHER CLOSED FEATURE SWEEP PROFILE

The use of a **closed general profile** as a **closed feature sweep profile.**

```
*)
ENTITY other_closed_feature_sweep_profile
  SUBTYPE OF (closed_feature_sweep_profile);
  sweep_profile : closed_general_profile;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_profile: The definition of the sweep profile.

4.8.1.110 OPEN FEATURE SWEEP PROFILE

A **feature sweep profile** consisting of a curve or composite curve which does not close. The open ends of the profile are assumed to extend to the material/air interface. For example, only the width and blends are specified for a "U"-shaped profile; the lengths of the vertical bars are determined by the profile's placement.

```
*)
ENTITY open_feature_sweep_profile
  SUPERTYPE OF (circular_arc_feature_sweep_profile XOR
                rounded_u_feature_sweep_profile XOR
                vee_feature_sweep_profile XOR
                square_u_feature_sweep_profile XOR
                tee_feature_sweep_profile XOR
                ell_feature_sweep_profile XOR
                line_plus_radius_feature_sweep_profile XOR
                half_obround_feature_sweep_profile XOR
                other_open_feature_sweep_profile)
  SUBTYPE OF (feature_sweep_profile);
END_ENTITY;
(*
```

4.8.1.111 CIRCULAR ARC FEATURE SWEEP PROFILE

An **open feature sweep profile** consisting of a circular arc.
The center of the defining circle lies on the profile's local AB-origin. The midpoint of the arc is on the negative B-axis.

```
*)
ENTITY circular_arc_feature_sweep_profile
```

```
  SUBTYPE OF (open_feature_sweep_profile);
  arc_size : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

arc_size: The dimension (radius or diameter) of the arc.

4.8.1.112 ROUNDED U FEATURE SWEEP PROFILE

An **open feature sweep profile** consisting of two parallel semi-infinite lines connected at their fixed ends by a semi-circle to form a "U" shape. The center of the semi-circle lies on the profile's local origin, and the parallel lines are parallel to the B-axis and in the positive B half-plane.
The center of the semi-circle lies on the profile's local origin, and the parallel lines are parallel to the B-axis and lie in the positive B half-plane.

```
*)
ENTITY rounded_u_feature_sweep_profile
  SUBTYPE OF (open_feature_sweep_profile);
  sweep_width : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_width: The width (full or half) dimension of the profile.

4.8.1.113 VEE FEATURE SWEEP PROFILE

An **open feature sweep profile** in the shape of a "V".
The profile is symmetric with respect to the positive B-axis, with the point of the V at the origin.

```
*)
ENTITY vee_feature_sweep_profile
  SUBTYPE OF (open_feature_sweep_profile);
  vee_angle : angle_parameter;
  blend     : OPTIONAL implicit_edge_blend;
WHERE
  (0.0 < vee_angle.dimension < 180.0);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

vee_angle: The full angle or semi-angle of the V.

blend: The blend at the point of the V.

PROPOSITIONS:

1. The full angle must be greater than zero and less than 180 degrees.

4.8.1.114 SQUARE_U FEATURE SWEEP PROFILE

An **open feature sweep profile** consisting of two parallel semi-infinite lines connected at their fixed ends by a line segment perpendicular to both.

The profile lies in the positive B half-plane with its line segment on the A-axis. The midpoint of the segment is at the origin.

```
*)
ENTITY square_u_feature_sweep_profile
  SUBTYPE OF (open_feature_sweep_profile);
  sweep_width  : size_parameter;
  blend1       : OPTIONAL implicit_edge_blend;
  blend2       : OPTIONAL implicit_edge_blend;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_width: The length of the line segment.

blend1: The blend at the bottom lefthand corner.

blend2: The blend at the bottom righthand corner.

4.8.1.115 TEE FEATURE SWEEP PROFILE

A profile in the shape of a "T". It consists of a *stem* and a *cross-bar*. The stem is represented by two parallel semi-infinite lines connected at their fixed ends by the cross-bar which is a rectangle. The corners of the profile may be blended.

The side of the cross-bar opposite the stem lies on the profile's X axis, the rest of the profile is in the positive Y half-plane and the profile is symmetric to the local Y axis (i.e the T is upside down in the normal XY view).

The side of the crossbar opposite the stem lies on the profile's A-axis, the rest of the profile is in the positive B half-plane, and the profile is symmetric with respect to the local B-axis.

```
*)
ENTITY tee_feature_sweep_profile
  SUBTYPE OF (open_feature_sweep_profile);
  stem_width            : size_parameter;
  crossbar_width        : size_parameter;
  crossbar_height       : size_parameter;
  stem_crossbar_blend1  : OPTIONAL implicit_edge_blend;
  stem_crossbar_blend2  : OPTIONAL implicit_edge_blend;
  crossbar_blend1       : OPTIONAL implicit_edge_blend;
  crossbar_blend2       : OPTIONAL implicit_edge_blend;
  crossbar_blend3       : OPTIONAL implicit_edge_blend;
  crossbar_blend4       : OPTIONAL implicit_edge_blend;
WHERE
  crossbar_width.dimension > stem_width.dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

stem_width: The width or half-width dimension of the stem (X dimension).

crossbar_width: The width or half-width dimension of the crossbar (X dimension).

crossbar_height: The height or half-height dimension of the crossbar (Y dimension).

stem_crossbar_blend1: The blend at the lefthand concave corner between the stem and crossbar.

stem_crossbar_blend2: The blend at the righthand concave corner between the stem and crossbar.

crossbar_blend1: The blend at the top lefthand corner of the crossbar.

crossbar_blend2: The blend at the bottom lefthand corner of the crossbar.

crossbar_blend3: The blend at the bottom righthand corner of the crossbar.

crossbar_blend4: The blend at the top righthand corner of the crossbar.

4.8.1.116 ELL FEATURE SWEEP PROFILE

An **open feature sweep profile** in the shape of an "L". It consists of a *stem* and a perpendicular *endbar*. The stem is represented by two parallel semi-infinite lines connected at their fixed ends by the endbar. The endbar is a rectangle.
The endbar is in the positive B half-plane with the side opposite the stem lying on the A-axis. The stem is symmetric with respect to the B-axis. The ell profile orientation attribute indicates whether the endbar is primarily in the positive or negative A half-plane.

```
*)
ENTITY ell_feature_sweep_profile
  SUBTYPE OF (open_feature_sweep_profile);
  stem_width         : size_parameter;
  endbar_width       : size_parameter;
  endbar_height      : size_parameter;
  ell_orientation    : ell_orientation_types;
  stem_endbar_blend  : OPTIONAL implicit_edge_blend;
  endbar_blend1      : OPTIONAL implicit_edge_blend;
  endbar_blend2      : OPTIONAL implicit_edge_blend;
  endbar_blend3      : OPTIONAL implicit_edge_blend;
WHERE
  endbar_width.dimension > stem_width.dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

stem_width: The width or half-width (X dimension) of the stem.

endbar_width: The width or half-width (X dimension) of the end-bar.

endbar_height: The height or half-height (Y dimension) of the end-bar.

ell_orientation: Whether the end of the end-bar is in the positive or negative X halfplane.

stem_endbar_blend: The blend at the concave corner between the stem and end-bar.

endbar_blend1: The blend at the bottom lefthand corner of the L.

endbar_blend2: The blend at the bottom righthand corner of the L.

endbar_blend3: The blend at the remaining corner of the L.

PROPOSITIONS:

1. The width of the endbar must be greater than the width of the stem.

4.8.1.117 LINE PLUS RADIUS FEATURE SWEEP PROFILE

An open feature sweep profile consisting of a semi-infinite line and a tangent semi-infinite circular arc connected at their fixed ends.

The intersection point of the arc and line is at the profile's local origin. The line coincides with the local A-axis, and the circular arc extends into the positive B half-plane.

```
*)
ENTITY line_plus_radius_feature_sweep_profile
  SUBTYPE OF (open_feature_sweep_profile);
  size_dim : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

size_dim: The radius (preferably) or the diameter dimension of the circular portion of the profile.

4.8.1.118 HALF OBROUND FEATURE SWEEP PROFILE

An open feature sweep profile consisting of two semi-infinite circular arcs connected at their fixed ends by a line segment.

The line segment is on the local A-axis with its center at the origin. The arcs are tangent to the segment, and extend into the positive B half-plane.

```
*)
ENTITY half_obround_feature_sweep_profile
  SUBTYPE OF (open_feature_sweep_profile);
  circle_dim : size_parameter;
  length_dim : size_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

circle_dim: The common radius (preferably) or diameter dimension of the two circular portions of the profile.

length_dim: The length (full or half) of the linear portion of the profile.

4.8.1.119 OTHER OPEN FEATURE SWEEP PROFILE

The use of an open general profile as an open feature sweep profile.

```
*)
ENTITY other_open_feature_sweep_profile
  SUBTYPE OF (open_feature_sweep_profile);
  sweep_profile : open_general_profile;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sweep_profile: The definition of the sweep profile

4.8.1.120 GENERAL PROFILE

A sequence of non-overlapping curve segments in a plane, connected end to end.

```
*)
ENTITY general_profile
  SUPERTYPE OF (open_general_profile XOR
                closed_general_profile);
  first_curve   : curve;
  element_pairs : SET [0:#] OF profile_pair;
WHERE
  sequenced(first_curve,element_pairs);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

first_curve: The first curve of a general profile.

element_pairs: A pairing of two contiguous elements of a profile.

4.8.1.121 PROFILE PAIR

A predecessor/successor relationship between two curves of a general profile. The sequence of curves that constitutes the profile is modeled by a set of these.

```
*)
ENTITY profile_pair;
  predecessor_curve : curve;
  successor_curve   : curve;
  blend             : OPTIONAL implicit_edge_blend;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

predecessor_curve: The first element of the element pair.

successor_curve: The second element of the element pair.

blend: A definition of the desired blend.

PROPOSITIONS:

1. A given CURVE can appear at most once in a given GENERAL PROFILE.

4.8.1.122 OPEN GENERAL PROFILE

A general profile that does not enclose an area of the plane in which it exists.

```
*)
ENTITY open_general_profile
  SUBTYPE OF (general_profile);
WHERE
  NOT closed(element_pairs);
  planar(element_pairs);
END_ENTITY;
(*
```

PROPOSITIONS:

1. The

4.8.1.123 CLOSED GENERAL PROFILE

A general profile that encloses an area of the plane in which it exists.

```
*)
ENTITY closed_general_profile
  SUBTYPE OF (general_profile);
WHERE
  closed(element_pairs);
  planar(element_pairs);
END_ENTITY;
(*
```

PROPOSITIONS:

1. The

4.8.2 Form Feature IPIM Classification Structure

The following indented listing shows the classification structure for the Integrated Form Features model.

ALONG FEATURE SWEEP END
  ALONG FEATURE SWEEP FLAT END
  ALONG FEATURE SWEEP RADIUSED END
ARRAY PATTERN OMISSION
AXISYMMETRIC FEATURE SWEEP END
  AXISYMMETRIC FEATURE SWEEP CONICAL END
  AXISYMMETRIC FEATURE SWEEP FLAT END
  AXISYMMETRIC FEATURE SWEEP SPHERICAL END
BEND DIMENSION
BEND POINT
CIRCULAR PATTERN OFFSET MEMBER
CIRCULAR PATTERN OMISSION
COUPLING TIMER
DEPRESSION INTERMEDIATE BOUND
EDGE BLENDED INTERSECTION
FEATURE SWEEP PATH
  CIRCULAR FEATURE SWEEP PATH
    COMPLETE CIRCULAR FEATURE SWEEP PATH
    PARTIAL CIRCULAR FEATURE SWEEP PATH
  LINEAR FEATURE SWEEP PATH
  OTHER FEATURE SWEEP PATH
  SPIRAL FEATURE SWEEP PATH
  SURFACE CONFORMING FEATURE SWEEP PATH
FEATURE SWEEP PROFILE
  CLOSED FEATURE SWEEP PROFILE
    OTHER CLOSED FEATURE SWEEP PROFILE
    STANDARD CLOSED FEATURE SWEEP PROFILE
      NGON FEATURE SWEEP PROFILE
      RECTANGULAR FEATURE SWEEP PROFILE
  OPEN FEATURE SWEEP PROFILE
    CIRCULAR ARC FEATURE SWEEP PROFILE
    ELL FEATURE SWEEP PROFILE
    HALF OBROUND FEATURE SWEEP PROFILE
    LINE PLUS RADIUS FEATURE SWEEP PROFILE
    OTHER OPEN FEATURE SWEEP PROFILE
    ROUNDED U FEATURE SWEEP PROFILE
    SQUARE U FEATURE SWEEP PROFILE
    TEE FEATURE SWEEP PROFILE
    VEE FEATURE SWEEP PROFILE
FEATURE VOLUME
  FEATURE RULING
  FEATURE SWEEP
    ALONG FEATURE SWEEP
    AXISYMMETRIC FEATURE SWEEP
      CONSTANT DIAMETER AXISYMMETRIC FEATURE SWEEP
      OTHER AXISYMMETRIC FEATURE SWEEP
      TAPERED AXISYMMETRIC FEATURE SWEEP
    IN OUT FEATURE SWEEP
      CONTOURED PROFILE IN OUT SWEEP

```
      CONSTANT PROFILE IN OUT SWEEP
      TAPERED PROFILE IN OUT SWEEP
FORM FEATURE
FULL THREADING SPECIFICATION
GENERAL PROFILE
  CLOSED GENERAL PROFILE
  OPEN GENERAL PROFILE
GEOMETRIC MODEL IMPLICIT FORM FEATURE ASSN
IMPLICIT FEATURE BOUND
IMPLICIT FEATURE PRECEDENCE
IMPLICIT FORM FEATURE
  IMPLICIT AREA FEATURE
    IMPLICIT COUPLING
    IMPLICIT KNURL
      IMPLICIT DIAGONALKNURL
      IMPLICIT DIAMOND KNURL
      IMPLICIT STRAIGHT KNURL
    IMPLICIT MARKING
    IMPLICIT THREAD
  IMPLICIT DEFORMATION
    IMPLICIT BEND
      IMPLICIT CUTOUT FLANGE
      IMPLICIT GENERAL BEND
      IMPLICIT STRAIGHT BEND
    IMPLICIT EMBOSS
      IMPLICIT CORNER RIB
      IMPLICIT ROUND BEAD
      IMPLICIT SPHERICAL EMBOSS
      IMPLICIT V BEAD
    IMPLICIT PARTIAL CUTOUT
      IMPLICIT CIRCULAR KNOCKOUT
      IMPLICIT LOUVER
      IMPLICIT TAB
    IMPLICIT TUBE DEFORMATION
      IMPLICIT TUBE BEND
      IMPLICIT TUBE FLARE
      IMPLICIT TUBE FLATTENING
      IMPLICIT TUBE NECK
      IMPLICIT TUBE ROLL
    IMPLICIT TWIST
  IMPLICIT DEPRESSION
  IMPLICIT PASSAGE
  IMPLICIT PROTRUSION
  IMPLICIT TRANSITION
    IMPLICIT CORNER BLEND
      IMPLICIT CORNER FLAT
      IMPLICIT OUTSIDE CORNER ROUND
    IMPLICIT EDGE BLEND
      IMPLICIT EDGE FLAT
```

```
      IMPLICIT EDGE ROUND
IMPLICIT FORM FEATURE PATTERN
  IMPLICIT ARRAY FORM FEATURE PATTERN
    PARALLEL EQUAL SPACING ARRAY PATTERN
    PARAMETRIC EQUAL SPACING ARRAY PATTERN
  IMPLICIT CIRCULAR FORM FEATURE PATTERN
PARALLEL ARRAY PATTERN OFFSET MEMBER
PARAMETRIC ARRAY PATTERN OFFSET MEMBER
PASSAGE BOUND
PASSAGE BOUNDARY BLEND
PASSAGE INTERMEDIATE BOUND
PITCH APEX
PROFILE PAIR
REPLICATE FORM FEATURE
SPIRAL TAPER
THREAD TIMER

*)
END_SCHEMA; -- end of FEATURES schema
(*
```

4.9 Shape Representation Interface

4.9.1 INTEGRATION CORE MODEL INTRODUCTION

This Section describes the Integration Core Model as developed by the Integration Subcommittee of the Logical Layer. The intention of this model is to create the interface points between the resource models and the application models in a manner that allows the referencing models to ignore the details of shape representation.

The primary value added by the Integration Core Model is the separation of the concepts of referencing the product's shape aspects, and referencing the representation of those shape aspects within the several Geometric Models. The idea of an aspect of a shape is available in this model for referencing totally independent of how that shape aspect has been represented or modeled.

4.9.2 INTEGRATION SCHEMA

```
*)
SCHEMA ipim_shape_interface_schema;

EXPORT EVERYTHING;

ASSUME (ipim_design_shape_schema,
        ipim_features_schema,
        ipim_geometry_schema,
        ipim_nominal_shape_schema,
        ipim_solids_schema,
        ipim_topology_schema);

(*
```

4.9.2.1 INTEGRATION TYPE DEFINITIONS

4.9.2.1.1 GEOMETRIC MODEL

A formal logical/mathematical representation of a shape.

```
*)
TYPE geometric_model = SELECT
     (manifold_solid_brep,
      facetted_brep,
      csg_solid,
      csg_primitive,
      surface_model,
      half_space,
      wireframe_model,
      solid_of_revolution,
      solid_of_linear_extrusion,
      geometric_set,
      assembly_model);
END_TYPE;
(*
```

4.9.2.1.2 PARTIAL_REV SOR FACE TYPES

```
*)
TYPE partial_rev_sor_face_types = ENUMERATION OF
     (starting_sor_face,
      ending_sor_face);
END_TYPE;
(*
```

4.9.2.1.3 SOLE FACE TYPES

```
*)
TYPE sole_face_types = ENUMERATION OF
     (starting_sole_face,
      ending_sole_face);
END_TYPE;
(*
```

4.9.2.2 SHAPE

The size, spatial configuration, and proportions of a real or conceived thing, independent of whether or how it is represented.

```
*)
ENTITY shape;
  elements : LIST [0:#] OF shape_element;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

elements: A list of the individual **shape elements** that comprise the shape of the product.

4.9.2.3 SHAPE ELEMENT

A distinguishable aspect of a **shape.**

```
*)
ENTITY shape_element
  SUPERTYPE OF (dimensionality_3_shape_element XOR
                dimensionality_2_shape_element XOR
                dimensionality_1_shape_element XOR
                dimensionality_0_shape_element);
END_ENTITY;
(*
```

4.9.2.4 DIMENSIONALITY 3 SHAPE ELEMENT

A **shape element** that has dimensionality three, i.e., that is a volume.

```
*)
ENTITY dimensionality_3_shape_element
  SUPERTYPE OF (object_shape_element XOR
                object_assembly_shape_element)
  SUBTYPE OF (shape_element);
  d3se_representations : LIST [0:#] OF
                                dim_3_shape_element_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

d3se_representations: The aggregation of the modeling representations to be associated with the dimensionality 3 shape element.

4.9.2.5 OBJECT SHAPE ELEMENT

A dimensionality 3 shape element that is arcwise connected. It may have interior voids.

```
*)
ENTITY object_shape_element
  SUBTYPE OF (dimensionality_3_shape_element);
  obj_representations : LIST [0:#] OF object_representation;
  voidless_volume     : OPTIONAL voidless_volume_shape_element;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

obj_representations: The aggregation of the modeling representations to be associated with the object shape element.

voidless_volume : An indication that the OBJECT is a voidless volume.

4.9.2.6 VOIDLESS VOLUME SHAPE ELEMENT

An object shape element that contains no voids.

```
*)
ENTITY voidless_volume_shape_element;
  voidless_vol_representations : LIST [0:#] OF
                                        voidless_volume_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

voidless_vol_representations: The aggregation of the modeling representations to be associated with the voidless volume shape element.

4.9.2.7 OBJECT ASSEMBLY SHAPE ELEMENT

A **dimensionality 3 shape element** that is a collection of two or more positioned **object shape elements.**

```
*)
ENTITY object_assembly_shape_element
  SUBTYPE OF (dimensionality_3_shape_element);
  obj_assy_representations : LIST [0:#] OF
                                    object_assembly_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

obj_assy_representations: The aggregation of the modeling representations to be associated with the **object assembly shape element.**

4.9.2.8 DIMENSIONALITY 2 SHAPE ELEMENT

A **shape element** that has dimensionality two, i.e., that is a portion of the surface area of a shape.

```
*)
ENTITY dimensionality_2_shape_element
  SUPERTYPE OF (zone_shape_element XOR
                area_shape_element)
  SUBTYPE OF (shape_element);
  feature : OPTIONAL form_feature;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

feature : A reference to a **form feature** that is used for a **dimensionality 2 shape element.**

4.9.2.9 AREA SHAPE ELEMENT

A **dimensionality 2 shape element** that is arcwise connected and has a uniform underlying mathematical surface.

```
*)
ENTITY area_shape_element
  SUPERTYPE OF (maximal_area_shape_element XOR
                nonmaximal_area_shape_element)
  SUBTYPE OF (dimensionality_2_shape_element);
  area_representations : LIST [0:#] OF area_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

area_representations: The aggregation of the modeling representations to be associated with the **area shape element.**

4.9.2.10 MAXIMAL AREA SHAPE ELEMENT

An **area shape element** that has no adjacent **area shape element** with the same underlying mathematical surface.

```
*)
ENTITY maximal_area_shape_element
  SUBTYPE OF (area_shape_element);
  max_area_representations : LIST [0:#] OF
                                    maximal_area_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

max_area_representations: The aggregation of the modeling representations to be associated with the **maximal area shape element.**

4.9.2.11 NONMAXIMAL AREA SHAPE ELEMENT

An **area shape element** that has at least one adjacent **area shape element** with the same underlying mathematical surface.

```
*)
ENTITY nonmaximal_area_shape_element
  SUBTYPE OF (area_shape_element);
  nonmax_area_representations : LIST [0:#] OF
                                       nonmaximal_area_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

nonmax_area_representations: The aggregation of the modeling representations to be associated with the **nonmaximal area shape element.**

4.9.2.12 ZONE SHAPE ELEMENT

A **dimensionality 2 shape element** that is the union of other **dimensionality 2 shape elements.**

```
*)
ENTITY zone_shape_element
  SUBTYPE OF (dimensionality_2_shape_element);
  components : LIST [1:#] OF zone_shape_element_component;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

components: An aggregation of the constituents of the **zone shape element.**

4.9.2.13 ZONE SHAPE ELEMENT COMPONENT

A **dimensionality 2 shape element** that is part of a **zone shape element.**

```
*)
ENTITY zone_shape_element_component;
  element : dimensionality_2_shape_element;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

element : The reference to the **dimensionality 2 shape element** that is a component of a **zone shape element.**

4.9.2.14 DIMENSIONALITY 1 SHAPE ELEMENT

A **shape element** that has no discontinuity and has dimensionality one, i.e., that is a path on the surface area of a shape.

```
*)
ENTITY dimensionality_1_shape_element
  SUPERTYPE OF (seam_shape_element XOR
                perimeter_shape_element)
  SUBTYPE OF (shape_element);
END_ENTITY;
(*
```

4.9.2.15 SEAM SHAPE ELEMENT

A **dimensionality 1 shape element** that has a uniform underlying mathematical curve.

```
*)
ENTITY seam_shape_element
  SUPERTYPE OF (edge_shape_element XOR
                subedge_shape_element XOR
                interior_seam_shape_element)
  SUBTYPE OF (dimensionality_1_shape_element);
  seam_representations : LIST [0:?] OF seam_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

seam_representations: The aggregation of the modeling representations to be associated with the **seam shape element.**

4.9.2.16 EDGE SHAPE ELEMENT

A **seam shape element** that is the intersection of two **area shape elements**.

```
*)
ENTITY edge_shape_element
  SUBTYPE OF (seam_shape_element);
END_ENTITY;
(*
```

4.9.2.17 SUBEDGE SHAPE ELEMENT

A **seam shape element** that is a proper subset of an **edge shape element**.

```
*)
ENTITY subedge_shape_element
  SUBTYPE OF (seam_shape_element);
END_ENTITY;
(*
```

4.9.2.18 INTERIOR SEAM SHAPE ELEMENT

A **seam shape element** that, with the possible exception of one or both of its end points, lies in the interior of an **area shape element**.

```
*)
ENTITY interior_seam_shape_element
  SUBTYPE OF (seam_shape_element);
END_ENTITY;
(*
```

4.9.2.19 PERIMETER SHAPE ELEMENT

A **dimensionality 1 shape element** that is a closed boundary of an **area shape element**.

```
*)
ENTITY perimeter_shape_element
  SUBTYPE OF (dimensionality_1_shape_element);
  perimeter_representations : LIST [0:#] OF
                                  perimeter_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

perimeter_representations: The aggregation of the modeling representations to be associated with the **perimeter shape element**.

4.9.2.20 DIMENSIONALITY 0 SHAPE ELEMENT

A **shape element** that has dimensionality zero, i.e., that is a location on the surface area of a shape.

```
*)
ENTITY dimensionality_0_shape_element
  SUPERTYPE OF (corner_shape_element XOR
                boundary_location_shape_element XOR
                interior_location_shape_element)
  SUBTYPE OF (shape_element);
  d0se_representations : LIST [0:#] OF
                              dim_0_shape_element_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

d0se_representations: The aggregation of the modeling representations to be associated with the **dimensionality 0 shape element**.

4.9.2.21 CORNER SHAPE ELEMENT

A **dimensionality 0 shape element** that is the intersection of three or more **area shape elements** or that is the apex of a cone-like shape.

```
*)
ENTITY corner_shape_element
  SUBTYPE OF (dimensionality_0_shape_element);
  corner_representations : LIST [0:#] OF corner_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

corner_representations: The aggregation of the modeling representations to be associated with the **corner shape element**.

4.9.2.22 BOUNDARY LOCATION SHAPE ELEMENT

A **dimensionality 0 shape element** that is on an **edge shape element**, but that is not an end point.

```
*)
ENTITY boundary_location_shape_element
  SUBTYPE OF (dimensionality_0_shape_element);
  bdry_loc_representations : LIST [0:#] OF
                                  partial_rev_sor_bdry_loc_rep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

bdry_loc_representations: The aggregation of the modeling representations to be associated with the **boundary location shape element**.

4.9.2.23 INTERIOR LOCATION SHAPE ELEMENT

A dimensionality 0 shape element that is in the interior of an area shape element.

```
*)
ENTITY interior_location_shape_element
  SUBTYPE OF (dimensionality_0_shape_element);
  int_loc_representations : LIST [0:#] OF
                                    interior_location_representation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

int_loc_representations: The aggregation of the modeling representations to be associated with the interior location shape element.

4.9.2.24 PAIR OF EQUIVALENT GEOMETRIC MODELS

Identifies alternative representations of shape.

```
*)
ENTITY pair_of_equivalent_geometric_models;
  model_one : geometric_model;
  model_two : geometric_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

model_one: One of the geometric models that are equivalent.

model_two: Another of the geometric models that are equivalent.

4.9.2.25 DIM 3 SHAPE ELEMENT REPRESENTATION

The symbolic description of a dimensionality 3 shape element in a geometric model.

```
*)
ENTITY dim_3_shape_element_representation
  SUPERTYPE OF (csg_solid_dim_3_se_rep XOR
                surface_dim_3_se_rep XOR
                wireframe_dim_3_se_rep XOR
                geometric_set_dim_3_se_rep);
END_ENTITY;
(*
```

4.9.2.26 CSG SOLID DIM 3 SE REP

The use of a csg solid model to represent a dimensionality 3 shape element.

```
*)
ENTITY csg_solid_dim_3_se_rep
  SUBTYPE OF (dim_3_shape_element_representation);
  definition : csg_solid;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.27 SURFACE DIM 3 SE REP

The use of a surface model to represent a dimensionality 3 shape element.

```
*)
ENTITY surface_dim_3_se_rep
  SUBTYPE OF (dim_3_shape_element_representation);
  definition : surface_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.28 WIREFRAME DIM 3 SE REP

The use of a wireframe model to represent a dimensionality 3 shape element.

```
*)
ENTITY wireframe_dim_3_se_rep
  SUBTYPE OF (dim_3_shape_element_representation);
  definition : wireframe_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.29 GEOMETRIC SET DIM 3 SE REP

The use of a geometry in a geometric set to represent a dimensionality 3 shape element.

```
*)
ENTITY geometric_set_dim_3_se_rep
  SUBTYPE OF (dim_3_shape_element_representation);
  definition : geometric_set;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.30 OBJECT REPRESENTATION

The symbolic description of an object shape element in a geometric model.

```
*)
ENTITY object_representation
  SUPERTYPE OF (brep_object_rep XOR
                facetted_brep_object_rep XOR
                full_rev_sor_object_rep);
END_ENTITY;
(*
```

4.9.2.31 BREP OBJECT REP

The use of a manifold solid brep model to represent an object shape element.

```
*)
ENTITY brep_object_rep
  SUBTYPE OF (object_representation);
  definition : manifold_solid_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.32 FACETTED BREP OBJECT REP

The use of a facetted brep to represent an object shape element.

```
*)
ENTITY facetted_brep_object_rep
  SUBTYPE OF (object_representation);
  definition : facetted_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.33 FULL REV SOR OBJECT REP

The use of a fully revolved solid of revolution to represent an object shape element.

```
*)
ENTITY full_rev_sor_object_rep
  SUBTYPE OF (object_representation);
  definition : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.34 VOIDLESS VOLUME REPRESENTATION

The symbolic description of a voidless volume shape element in a geometric model.

```
*)
ENTITY voidless_volume_representation
  SUPERTYPE OF (brep_volume_rep XOR
                facetted_brep_volume_rep XOR
                csg_primitive_volume_rep XOR
                half_space_volume_rep XOR
                partial_rev_sor_volume_rep XOR
                full_rev_sor_volume_rep XOR
                sole_volume_rep);
END_ENTITY;
(*
```

4.9.2.35 BREP VOLUME REP

The use of a shell in a manifold solid brep to represent a voidless volume shape element.

```
*)
ENTITY brep_volume_rep
  SUBTYPE OF (voidless_volume_representation);
  definition : shell;
  context    : manifold_solid_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.36 FACETTED BREP VOLUME REP

The use of a **shell** in a **facetted brep** to represent a **voidless volume shape element.**

```
*)
ENTITY facetted_brep_volume_rep
  SUBTYPE OF (voidless_volume_representation);
  definition : shell;
  context    : facetted_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular **geometric model** that the definition belongs to.

4.9.2.37 CSG PRIMITIVE VOLUME REP

The use of a **csg primitive** to represent a **voidless volume shape element.**

```
*)
ENTITY csg_primitive_volume_rep
  SUBTYPE OF (voidless_volume_representation);
  definition : csg_primitive;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

defintion: The reference to the defining modeling information for this representation.

4.9.2.38 HALF SPACE VOLUME REP

The use of a **half space** to represent a **voidless volume shape element.**

```
*)
ENTITY half_space_volume_rep
  SUBTYPE OF (voidless_volume_representation);
  definition : half_space;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

defintion: The reference to the defining modeling information for this representation.

4.9.2.39 PARTIAL REV SOR VOLUME REP

The use of a partially revolved **solid of revolution** to represent a **voidless volume shape element.**

```
*)
ENTITY partial_rev_sor_volume_rep
  SUBTYPE OF (voidless_volume_representation);
  definition : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.40 FULL REV SOR VOLUME REP

The use of a **loop logical structure** in a fully revolved **solid of revolution** to represent a **voidless volume shape element.**

```
*)
ENTITY full_rev_sor_volume_rep
  SUBTYPE OF (voidless_volume_representation);
  definition : loop_logical_structure;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular **geometric model** that the definition belongs to.

4.9.2.41 SOLE VOLUME REP

The use of a **solid of linear extrusion** to represent a **voidless volume shape element.**

```
*)
ENTITY sole_volume_rep
  SUBTYPE OF (voidless_volume_representation);
  definition : solid_of_linear_extrusion;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.42 OBJECT ASSEMBLY REPRESENTATION

The symbolic description of an **object assembly shape element** in a **geometric model.**

```
*)
ENTITY object_assembly_representation;
  definition : assembly_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.43 AREA REPRESENTATION

The symbolic description of an **area shape element** in a **geometric model.**

```
*)
ENTITY area_representation
  SUPERTYPE OF (brep_area_rep XOR
                facetted_brep_area_rep XOR
                surface_area_rep XOR
                wireframe_area_rep XOR
                sor_edge_area_rep XOR
                sole_edge_area_rep XOR
                geometric_set_area_rep);
END_ENTITY;
(*
```

4.9.2.44 BREP AREA REP

The use of a **face** in a **manifold solid brep** to represent an **area shape element.**

```
*)
ENTITY brep_area_rep
  SUBTYPE OF (area_representation);
  definition : face;
  context    : manifold_solid_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular **geometric model** that the definition belongs to.

4.9.2.45 FACETTED BREP AREA REP

The use of a face in a facetted brep to represent an area shape element.

```
*)
ENTITY facetted_brep_area_rep
  SUBTYPE OF (area_representation);
  definition : face;
  context    : facetted_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.46 SURFACE AREA REP

The use of a face in a surface model to represent an area shape element.

```
*)
ENTITY surface_area_rep
  SUBTYPE OF (area_representation);
  defintion : face;
  context   : surface_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.47 PARTIAL REV SOR MAX AREA REP

The use of a partially revolved solid of revolution to represent a maximal area shape element.

```
*)
ENTITY partial_rev_sor_max_area_rep
  SUBTYPE OF (maximal_area_representation);
  definition : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.48 MAXIMAL AREA REPRESENTATION

The symbolic description of a maximal area shape element in a geometric model.

```
*)
ENTITY maximal_area_representation
  SUPERTYPE OF (sole_max_area_rep XOR
                partial_rev_sor_max_area_rep);
END_ENTITY;
(*
```

4.9.2.49 SOLE MAX AREA REP

The use of a solid of linear extrusion to represent a maximal area shape element.

```
*)
ENTITY sole_max_area_rep
  SUBTYPE OF (maximal_area_representation);
  definition : solid_of_linear_extrusion;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

4.9.2.50 WIREFRAME AREA REP

The use of a loop in a wireframe model to represent an area shape element.

```
*)
ENTITY wireframe_area_rep
  SUBTYPE OF (area_representation);
  definition : loop;
  context    : wireframe_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.51 SOR EDGE AREA REP

The use of an edge in a solid of revolution to represent an area shape element.

```
*)
ENTITY sor_edge_area_rep
```

```
  SUBTYPE OF (area_representation);
  definition : edge;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.52 SOLE EDGE AREA REP

The use of an edge in a solid of linear extrusion to represent an area shape element.

```
*)
ENTITY sole_edge_area_rep
  SUBTYPE OF (area_representation);
  definiton : edge;
  context   : solid_of_linear_extrusion;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.53 GEOMETRIC SET AREA REP

The use of a geometry in a geometric set to represent an area shape element.

```
*)
ENTITY geometric_set_area_rep
  SUBTYPE OF (area_representation);
  definition : geometry;
  context    : geometric_set;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.54 NONMAXIMAL AREA REPRESENTATION

The symbolic description of a nonmaximal area shape element in a geometric model.

```
*)
ENTITY nonmaximal_area_representation
  SUPERTYPE OF (brep_nm_area_rep XOR
                facetted_brep_nm_area_rep XOR
                wireframe_nm_area_rep XOR
                surface_nm_area_rep XOR
                partial_rev_sor_subface_nm_area_rep XOR
                sole_subface_nm_area_rep XOR
                sor_edge_nm_area_rep XOR
                sole_edge_nm_area_rep XOR
                geometric_set_nm_area_rep);
END_ENTITY;
(*
```

4.9.2.55 BREP NM AREA REP

The use of a subface in a manifold solid brep to represent a nonmaximal area shape element.

```
*)
ENTITY brep_nm_area_rep
  SUBTYPE OF (nonmaximal_area_representation);
  definition : subface;
  context    : manifold_solid_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.56 FACETTED BREP NM AREA REP

The use of a subface in a facetted brep to represent a nonmaximal area shape element.

```
*)
ENTITY facetted_brep_nm_area_rep
  SUBTYPE OF (nonmaximal_area_representation);
  definition : subface;
  context    : facetted_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.57 SURFACE NM AREA REP

The use of a subface in a surface model to represent a nonmaximal area shape element.

```
*)
ENTITY surface_nm_area_rep
  SUBTYPE OF (nonmaximal_area_representation);
  definition : subface;
  context    : surface_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.58 PARTIAL REV SOR SUBFACE NM AREA REP

The use of a subface in a partially revolved solid of revolution to represent a nonmaximal area shape element.

```
*)
ENTITY partial_rev_sor_subface_nm_area_rep
  SUBTYPE OF (nonmaximal_area_representation);
  definition : subface;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.59 SOLE SUBFACE NM AREA REP

The use of a subface in a solid of linear extrusion to represent a nonmaximal area shape element.

```
*)
ENTITY sole_subface_nm_area_rep
  SUBTYPE OF (nonmaximal_area_representation);
  definition : subface;
  context    : solid_of_linear_extrusion;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.60 WIREFRAME NM AREA REP

The use of a loop in a wireframe model to represent a nonmaximal area shape element.

```
*)
ENTITY wireframe_nm_area_rep
  SUBTYPE OF (nonmaximal_area_representation);
  definition : loop;
  context    : wireframe_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.61 SOR EDGE NM AREA REP

The use of an edge in a solid of revolution to represent a nonmaximal area shape element.

```
*)
ENTITY sor_edge_nm_area_rep
  SUBTYPE OF (nonmaximal_area_representation);
  definition : edge;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.62 SOLE EDGE NM AREA REP

The use of an edge in a solid of linear extrusion to represent a nonmaximal area shape element.

```
*)
ENTITY sole_edge_nm_area_rep
  SUBTYPE OF (nonmaximal_area_representation);
  definition : edge;
  context    : solid_of_linear_extrusion;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.63 GEOMETRIC SET NM AREA REP

The use of a geometry in a geometric set to represent a nonmaximal area shape element.

```
*)
ENTITY geometric_set_nm_area_rep
  SUBTYPE OF (nonmaximal_area_representation);
  definition : geometry;
  context    : geometric_set;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.64 SEAM REPRESENTATION

The symbolic description of a seam shape element in a geometric model.

```
*)
ENTITY seam_representation
  SUPERTYPE OF (brep_seam_rep XOR
                surface_seam_rep XOR
                wireframe_seam_rep XOR
                partial_rev_sor_edge_seam_rep XOR
                sole_edge_seam_rep XOR
                sor_vertex_seam_rep XOR
                sole_vertex_seam_rep XOR
                geometric_set_seam_rep);
END_ENTITY;
(*
```

4.9.2.65 BREP SEAM REP

The use of an edge in a manifold solid brep to represent a seam shape element.

```
*)
ENTITY brep_seam_rep
  SUBTYPE OF (seam_representation);
  definition : edge;
  context    : manifold_solid_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.66 SURFACE SEAM REP

The use of an edge in a surface model to represent a seam shape element.

```
*)
ENTITY surface_seam_rep
  SUBTYPE OF (seam_representation);
  definition : edge;
  context    : surface_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.67 WIREFRAME SEAM REP

The use of an edge in a wireframe model to represent a seam shape element.

```
*)
ENTITY wireframe_seam_rep
  SUBTYPE OF (seam_representation);
  definition : edge;
  context    : wireframe_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.68 PARTIAL REV SOR EDGE SEAM REP

The use of an edge in a partially revolved solid of revolution to represent a seam shape element.

```
*)
ENTITY partial_rev_sor_edge_seam_rep
  SUBTYPE OF (seam_representation);
  definition : edge;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.69 SOLE EDGE SEAM REP

The use of an edge in a solid of linear extrusion to represent a seam shape element.

```
*)
ENTITY sole_edge_seam_rep
  SUBTYPE OF (seam_representation);
  definition : edge;
  context    : solid_of_linear_extrusion;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.70 SOR VERTEX SEAM REP

The use of a vertex in a solid of revolution to represent a seam shape element.

```
*)
ENTITY sor_vertex_seam_rep
  SUBTYPE OF (seam_representation);
  definition : vertex;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.71 SOLE VERTEX SEAM REP

The use of a vertex in a solid of linear extrusion to represent a seam shape element.

```
*)
ENTITY sole_vertex_seam_rep
  SUBTYPE OF (seam_representation);
  definition : vertex;
  context    : solid_of_linear_extrusion;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.72 GEOMETRIC SET SEAM REP

The use of a geometry in a geometric set to represent a seam shape element.

```
*)
ENTITY geometric_set_seam_rep
  SUBTYPE OF (seam_representation);
  definition : geometry;
  context    : geometric_set;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.73 PERIMETER REPRESENTATION

The symbolic description of a perimeter shape element in a geometric model.

```
*)
ENTITY perimeter_representation
  SUPERTYPE OF (brep_perimeter_rep XOR
                facetted_brep_perimeter_rep XOR
                surface_perimeter_rep XOR
                wireframe_perimeter_rep XOR
                sole_edge_perimeter_rep XOR
                sole_loop_perimeter_rep XOR
                partial_rev_sor_edge_perimeter_rep XOR
                partial_rev_sor_loop_perimeter_rep XOR
                full_rev_sor_vertex_perimeter_rep XOR
                geometric_set_perimeter_rep);
END_ENTITY;
(*
```

4.9.2.74 BREP PERIMETER REP

The use of a loop in a manifold solid brep to represent a perimeter shape element.

```
*)
ENTITY brep_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : loop;
  context    : manifold_solid_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.75 SURFACE PERIMETER REP

The use of a loop in a surface model to represent a perimeter shape element.

```
*)
ENTITY surface_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : loop;
  context    : surface_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.76 WIREFRAME PERIMETER REP

The use of a loop in a wireframe model to represent a perimeter shape element.

```
*)
ENTITY wireframe_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : loop;
  context    : wireframe_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.77 PARTIAL REV SOR LOOP PERIMETER REP

The use of a loop in a partially revolved solid of revolution to represent a perimeter shape element.

```
*)
ENTITY partial_rev_sor_loop_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : loop;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.78 SOLE LOOP PERIMETER REP

The use of a loop in a solid of linear extrusion to represent a perimeter shape element.

```
*)
ENTITY sole_loop_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : loop;
  context    : solid_of_linear_extrusion;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.79 FACETTED BREP PERIMETER REP

The use of a poly loop in a facetted brep to represent a perimeter shape element.

```
*)
ENTITY facetted_brep_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : poly_loop;
  context    : facetted_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.80 FULL REV SOR VERTEX PERIMETER REP

The use of a vertex in a fully revolved solid of revolution to represent a perimeter shape element.

```
*)
ENTITY full_rev_sor_vertex_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : vertex;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.81 PARTIAL REV SOR EDGE PERIMETER REP

The use of an edge in a partial solid of revolution to represent a perimeter shape element.

```
*)
ENTITY partial_rev_sor_edge_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : edge;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.82 SOLE EDGE PERIMETER REP

The use of an edge in a solid of linear extrusion to represent a perimeter shape element.

```
*)
ENTITY sole_edge_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : edge;
  context    : solid_of_linear_extrusion;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.83 GEOMETRIC SET PERIMETER REP

The use of a geometry in a geometric set to represent a perimeter shape element.

```
*)
ENTITY geometric_set_perimeter_rep
  SUBTYPE OF (perimeter_representation);
  definition : geometry;
  context    : geometric_set;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.84 DIM 0 SHAPE ELEMENT REPRESENTATION

The symbolic description of a **dimensionality 0 shape element** in a **geometric model.**

```
*)
ENTITY dim_0_shape_element_representation
  SUPERTYPE OF (brep_dim_0_se_rep XOR
                surface_dim_0_se_rep XOR
                wireframe_dim_0_se_rep XOR
                sole_dim_0_se_rep XOR
                facetted_brep_dim_0_se_rep XOR
                geometric_set_dim_0_se_rep);
END_ENTITY;
(*
```

4.9.2.85 BREP DIM 0 SE REP

The use of a **vertex** in a **manifold solid brep** to represent a **dimensionality 0 shape element.**

```
*)
ENTITY brep_dim_0_se_rep
  SUBTYPE OF (dim_0_shape_element_representation);
  definition : vertex;
  context    : manifold_solid_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.86 SURFACE DIM 0 SE REP

The use of a **vertex** in a **surface model** to represent a **dimensionality 0 shape element.**

```
*)
ENTITY surface_dim_0_se_rep
  SUBTYPE OF (dim_0_shape_element_representation);
  definition : vertex;
  context    : surface_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.87 WIREFRAME DIM 0 SE REP

The use of a vertex in a wireframe model to represent a dimensionality 0 shape element.

```
*)
ENTITY wireframe_dim_0_se_rep
  SUBTYPE OF (dim_0_shape_element_representation);
  definition : vertex;
  context    : wireframe_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.88 SOLE DIM 0 SE REP

The use of a vertex in a solid of linear extrusion to represent a dimensionality 0 shape element.

```
*)
ENTITY sole_dim_0_se_rep
  SUBTYPE OF (dim_0_shape_element_representation);
  definition : vertex;
  context    : solid_of_linear_extrusion;
  face       : sole_face_types;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

face: Whether the starting or ending face is the one to consider.

4.9.2.89 FACETTED BREP DIM 0 SE REP

The use of a point in a facetted brep to represent a dimensionality 0 shape element.

```
*)
ENTITY facetted_brep_dim_0_se_rep
SUBTYPE OF (dim_0_shape_element_representation);
    definition : point;
    context    : facetted_brep;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.90 GEOMETRIC SET DIM 0 SE REP

The use of a geometry in a geometric set to represent a dimensionality 0 shape element.

```
*)
ENTITY geometric_set_dim_0_se_rep
  SUBTYPE OF (dim_0_shape_element_representation);
  definition : geometry;
  context    : geometric_set;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.91 CORNER REPRESENTATION

The symbolic description of a corner shape element in a geometric model.

```
*)
ENTITY corner_representation
  SUPERTYPE OF (partial_rev_sor_corner_rep OR
                full_rev_sor_corner_rep);
END_ENTITY;
(*
```

4.9.2.92 FULL REV SOR CORNER REP

The use of a vertex in a fully revolved solid of revolution to represent a corner shape element.

```
*)
ENTITY full_rev_sor_corner_rep
  SUBTYPE OF (corner_representation);
  definition : vertex;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.93 PARTIAL REV SOR CORNER REP

The use of a vertex in a partial solid of revolution to represent a corner shape element.

```
*)
ENTITY partial_rev_sor_corner_rep
  SUBTYPE OF (corner_representation);
  definition : vertex;
  context    : solid_of_revolution;
  face       : partial_rev_sor_face_types;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

face: Whether the starting or ending face is the one to consider.

4.9.2.94 PARTIAL REV SOR BDRY LOC REP

The use of a vertex in a partial solid of revolution to represent a boundary location shape element.

```
*)
ENTITY partial_rev_sor_bdry_loc_rep;
  definition : vertex;
  context    : solid_of_revolution;
  face       : partial_rev_sor_face_types;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

face: Whether the starting or ending face is the one to consider.

4.9.2.95 INTERIOR LOCATION REPRESENTATION

The symbolic description of an interior location shape element in a geometric model.

```
*)
ENTITY interior_location_representation
  SUPERTYPE OF (partial_rev_sor_int_loc_rep XOR
                full_rev_sor_int_loc_rep);
END_ENTITY;
(*
```

4.9.2.96 FULL REV SOR INT LOC REP

The use of a vertex in a fully revolved solid of revolution to represent an interior location shape element.

```
*)
ENTITY full_rev_sor_int_loc_rep
  SUBTYPE OF (interior_location_representation);
  definition : vertex;
  context    : solid_of_revolution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

4.9.2.97 PARTIAL REV SOR INT LOC REP

The use of a vertex in a partial solid of revolution to represent an interior location shape element.

```
*)
ENTITY partial_rev_sor_int_loc_rep
  SUBTYPE OF (interior_location_representation);
  definition : vertex;
  context    : solid_of_revolution;
  face       : partial_rev_sor_face_types;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the defining modeling information for this representation.

context: The reference to the particular geometric model that the definition belongs to.

face: Whether the starting or ending face is the one to consider.

4.9.3 Integration Core IPIM Classification Structure

The following indented listing shows the classification structure for the Integration Core model.

```
AREA REPRESENTATION
  BREP AREA REP
  FACETTED BREP AREA REP
  GEOMETRIC SET AREA REP
  SOLE EDGE AREA REP
  SOR EDGE AREA REP
  SURFACE AREA REP
```

WIREFRAME AREA REP
CORNER REPRESENTATION
FULL REV SOR CORNER REP
PARTIAL REV SOR CORNER REP
DIM 0 SHAPE ELEMENT REPRESENTATION
BREP DIM 0 SE REP
FACETTED BREP DIM 0 SE REP
GEOMETRIC SET DIM 0 SE REP
SOLE DIM 0 SE REP
SURFACE DIM 0 SE REP
WIREFRAME DIM 0 SE REP
DIM 3 SHAPE ELEMENT REPRESENTATION
CSG SOLID DIM 3 SE REP
GEOMETRIC SET DIM 3 SE REP
SURFACE DIM 3 SE REP
WIREFRAME DIM 3 SE REP
INTERIOR LOCATION REPRESENTATION
FULL REV SOR INT LOC REP
PARTIAL REV SOR INT LOC REP
MAXIMAL AREA REPRESENTATION
PARTIAL REV SOR MAX AREA REP
SOLE MAX AREA REP
NONMAXIMAL AREA REPRESENTATION
BREP NM AREA REP
FACETTED BREP NM AREA REP
GEOMETRIC SET NM AREA REP
PARTIAL REF SOR SUBFACE NM AREA REP
SOLE EDGE NM AREA REP
SOLE SUBFACE NM AREA REP
SOR EDGE NM AREA REP
SURFACE NM AREA REP
WIREFRAME NM AREA REP
OBJECT ASSEMBLY REPRESENTATION
OBJECT REPRESENTATION
BREP OBJECT REP
FACETTED BREP OBJECT REP
FULL REV SOR OBJECT REP
PAIR OF EQUIVALENT GEOMETRIC MODELS
PARTIAL REV SOR BDRY LOC REP
PERIMETER REPRESENTATION
BREP PERIMETER REP
FACETTED BREP PERIMETER REP
FULL REV SOR VERTEX PERIMETER REP
GEOMETRIC SET PERIMETER REP
PARTIAL REV SOR EDGE PERIMETER REP
PARTIAL REV SOR LOOP PERIMETER REP
SOLE EDGE PERIMETER REP
SOLE LOOP PERIMETER REP
SURFACE PERIMETER REP

```
    WIREFRAME PERIMETER REP
  SEAM REPRESENTATION
    BREP SEAM REP
    GEOMETRIC SET SEAM REP
    PARTIAL REV SOR EDGE SEAM REP
    SOLE EDGE SEAM REP
    SOLE VERTEX SEAM REP
    SOR VERTEX SEAM REP
    SURFACE SEAM REP
    WIREFRAME SEAM REP
  SHAPE
  SHAPE ELEMENT
    DIMENSIONALITY 0 SHAPE ELEMENT
      BOUNDARY LOCATION SHAPE ELEMENT
      CORNER SHAPE ELEMENT
      INTERIOR LOCATION SHAPE ELEMENT
    DIMENSIONALITY 1 SHAPE ELEMENT
      PERIMETER SHAPE ELEMENT
      SEAM SHAPE ELEMENT
        EDGE SHAPE ELEMENT
        INTERIOR SEAM SHAPE ELEMENT
        SUBEDGE SHAPE ELEMENT
    DIMENSIONALITY 2 SHAPE ELEMENT
      AREA SHAPE ELEMENT
        MAXIMAL AREA SHAPE ELEMENT
        NONMAXIMAL AREA SHAPE ELEMENT
      ZONE SHAPE ELEMENT
    DIMENSIONALITY 3 SHAPE ELEMENT
      OBJECT ASSEMBLY SHAPE ELEMENT
      OBJECT SHAPE ELEMENT
  VOIDLESS VOLUME REPRESENTATION
    BREP VOLUME REP
    CSG PRIMITIVE VOLUME REP
    FACETTED BREP VOLUME REP
    FULL REV SOR VOLUME REP
    HALF SPACE VOLUME REP
    PARTIAL REV SOR VOLUME REP
    SOLE VOLUME REP
  VOIDLESS VOLUME SHAPE ELEMENT
  ZONE SHAPE ELEMENT COMPONENT

  *)
  END_SCHEMA; -- ipim_shape_interface_schema
  (*
```

4.10 Tolerancing

4.10.1 Introduction

```
*)
SCHEMA ipim_tolerances_schema;

  EXPORT EVERYTHING;
  ASSUME(ipim_shape_interface_schema,
         ipim_features_schema,
         ipim_geometry_schema);
(*
```

Tolerances define allowable deviations from an exact condition. There are a variety of tolerances, which can apply to the size of an object, roughness of a surface, temperature of a process, voltage, torque, and other measurable conditions. This specification presently addresses the problem of tolerancing as it applies to the size and shape of an object.

4.10.1.1 Assumptions

- Product models are assumed to be complete, unambiguous, three dimensional definitions of the shape of a desired object, typically represented by a Boundary Representation (Brep) Solid model or an equivalent bounded, surfaced Wireframe (Geometric) Model. The general requirement is the capability to reference shape defining elements of the surface of a modeled product. The terms *Area*, *Seam* and *Corner* are used within this model to refer to the appropriate Shape Element (and correspond roughly to the Brep notion of Face, Edge and Vertex).
- Geometric elements are assumed to underly the topological elements of Face, Edge and Vertex.
- Product models contain exact definitions of nominal product geometry, i.e the model is considered BASIC.
- The value of the dimension is implicit in model geometry or is an explicit parameter of an Implicit Feature. Each Tolerance specified within a Model implies an underlying dimension.
- Graphical display of tolerance information will conform to ANSI Y14.5M 1982, specifically the Feature Control Frame.
- The tolerances described in this model include both Coordinate and Geometric Tolerances. Both types of tolerance are described as they pertain to common tolerancing practices which are used to define the form, fit and function of the modeled product.
- Independent or dependent geometric constructs may be needed in addition to shape defining elements to specify certain types of tolerances. Examples of these are hole centerlines, off-part datums or other derived geometries. Therefore, it is assumed that these constructs are present or available as part of the product definition model for reference in tolerances.
- All tolerance dimensions are derived from shape/size elements or explicitly stated as part of Implicit Features.

Note on Model Content This version of the Tolerance Schema expands on two concepts which wqere only cursorily addressed in earlier versions: *dimensions* and *geometric derivations*.

Coordinate tolerances, as defined in previous versions of the model, contained "extra" data that was intended to facilitate the determination of the value of the intended dimension. This version of the model flips the emphasis from the tolerance to the dimension: coordinate dimensions are defined and the tolerance values are the "extra" data.

Geometric derivations are defined to accommodate typical dimension practices which reference geometry that is not part of the shape definition of the object but is derivable from shape elements. The most common examples are centerlines of holes and the spatial intersection of two part surfaces.

4.10.1.2 Tolerance Model TYPES

4.10.1.2.1 AREA OR FOS

```
*)
TYPE area_or_fos =  SELECT
     (area_shape_element,
      feature_of_size);
END_TYPE;
(*
```

4.10.1.2.2 AREA OR SEAM

```
*)
TYPE area_or_seam = SELECT
     (area_shape_element,
      seam_shape_element);
END_TYPE;
(*
```

4.10.1.2.3 AREA OR SEAM OR FOS

```
*)
TYPE area_or_seam_or_fos = SELECT
     (area_shape_element,
      seam_shape_element,
      feature_of_size);
END_TYPE;
(*
```

4.10.1.2.4 DIMENSION MEASUREMENT

Dimensions are sometimes specified across the full "width" of a feature (e.g a cone angle or the diameter of a cylinder) and at other times across the "half width" (e.g a cone semi-angle or the radius of a cylinder or circular blend).
This TYPE specifies whether the dimension is **full** or **half**.

```
*)
TYPE dimension_measurement = ENUMERATION OF
     (full,
```

```
      half);
END_TYPE;
(*
```

4.10.1.2.5 DT FEATURE

A **dt feature** (dimension/tolerance feature) is a categorization of things which may be the target or origin of dimension / tolerance and corresponds to the term "feature" as used in the ANSI Y14.5 Standard. It is some characteristic of the shape of a product that can be uniquely identified. Categories of a **dt feature** include:

- **shape element** (Area, Seam and Corner)
- **Form feature**
- **Feature of size** (a subset of which are **form feature**)
- **Geometric derivation.**

Angle and location dimensions reference **dt features** and **size dimension** reference **feature of size.** It is an "artificial" classification defined for the purposes of the tolerance model.

```
*)
TYPE dt_feature = SELECT
      (dt_shape_element,
       form_feature,
       feature_of_size,
       geometric_derivation);
END_TYPE;
(*
```

4.10.1.2.6 DT SHAPE ELEMENT

Shape elements or entities, within the scope of the Tolerance schema, are constructs that define touchable or viewable portions of a part. Shape elements are defined in various ways depending on the type of shape representation method used. A **dt shape element** is a subset of **shape element** defined for the purpose of tolerancing.

```
*)
TYPE dt_shape_element = SELECT
      (area_shape_element,
       seam_shape_element,
       corner_shape_element);
END_TYPE;
(*
```

4.10.1.2.7 SHAPE OR DERIVED

```
*)
TYPE shape_or_derived = SELECT
      (dt_shape_element,
```

```
      geometric_derivation);
END_TYPE;
(*
```

4.10.1.2.8 TOL IBO

The tolerance zone for profile tolerances (**profile line**, **profile surface**) may be bilaterally disposed about the toleranced feature, or may be specified as being entirely inside or outside of the shape of an object.

```
*)
TYPE tol_ibo  = ENUMERATION OF
      (inside,
       bilateral,
       outside);
END_TYPE;
(*
```

4.10.1.2.9 TOL MLSN

The material condition modifier is a flag which affects the size of the tolerance zone based on the measured size of the toleranced feature. It is meaningful only to features subject to variation in size.

```
*)
TYPE tol_mlsn = ENUMERATION OF
      (maxmc,
       leastmc,
       regardless,
       none);
END_TYPE;
(*
```

where **maxmc** = maximum material condition, **leastmc** = least material condition, **regardless** = regardless of feature size, **none** = not applicable.

4.10.1.3 TOLERANCE

```
*)
ENTITY tolerance
  SUPERTYPE OF (shape_tolerance);
END_ENTITY;
(*
```

4.10.1.4 TOLERANCE RANGE

The allowable upper bound deviation and lower bound deviation which define a **tolerance**.

```
*)
ENTITY tolerance_range
```

```
  SUPERTYPE OF (coordinate_tolerance_range XOR
                NULL);
  plus_tol  : REAL;
  minus_tol : REAL;
WHERE
  plus_tol >= 0;
  minus_tol >= 0;
  plus_tol + minus_tol > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

plus_tol: The absolute value of the tolerance that is added to the nominal value to establish the maximum allowable value.

minus_tol: The absolute value of the tolerance that is subtracted from the nominal value to establish the minimum allowable value.

PROPOSITIONS:

1. Plus tol must be greater than or equal to zero
2. Minus tol must be greater than or equal to zero
3. The two tolerance values must not both be zero

4.10.1.5 TOLERANCE MAGNITUDE

A non-zero, positive, real value that defines the size of a tolerance zone.

```
*)
ENTITY tolerance_magnitude;
  tol_mag : REAL;
WHERE
  tol_mag > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

tol_mag: The absolute magnitude of a tolerance.

PROPOSITIONS:

1. The magnitude must be greater than zero.

4.10.1.6 TOLERANCED REAL

The association of a tolerance range with a real value.

```
*)
ENTITY toleranced_real;
  real_value : REAL;
  range      : tolerance_range;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

real_value: A REAL number.

range: The allowable variation on the value of real value.

PROPOSITIONS:

4.10.2 SHAPE TOLERANCE

The allowable deviation of a measurable characteristic of the shape of a product from its design nominal geometry.

```
*)
ENTITY shape_tolerance
  SUPERTYPE OF (coordinate_tolerance_range XOR
                geometric_tolerance)
  SUBTYPE OF (tolerance);
END_ENTITY;
(*
```

4.10.2.1 COORDINATE TOLERANCE RANGE

A coordinate tolerance range is the numeric values added to and subtracted from the nominal dimensional value calculated from the shape definition of a product. Coordinate tolerance range represents the traditional plus minus tolerances found on dimensions on drawings.

```
*)
ENTITY coordinate_tolerance_range
  SUBTYPE OF (tolerance_range,
              shape_tolerance);
END_ENTITY;
(*
```

4.10.2.2 GEOMETRIC TOLERANCE

Geometric tolerances, as defined by the ANSI Y14.5M 1982 standard, tolerance the deviation of form, orientation, location, and runout of a produced feature from its design nominal.

```
*)
ENTITY geometric_tolerance
  SUPERTYPE OF (angularity XOR
                circular_runout XOR
```

```
                - circularity XOR
                  concentricity XOR
                  cylindricity XOR
                  flatness XOR
                  parallelism XOR
                  perpendicularity XOR
                  position XOR
                  profile_line XOR
                  profile_surface XOR
                  straightness XOR
                  total_runout)
  SUBTYPE OF (shape_tolerance);
END_ENTITY;
(*
```

4.10.2.3 ANGULARITY TOLERANCE

Angularity is the condition of a surface or axis at a specified angle (other than 90 degrees) from a datum plane or axis. An angularity tolerance specifies a tolerance zone defined by two parallel planes at the specified basic angle from the datum plane or axis within which the surface or axis of the considered feature must lie. See *ANSI Y14.5M 1982, page 106, section 6.6.2* and see *ISO 1101, pages 18-19, section 5.9.*

```
*)
ENTITY angularity
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents     : SET [1 : ?] OF area_or_seam_or_fos;
  magnitude           : tolerance_magnitude;
  material_condition  : tol_misn;
  projection          : optional projected_tolerance_zone;
  primary_datum       : conditioned_datum;
  secondary_datum     : optional conditioned_datum;
  tertiary_datum      : optional conditioned_datum;
WHERE
  secondary_datum <> primary_datum;
  tertiary_datum <> primary_datum;
  tertiary_datum <> secondary_datum;
  valid_misn(material_condition, toleranced_ents);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of the tolerance

material_condition: An enumerated list that indicates the material condition at which the tolerance applies: maximum material condition; least material condition; regardless of feature size; or Not Applicable.

projection: A **projected tolerance zone** which specifies the additional height and direction of the projected tolerance zone outside the feature boundary. Note: Must be parallel to the toleranced features. Length of vector determines the extent of the zone.

primary_datum: A **conditioned datum** from which the dimension is measured. This datum is the most important datum relative to the tolerance.

secondary_datum: A **conditioned datum** which is the second most important datum relative to the tolerance.

tertiary_datum: A **conditioned datum** which is the third most important datum relative to the tolerance With the primary and secondary datums, it establishes a datum reference frame that exactly locates the toleranced feature.

PROPOSITIONS:

1. May have a projected tolerance zone defined by one **projected tolerance zone.**
2. May have a secondary datum of one **conditioned datum.**
3. May have a tertiary datum of one **conditioned datum.**
4. The datums must be distinct.
5. The **material condition** must be appropriate for the type of entity(s) toleranced.

4.10.2.4 CIRCULAR RUNOUT TOLERANCE

Circular runout tolerance defines the maximum allowable deviation of position of a point on the toleranced feature during one complete revolution of the feature about the datum axis, without relative axial displacement of the measuring position. Where applied to surfaces constructed around a datum axis, it is used to to control the cumulative variations of circularity and coaxiality. Where applied to surfaces constructed at right angles to the datum axis, circular runout controls circular elements of a planar surface. See *ANSI Y14.5M 1982, page 109, section 6.7.2.1* and see *ISO 1101, page 22-23, section 5.12.*

```
*)
ENTITY circular_runout
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents    : SET [1 : ?] OF area_or_fos;
  magnitude          : tolerance_magnitude;
  primary_datum      : unconditioned_datum;
  co_primary_datum   : OPTIONAL unconditioned_datum;
WHERE
  co_primary_datum <> primary_datum;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of the tolerance.

primary_datum: A datum from which the dimension is measured. The primary datum is the most important datum relative to the tolerance.

co_primary_datum: An additional datum which establishes an axis with the primary datum.

```
*)
RULE circular_runout_rule FOR (circular_runout);
  LOCAL
    i : INTEGER;
  END_LOCAL;
   REPEAT i := 1 to SIZEOF(toleranced_ents);
     IF NOT (is_circular(toleranced_ents[i]) OR
             is_disk(toleranced_ents[i])) THEN
       VIOLATION;
     END_IF;
   END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. May have a co-datum of one unconditioned datum.
2. The datums must be distinct.
3. Each toleranced entity must be axisymmetric

4.10.2.5 CIRCULARITY TOLERANCE

Circularity is a condition of a surface of revolution such that all points of the surface intersected by a plane perpendicular to the axis or center point are equidistant from the intersection point of the plane and the axis or center point. A circularity tolerance defines the distance between two concentric circles within which the coordinates of the toleranced feature must lie. These circles are perpendicular to and centered on the axis of the toleranced feature. See ANSI Y14.5M 1982, page 95, section 6.4.3 See ISO 1101, page 14, section 5.3

```
*)
ENTITY circularity
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents : SET [1 : ?] OF area_or_seam_or_fos;
  magnitude       : tolerance_magnitude;
WHERE
  is_circular(toleranced_ents);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of the tolerance.

PROPOSITIONS:

1. The toleranced entities must be circular

4.10.2.6 CONCENTRICITY TOLERANCE

Concentricity tolerance specifies the diameter of cylinder centered on a datum point or axis within which the center or axis of the toleranced circular or cylindrical feature must lie. The specified tolerance and datum reference apply only on a Regardless-of-Feature-Size basis. See *ANSI Y14.5M 1982, page 84, section 5.11.3* and see *ISO 1101, page 21, section 5.11.1.*

```
*)
ENTITY concentricity
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents   : SET [1 : ?] OF area_or_fos;
  magnitude         : tolerance_magnitude;
  cylindrical_zone  : LOGICAL;
  primary_datum     : unconditioned_datum;
  co_primary_datum  : OPTIONAL unconditioned_datum;
WHERE
  co_primary_datum <> primary_datum;
  is_circular(toleranced_ents);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of the tolerance.

cylindrical_zone: A boolean (true/false) flag that indicates that the tolerance value is the diameter of a cylindrical zone within which the axis or line must lie. If false, the zone is parallelepipedic or the space between two parallel lines or planes.

primary_datum: A datum from which the dimension is measured. The primary datum is the most important datum relative to the tolerance.

co_primary_datum: An additional datum which establishes an axis with the primary datum.

PROPOSITIONS:

1. May have a datum of one **unconditioned datum**.
2. The datums must be distinct.
3. Each toleranced entity must be circular.

4.10.2.7 CYLINDRICITY TOLERANCE

Cylindricity tolerance defines the distance between two coaxial cylinders within which the toleranced cylindrical feature must lie. Unlike **circularity**, the tolerance applies simultaneously to both circular and longitudinal elements of the surface. See *ANSI Y14.5M 1982, page 96, section 6.4.4.* and see *ISO 1101, page 14, section 5.4.*

```
*)
ENTITY cylindricity
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents : SET [1 : #] OF area_or_fos;
  magnitude       : tolerance_magnitude;
WHERE
  is_cylindrical(toleranced_ents);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of tolerance.

PROPOSITIONS:

1. Each toleranced entity must be cylindrical.

4.10.2.8 FLATNESS TOLERANCE

Flatness tolerance defines the distance between two parallel planes between which the toleranced surface must lie. The tolerance may be applied on a unit basis as a means of preventing abrupt surface variation within a small area of the feature. See *ANSI Y14.5M 1982, page 94, section 6.4.2* and see *ISO 1101, page 13, section 5.2.*

```
*)
ENTITY flatness
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents : SET [1 : #] OF area_shape_element;
  magnitude       : tolerance_magnitude;
  per_unit_square : OPTIONAL REAL;
WHERE
  per_unit_square > 0.0;
  planar(toleranced_ents);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of the tolerance.

per_unit_square: Specifies the side of square region on the toleranced-entity over which the tolerance value applies. Used to prevent abrupt surface variations in a relatively small area of the feature.

PROPOSITIONS:

1. Per unit square must be greater than zero
2. Each toleranced entity must be planar

4.10.2.9 PARALLELISM TOLERANCE

Parellelism is the condition of a surface equidistant at all points from a datum plane or an axis equidistant along the length from the datum axis. **Parallelism** tolerance specifies the distance between two planes or lines parallel to a datum plane or axis within which the line elements of the surface or axis of the considered feature must lie, or a cylindrical tolerance zone whose axis is parallel to the datum axis, within which the axis of the considered feature must lie. The allowable feature position zone may be cylindrical(fig 45), parallelepipedic (fig 51) or planar (fig 47, 54) for line features and is similar to a flatness tolerance zone for surface features (fig 57, 60). These figure references are to the ISO 1101 standard. See *ANSI Y14.5M 1982, page 106, section 6.6.3* and see *ISO 1101, page 15-17, section 5.7.*

```
*)
ENTITY parallelism
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents     : SET [1 : #] OF area_or_seam_or_fos;
  magnitude           : tolerance_magnitude;
  material_condition  : tol_mlsn;
  cylindrical_zone    : LOGICAL;
  projection          : OPTIONAL projected_tolerance_zone;
  primary_datum       : conditioned_datum;
WHERE
  valid_mlsn(material_condition, toleranced_ents);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of the tolerance.

material_condition: An enumerated list that indicates the material condition at which the tolerance applies: maximum material condition; least material condition; regardless of feature size; or Not applicable.

cylindrical_zone: A boolean (true/false) flag that indicates that the tolerance value is the diameter of a cylindrical zone within which the axis or line must lie. If false, the zone is parallelepipedic or the space between two parallel lines or planes.

projection: A **projected tolerance zone** which specifies the additional height and direction of the projected tolerance zone outside the feature boundary. Note: Must be parallel to the toleranced features.

primary_datum: A **conditioned datum** from which the dimension is measured.

PROPOSITIONS:

1. May have a projected tolerance zone defined by one **projected tolerance zone.**
2. The **material condition** must be appropriate for the type of entity toleranced.

4.10.2.10 PERPENDICULARITY

Perpendicularity tolerance defines the allowable linear deviation from a true right angle of line or surface features with respect to line or surface datums. Several interpretations of the exact tolerance zone are possible for line features with respect to surface datums. See figures 65,67 and 69 in ISO 1101,page 17. See *ANSI Y14.5M 1982, page 106, section 6.6.4* and see *ISO 1101, page 17-18, section 5.8.*

```
*)
ENTITY perpendicularity
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents     : SET [1 : ?] OF area_or_seam_or_fos;
  magnitude           : tolerance_magnitude;
  material_condition  : tol_mlsn;
  cylindrical_zone    : LOGICAL;
  projection          : OPTIONAL projected_tolerance_zone;
  primary_datum       : conditioned_datum;
  secondary_datum     : OPTIONAL conditioned_datum;
WHERE
  secondary_datum <> primary_datum;
  valid_mlsn(material_condition, toleranced_ents);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of the tolerance.

material_condition: An enumerated list that indicates the material condition at which the tolerance applies: maximum material condition; least material condition; regardless of feature size; or Not applicable.

cylindrical_zone: A boolean (true/false) flag that indicates that the tolerance value is the diameter of a cylindrical zone within which the axis or line must lie. If false, the zone is parallelepipedic or the space between two parallel lines or planes.

projection: A **projected tolerance zone** which specifies the additional height and direction of the projected tolerance zone outside the feature boundary. Note: Must be parallel to the toleranced features.

primary_datum: A **conditioned datum** from which the dimension is measured.

secondary_datum: A **conditioned datum** which is the second most important datum relative to the tolerance.

PROPOSITIONS:

1. May have a secondary datum of one **conditioned datum**.
2. The datums must be distinct.
3. May have a projected tolerance zone defined by one **projected tolerance zone**.
4. The **material condition** must be appropriate for the type of entity(s) toleranced.

4.10.2.11 POSITION

Position tolerance defines the allowable deviation of position of the center point, axis or center plane of a feature of size. A center point tolerance defines the diameter of a spherical or circular zone, an axis tolerance defines the measure of a cylindrical, parallelepipedic or planar zone and a center plane tolerance defines a zone specified by the distance between two bounding parallel planes. Each zone is considered to contain the true position of the toleranced feature. See *ANSI Y14.5M 1982, page 53-89, section 5.2* and see *ISO 1101, page 19-20, section 5.10.*

```
*)
ENTITY position
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents     : SET [1 : ?] OF feature_of_size;
  magnitude           : tolerance_magnitude;
  material_condition  : tol_misn;
  cylindrical_zone    : LOGICAL;
  projection          : OPTIONAL projected_tolerance_zone;
  primary_datum       : conditioned_datum;
  secondary_datum     : OPTIONAL conditioned_datum;
  tertiary_datum      : OPTIONAL conditioned_datum;
WHERE
  secondary_datum <> primary_datum;
  tertiary_datum <> primary_datum;
  tertiary_datum <> secondary_datum;
  valid_misn(material_condition, toleranced_ents);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of the tolerance.

material_condition: An enumerated list that indicates the material condition at which the tolerance applies: maximum material condition; least material condition; regardless of feature size; or Not applicable.

cylindrical_zone: A boolean (true/false) flag that indicates that the tolerance value is the diameter of a cylindrical zone within which the axis or line must lie. If false, the zone is parallelepipedic or the space between two parallel lines or planes.

projection: A projected tolerance zone which specifies the additional height and direction of the projected tolerance zone outside the feature boundary. Note: Must be parallel to the toleranced features.

primary_datum: A conditioned datum from which the dimension is measured.

secondary_datum: A conditioned datum which is the second most important datum relative to the tolerance.

tertiary_datum: A conditioned datum which is the third most important datum relative to the tolerance.

PROPOSITIONS:

1. May have a secondary datum of one **conditioned datum**.
2. May have a tertiary datum of one **conditioned datum**.
3. The datums must be distinct.
4. May have a projected tolerance zone defined by one **projected tolerance zone**.
5. The **material condition** must be appropriate for the type of entity(s) toleranced.

4.10.2.12 PROFILE LINE

Profile of a line tolerance specifies the diameter of a circle which when its center or one tangent moves along the design nominal curve feature, sweeps the region in which the feature must lie. The tolerance is two dimensional and applies normal (perpendicular) to the true profile at all points. Where a sharp corner is included, the tolerance zone extends to the intersection of the boundary lines. See *ANSI Y14.5M 1982, page 97-104, section 6.5.1* and see *ISO 1101, page 14, section 5.5*.

```
*)
ENTITY profile_line
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents : SET [1 : #] OF area_or_seam;
  magnitude       : tolerance_magnitude;
  directrix       : OPTIONAL direction;
  application     : tol_ibo;
  primary_datum   : OPTIONAL conditioned_datum;
  secondary_datum : OPTIONAL conditioned_datum;
  tertiary_datum  : OPTIONAL conditioned_datum;
WHERE
  secondary_datum <> primary_datum;
  tertiary_datum <> primary_datum;
  tertiary_datum <> secondary_datum;
  NOT ((toleranced_ents = seam) AND NOT planar(toleranced_ents));
  ((toleranced_ents = seam) AND (directrix = NULL) OR
   (toleranced_ents <> seam));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the the tolerance applies.

magnitude: Magnitude of the tolerance.

application: An enumerated list that specifies the tolerance application to be I - inside, B - bilateral O - Outside.

primary_datum: A **conditioned datum** from which the dimension is measured. This primary datum is the most important datum relative to the tolerance.

secondary_datum: A conditioned datum which is the second most important datum relative to the tolerance.

tertiary_datum: A conditioned datum which is the third most important datum relative to the tolerance. With the primary and secondary datums, it establishes a datum reference frame that exactly located the toleranced feature.

directrix: Cross sections of the toleranced face taken in planes normal to the directix establish the line profile that is toleranced. The directrix is required only if a datum is not included in the line tolerance entity.

PROPOSITIONS:

1. May have a primary datum of one **conditioned datum**.
2. May have a secondary datum of one **conditioned datum**.
3. May have a tertiary datum of one **conditioned datum**.
4. The datums must be distinct.
5. May have profile determined in planes normal to one **direction**.
6. If the entity to which the tolerance applies is a **seam**, then the base-curve to the **seam** must be defined within a plane and the **directrix** is meaningless in this case.
7. A value for the **directrix** may be assigned if there are entities in the list which are **areas**.

4.10.2.13 PROFILE SURFACE

A profile of a surface tolerance specifies the distance between two "ball-offset" surfaces located equally on either side, or totally on one side of the design nominal feature. The toleranced feature must lie between these surfaces. The tolerance is three-dimensional and applies normal (perpendicular) to the true profile at all points. Where a sharp corner is included, the tolerance zone extends to the intersection of the boundary. See *ANSI Y14.5M 1982, page 97-104, section 6.5.1* and see *ISO 1101, page 14, section 5.5*.

```
*)
ENTITY profile_surface
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents : SET [0 : ?] OF area_shape_element;
  magnitude       : REAL;
  default         : LOGICAL;
  application     : tol_ibo;
  primary_datum   : OPTIONAL conditioned_datum;
  secondary_datum : OPTIONAL conditioned_datum;
  tertiary_datum  : OPTIONAL conditioned_datum;
WHERE
  secondary_datum <> primary_datum;
  tertiary_datum <> primary_datum;
  tertiary_datum <> secondary_datum;
  magnitude >= 0;
  empty_set(toleranced_ents) = default;
```

```
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of tolerance.

default: A Boolean flag used to denote the instance of the **profile surface** tolerance that is used as the default tolerance within the Product Model. Only one instance of this entity may have this attribute set to TRUE, and this one may not reference any toleranced entity.

application: An enumerated list that specifies the tolerance application to be I - inside, B - bilateral O - Outside.

primary_datum: A **conditioned datum** from which the dimension is measured. This primary datum is the most important datum relative to the tolerance.

secondary_datum: A **conditioned datum** which is the second most important datum relative to the tolerance.

tertiary_datum: A **conditioned datum** which is the third most important datum relative to the tolerance. With the primary and secondary datums, it establishes a datum reference frame that exactly located the toleranced feature.

```
*)
RULE default_profile_surface FOR (profile_surface);
  LOCAL
    k : INTEGER;
  END_LOCAL;
  k := 0;
  REPEAT FOR EACH profile_surface IN MODEL;
    IF (profile_surface.default = TRUE) THEN
      k := k + 1;
    END_IF;
  END_REPEAT;
  IF (k > 1) THEN
    VIOLATION;
  END_IF;
END_RULE;
(*
```

PROPOSITIONS:

1. The magnitude must not be less than zero
2. May have a primary datum of one **conditioned datum**.
3. May have a secondary datum of one **conditioned datum**.
4. May have a tertiary datum of one **conditioned datum**.
5. The datums must be distinct.

6. If the Tol-Ent set is EMPTY, then the Default flag must be TRUE. In addition, the instance of the entity where this condition is true must be UNIQUE (i.e. there is only one entity which has a default flag set to TRUE.)

4.10.2.14 STRAIGHTNESS

Straightness tolerance defines the allowable deviation of a line or of line elements of a surface feature. The tolerance zones for line features are cylindrical, parallelepipedic and parallel planes. Straightness tolerance for surface features specify the distance between two parallel lines within which a linear element of the surface in a specified direction, must lie. The linear elements is a cross section of the surface in a plane parallel to the direction vector and normal to the surface. See figures 27,29 and 31 in the ISO 1101 standard, page 17. See *ANSI Y14.5M 1982, page 91-94, section 6.4.1* and see *ISO 1101, page 13, section 5.1.*

```
*)
ENTITY straightness
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents        : SET [1 : #] OF area_or_seam_or_fos;
  magnitude              : tolerance_magnitude;
  application_direction  : OPTIONAL direction;
  material_condition     : tol_mlsn;
  cylindrical_zone       : LOGICAL;
  per_unit_length        : REAL;
WHERE
  per_unit_length > 0.0;
  valid_mlsn(material_condition, toleranced_ents);
  (tol_ent = seam AND application_direction = NULL) OR
  (linear_sections(application_direction, tol_ent));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of tolerance.

application_direction: A direction that specifies the straightness tolerance for linear surface elements. Required when the toleranced entity is a surface. It must be parallel to the surface.

material_condition: An enumerated list that indicates the material condition at which the tolerance applies: maximum material condition; least material condition; regardless of feature size; or Not applicable.

cylindrical_zone: A boolean (true/false) flag that indicates that the tolerance value is the diameter of a cylindrical zone within which the axis or line must lie. If false, the zone is parallelepipedic or the space between two parallel lines or planes.

per_unit_length: Specifies the linear distance within which the tolerance value applies. Used to prevent abrupt changes in the direction of the toleranced entity.

PROPOSITIONS:

1. The **material condition** must be appropriate for the type of entity(s) toleranced.
2. If the set **toleranced ents** contains only **seams**, then **application direction** must be NULL.
3. **Application direction** does not apply to a **seam**.
4. May have a direction of application defined by one **direction**.
5. **Linear sections** is a function which returns a value TRUE if all cross sections of the toleranced surface (or members of a list of entities) taken in planes tangent to the direction are straight lines.

4.10.2.15 TOTAL RUNOUT

A **total runout** tolerance specifies the maximum allowable deviation of position for all points on the toleranced feature during one complete revolution of the feature about the datum axis, with relative axial displacement of the measuring position. Where applied to surfaces constructed around a datum axis, it is used to control the cumulative variations of circularity, straightness, angularity, taper, profile of surface, and coaxiality. Where applied to surfaces constructed at right angles to the datum axis, total runout controls perpendicularity and flatness. See *ANSI Y14.5M 1982, page 109, section 6.7.2.2* and see *ISO 1101, page 22-23, section 5.12*.

Note that ANSI and ISO do not explicitly define total runout for non-cylindrical surfaces. It is assumed that this is due to mechanical measurement limitations.

```
*)
ENTITY total_runout
  SUBTYPE OF (geometric_tolerance);
  toleranced_ents   : SET [1 : #] OF area_or_fos;
  magnitude         : tolerance_magnitude;
  primary_datum     : unconditioned_datum;
  co_primary_datum  : OPTIONAL unconditioned_datum;
WHERE
  co_primary_datum <> primary_datum;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

toleranced_ents: A set of entities to which the tolerance applies.

magnitude: Magnitude of tolerance.

primary_datum: A datum from which the dimension is measured. The primary datum is the most important datum relative to the tolerance.

co_primary_datum: An additional datum which establishes an axis with the primary datum.

```
*)
RULE total_runout_rule FOR (total_runout);
  LOCAL
    i : INTEGER;
  END_LOCAL;
```

```
  REPEAT i := 1 TO SIZEOF(toleranced_ents);
    IF NOT (is_circular(tol_ent[i]) or is_disk(tol_ent[i])) THEN
      VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. May have a co-datum of one **unconditioned datum**.
2. The datums must be distinct.
3. Each member in the **tol ent** set must be either Circular or a Disk.

4.10.2.16 DATUM

A theoretically exact geometric reference to which toleranced features are related.

```
*)
ENTITY datum
  SUPERTYPE OF (conditioned_datum XOR
                unconditioned_datum);
  reference : dt_feature;
  name      : STRING;
WHERE
    1 <= SIZEOF(name) <= 2;
    ( ('A' <= name  <= 'Z') OR ('AA' <= name <= 'ZZ') ) AND
         (((name[1] <> 'I' AND name[2] <> 'I'  ) AND
         (( name[1] <> 'O' AND name[2] <> 'O' )) AND
         (  name[1] <> 'Q' AND name[2] <> 'Q')));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

reference: An entity which serves as the exact definition of the datum.

name: An alphabetic string that provides a unique designation for the datum.

PROPOSITIONS:

1. **Name** must be one or two alphabetic characters.
2. **Name** must not include the letters I, O or Q.

4.10.2.17 CONDITIONED DATUM

A **conditioned datum** is a **datum** to which a material condition specification applies (i.e. is defined by a Feature of Size).

```
*)
ENTITY conditioned_datum
  SUBTYPE OF (datum);
  material_condition  : tol_mlsn;
WHERE
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

material_condition: An enumerated list that indicates the material condition at which the tolerance applies: maximum material condition; least material condition; regardless of feature size; or Not Applicable.

```
*)
RULE datum_condition FOR
     (conditioned_datum,
      geometric_tolerance);
  REPEAT FOR EACH conditioned_datum IN MODEL;
    IF NOT valid_mlsn(conditioned_datum.material_condition,
                      datum.reference) THEN
      VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. The material condition must be appropriate to the reference.

4.10.2.18 UNCONDITIONED DATUM

An unconditioned datum is a datum to which a material condition modifier does not apply.

```
*)
ENTITY unconditioned_datum
  SUBTYPE OF (datum);
END_ENTITY;
(*
```

4.10.2.19 FEATURE OF SIZE

A feature of size is a grouping of areas or seams which are characterized by a set of opposing areas or seams and a point or curve (axis) or surface of symmetry. It has two subsets:

1. all defined size features are features of size;

2. all form features which have been defined as features of size (e.g. hole, tab, slot).

```
*)
ENTITY feature_of_size
  SUPERTYPE OF (size_feature XOR
                form_feature_of_size);
END_ENTITY;
(*
```

4.10.2.20 FORM FEATURE OF SIZE

A **form feature of size** is a **form feature** which has been explicitly identified as a **feature of size**. Typical form features which are features of size include holes, slots and tabs.

```
*)
ENTITY form_feature_of_size
  SUBTYPE OF (feature_of_size);
  ffos : form_feature;
WHERE
  is_symmetric(ffos);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

ffos: A **form feature** which is a **feature of size**.

PROPOSITIONS:

1. The **form feature** must be symmetric.

4.10.2.21 SIZE FEATURE

A collection of **areas** or **seams** which has a tolerancable geometric location that is derived from physical feature geometry and is symmetrical about that geometric location (e.g. point, axis, curve or surface.) This entity exists because a complete set of Form Features has not been defined and the concept of **feature of size** is required for tolerancing. This construct will allow the application of tolerances to shape definitions which to not included **form features**.

NOTE: The number of **areas** and **seams** used to define a **size feature** depends on the particular implementation of the shape representation. In some, a single surface which wraps around to form a cylinder and joins itself at a seam would be an appropriate single component of a **size feature**; others use two or more faces to define a cylinder.

```
*)
ENTITY size_feature
  SUBTYPE OF (feature_of_size);
  sf_components  : LIST [1 : ?] OF area_or_seam;
WHERE
  is_symmetric(sf_components);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sf_components: The list of **areas** or **seams** forming the feature.

```
*)
RULE size_feature_components FOR (size_feature);
  LOCAL
    i : INTEGER;
  END_LOCAL;
  IF NOT ((FOR i := 1 TO SIZEOF(sf_components)
             sf_components[i] = area) OR
          (FOR i := 1 TO SIZEOF(sf_components)
             sf_components[i] = seam))
    THEN VIOLATION;
  END_IF;
END_RULE;
(*
```

PROPOSITIONS:

1. Must be symmetric.
2. All members of **sf components** must be either all **areas** or all **seams**.

4.10.2.22 GEOMETRIC DERIVATION

A **geometric derivation** is a geometric entity which is derived from shape representation elements. An actual entity corresponding to this may or may not be included in the shape model. The key distinction is that the **geometric derivation** is not a definitional element of the shape of an object, but is determined from shape elements or other **geometric derivations**. On an actual physical object the geometric derivation is virtual geometry that is calculated from reference to "hard points" on the object (surfaces, edges and corners which correspond to the areas, seams and locations in the object representation). Examples include: centerlines and centerplanes; breakout tangency planes; spatial intersection of two surfaces.

A **geometric derivation** is a feature which may be used as the target or the origin of a tolerance/dimension.

A **geometric derivation** consists primarily of the shape elements of other **geometric derivations** used to determine the derived geometry, and a type designation indicating the type of derivation. A reference to the derived geometry may be optionally included.

```
*)
ENTITY geometric_derivation;
  gd_type     : UNDEFINED;
  parameters  : LIST [1 : #] OF shape_or_derived;
  result      : OPTIONAL SET OF geometry;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

gd_type: A (to be) enumerated list of operations that result in specified types of geometry which are derived from the input parameters. Each operation would consist of an input list of parameters, the anticipated type of result, and an algorithmic description of the operation.

parameters: The inputs to the derivation operation. Most typically the parameters will be shape element entities from which the geometry is derived.

result: The evaluated results may be included for completeness. The result of the derivation operation, however, takes precedence over a predefined result that is given here.

4.10.2.23 PROJECTED TOLERANCE ZONE

A **projected tolerance zone** is a **direction** and a length value which establishes the extent of the projected tolerance zone.

```
*)
ENTITY projected_tolerance_zone;
  zone_direction : direction;
  zone_extent    : REAL;
WHERE
  zone_extent > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

zone_direction: The **direction** from the tolerance zone defined by the tolerance entity that the additional tolerance zone is projected. This direction must be parallel to the **curve** or **surface of symmetry** of the toleranced entity (if it is a **feature of size**) or the toleranced entity itself (if it is not a **feature of size**).

zone_extent: A real value that specifies the length of the projected tolerance zone in the direction of **zone direction**.

```
*)
RULE zone_and_tolerance FOR
                        (projected_tolerance_zone,
                         geometric_tolerance);
  IF NOT (is_parallel(projected_tolerance_zone.zone_direction,
                      geometric_tolerance.toleranced_ents)) THEN
    VIOLATION;
      (* The zone direction must be parallel to the
         center of symmetry of the
         toleranced entity *)
  END_IF;
END_RULE;
(*
```

PROPOSITIONS:

1. The **zone extent** must be greater than zero.
2. The zone direction must be parallel to the center of symmetry of the toleranced entity

4.10.3 Dimensions

4.10.3.1 Introduction

The analysis of dimensions brought out some very subtle aspects of the meaning of dimensions placed on engineering drawings. Ostensibly, the dimensions define the theoretically exact shape of the modeled object. However, further analysis reveals meaning embedded within the plane of the paper in which the dimension is drawn, within the witness lines and even in the shape of the dimension leaders. The aspect of the dimension with the least meaning is the dimensional value. All of these characteristics are added together with tolerance value(s) to communicate the allowable shape deviations of the physical object from the design model to the design analysts, machinists and inspectors who use them.

For example, one distinction that was implicitly made in earlier versions of the Tolerance Model and is more explicitly dealt with in these extensions is the difference between the thing which is toleranced versus the thing controlled by the tolerance. The plane of the paper of a drawing represents a series of section cuts (parallel to the plane of the paper) through the object representation and each dimension is placed within one of these section cuts. Witness lines or leader arrowheads indicate the lines or planar curves within a section cut that are the origin or target for the dimension. The toleranced thing is the line or a point on the planar curve, while the thing being controlled by the dimension/tolerance combination is the surface that intersects the plane of the section cut to form the line or curve. Another example is the location of a hole from a plane. The thing to which the tolerance applies is the centerline of the hole, but the thing being controlled by the tolerance is the surface of the hole.

4.10.3.2 COORDINATE DIMENSION

A **coordinate dimension** corresponds to dimensions typically found on engineering drawings. This is a classification of **angle dimension**, **location dimension** and **size dimension**.

```
*)
ENTITY coordinate_dimension
  SUPERTYPE OF (angle_dimension XOR
                location_dimension XOR
                size_dimension);
END_ENTITY;
(*
```

4.10.3.3 ANGLE DIMENSION

An **angle dimension** is the measure of the orientation of one feature with respect to another feature. The tolerance is the numeric range that constrains the value of the dimension.

An **angle dimension** is either directed in the same sense as a **location dimension**, or bi-directional in the same sense as a **size dimension** (in that it applies to a single feature). The value of the dimension is the angle between two straight shape representation elements or the thing derived from a shape representation element (planar areas, linear seams or elements which have a constant angular relationship to one another).

```
*)
ENTITY angle_dimension
  SUPERTYPE OF (directed_angle_dimension XOR
                bidirectional_angle_dimension XOR
```

```
                angle_parameter_dimension)
  SUBTYPE OF (coordinate_dimension);
  range : OPTIONAL coordinate_tolerance_range;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

range: The optional tolerance range on the dimension.

4.10.3.4 DIRECTED ANGLE DIMENSION

Each feature within a **directed angle dimension** consists of two characteristics:

1. the thing which is used as the origin for the dimension, and
2. the actual element which is being controlled (or is the controlling element for the dimension).

These characteristics may be embodied in the same shape element or may be separate elements in the case of complex features.
A **directed angle dimension** must consist of two additional pieces of information in order to determine the dimensional value:

1. an orientation vector (parallel to area features and normal to seam features) to establish a right hand rule for
2. a flag that indicates the sense (clockwise = TRUE) or (counter clockwise = FALSE) of measurement.

```
*)
ENTITY directed_angle_dimension
  SUBTYPE OF (angle_dimension);
  target           : dt_feature;
  origin           : dt_feature;
  clockwise_sense  : logical;
  orientation      : direction;
WHERE
  (target <> location_shape_element);
  (origin <> location_shape_element);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

target: The feature which is the "to" feature of the dimension.

origin: The feature which is the "from" feature of the dimension.

clockwise_sense: TRUE if the measurement is clockwise with respect to the orientation vector.

orientation: A vector defining the Right Hand Rule for measurement.

PROPOSITIONS:

1. Neither target nor origin can be a location shape element.

4.10.3.5 BI-DIRECTIONAL ANGLE DIMENSION

The feature within a **bi-directional angle** (angle size) **dimension** is a **feature of size** and must be composed of linear elements equally disposed about the center of symmetry.

A **bi-directional angle dimension** requires one additional piece of information to determine the toleranced angle value: a flag indicating that the angle of interest is the one less than 180 degrees (TRUE) or greater than 180 degrees (FALSE).

```
*)
ENTITY bidirectional_angle_dimension
  SUBTYPE OF (angle_dimension);
  dimensioned_entity : feature_of_size;
  center_of_symmetry : OPTIONAL geometry;
  smaller_angle      : LOGICAL;
WHERE
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

dimensioned_entity: The **feature of size** being dimensioned. The dimension applies across the center of symmetry.

center_of_symmetry: The center of symmetry.

smaller_angle: TRUE if the measurement angle is less than 180 degrees.

PROPOSITIONS:

4.10.3.6 ANGLE PARAMETER DIMENSION

An **angle parameter dimension** is used in the definition of some **implicit form features**, such as the angle of a V-groove. The dimensional parameter of the form feature may be explicitly called out (an **angle parameter**) or may be derivable from other parameters of the form feature (**derivable angle dimension**).

```
*)
ENTITY angle_parameter_dimension
  SUPERTYPE OF (angle_parameter XOR
                derivable_angle_dimension)
  SUBTYPE OF (angle_dimension);
  measurement : dimension_measurement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

measurement: Specification of whether the dimension is the full angle or the half angle.

4.10.3.7 ANGLE PARAMETER

An **angle parameter** is an explicitly specified parameter of an angular dimensional characteristic of an **implicit form feature.**

```
*)
ENTITY angle_parameter
  SUBTYPE OF (angle_parameter_dimension);
  nominal_angle : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

nominal_angle: The nominal value of an angle.

4.10.3.8 DERIVABLE ANGLE DIMENSION

An **angle dimension** whose value can be derived (see **derivable dimension**).

```
*)
ENTITY derivable_angle_dimension
  SUBTYPE OF (angle_parameter_dimension);
END_ENTITY;
(*
```

4.10.3.9 LOCATION DIMENSION

A **location dimension** is the measurement of the location of one feature with respect to another feature. It is either Directed (i.e there is a distinction between the origin and the target of the Dimension) or is Bi-directional. The value of the **location dimension** is the distance between two parallel shape representation elements or things derived from shape representation elements (any combination of planar areas, linear seam or corners, or two elements separated by a constant distance. Each feature within a **location dimension** consists of two characteristics:

1. the thing which is the origin of the dimension, and
2. the actual shape representation element (surface/area, curve/seam or point/corner) on the object which is being controlled (or is the controlling surface of) the dimension.

These components may be the same thing, i.e there does not have to be two distinct entities (in fact, the thing being controlled may be implied through the thing being dimensioned).

The value of the dimension is typically measured along a linear path, although arc length and "true" dimensions require that a path be specified to determine the actual value.

This is not to imply that **location dimensions** may apply only to linear, planar and offset curves and surfaces. A complex (shape representation elements that are not planar, linear or cylindrical) Area or Seam (the thing which is being controlled) may be algorithmically resolved (through a **geometric derivation**) to a plane or line (the thing which is dimensioned), thereby satisfying the stipulations above.

```
*)
ENTITY location_dimension
  SUBTYPE OF (coordinate_dimension);
  range          : OPTIONAL coordinate_tolerance_range;
  target         : dt_feature;
  origin         : dt_feature;
  directed       : LOGICAL;
  linear         : LOGICAL;
  measure_path : OPTIONAL geometric_derivation;
WHERE
  ((linear AND (measure_path = null)) OR
   ((NOT linear) AND (measure_path <> NULL)));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

range: The optional value of the tolerance range.

target: The feature which is the "to" feature of the dimension.

origin: The feature which is the "from" feature of the dimension.

directed: TRUE if the measurement is "from" **origin** "to" **target** or FALSE if there is no precedence between **origin** and **target**.

linear: TRUE if the **measure path** is linear.

measure_path: The path along which the measurement is made e.g a non-linear curve.

PROPOSITIONS:

1. **Measure path** must be specified for a non-linear path and must not be specified for a linear path.
2. The **target** and **origin** entities called out must either be planar, linear or a point, OR they must be offset from one another by a uniform distance.

4.10.3.10 SIZE DIMENSION

A **size dimension** is a measure of an individual feature of the shape of an object. The tolerance is a numeric range that constrains the value of the dimension.
There are three characteristics of a **size dimension**:

1. the dimensional value (implicitly defined within the representation geometry or explicitly called out in a **feature** definition),
2. the shape representation elements which define the feature of interest,
3. the center of symmetry of the elements of the feature.

The value of a **size dimension** is expressed as a magnitude and is independent of the location of the feature. The feature of interest must be a **feature of size** (see — ANSI Y14.5) which are characterized by a point, curve or surface of symmetry. The value of the **size dimension** is determined by measuring in a straight line from a point on the feature through the center of symmetry (normal to the curve or surface of symmetry) to another point on the feature opposite the first. In most cases, the determination of the dimensional value is trivial, as in the diameter of a hole or the width of a slot.

```
*)
ENTITY size_dimension
  SUPERTYPE OF (size_characteristic_dimension XOR
                size_parameter_dimension)
  SUBTYPE OF (coordinate_dimension);
  range : OPTIONAL coordinate_tolerance_range;
  measurement : dimension_measurement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

range: The optional values of the tolerance range.

measurement: Specification of whether the dimension is the full measurement from one "side" of the feature to the other across the center of symmetry or is the "half" measurement between one "side" and the center of symmetry.

4.10.3.11 SIZE CHARACTERISTIC DIMENSION

A **size characteristic dimension** is the dimensional value calculated from shape elements which are explicitly contained in the model. The measurable characteristic of the shape used to determine this value must be a **feature of size**.

```
*)
ENTITY size_characteristic_dimension
  SUBTYPE OF (size_dimension);
  dimensioned_entity : feature_of_size;
  center_of_symmetry : geometry;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

dimensioned_entity: A **feature of size**.

center_of_symmetry: The Center of Symmetry of the **feature of size**.

4.10.3.12 SIZE PARAMETER DIMENSION

Implicit form features typically call out size parameters in the definition of the feature, such as the diameter of a hole. A **size parameter dimension** is a type of **size dimension** that specifies the value of the parameter and a tolerance on the parameter (inherited from its SUPERTYPE **size dimension**). A **size parameter dimension** may either be explicitly specified as a parameter value (a **size parameter**), or may be a derivable measurement of the feature (a **derivable dimension**).

```
*)
ENTITY size_parameter_dimension
  SUPERTYPE OF (size_parameter XOR
                derivable_dimension)
  SUBTYPE OF (size_dimension);
END_ENTITY;
(*
```

4.10.3.13 SIZE PARAMETER

A **size parameter** is a real value that has been explicitly called out as a parameter of an implicit form feature.

```
*)
ENTITY size_parameter
  SUBTYPE OF (size_parameter_dimension);
  nominal_size : real;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

nominal_size: The nominal value of the size.

PROPOSITIONS:

4.10.3.14 DERIVABLE DIMENSION

A dimensional characteristic of a form feature which is not among the independent ("minimal") attributes chosen to specify feature shape. It is used by the form features information model (FFIM) in order that tolerances on derivable dimensions of form features can be handled. It differs from the **size parameter** entity of the shape tolerance information model (STIM) only by not having an attribute giving the value of the dimension of interest.

Perhaps the following example will clarify **derivable dimension** and its motive. Suppose that the FFIM has a RECTANGLE entity. Any two of the following will suffice to specify a rectangle: LENGTH, WIDTH, DIAGONAL. STEP policy apparently requires that two of these be chosen; the FFIM opts for LENGTH and WIDTH, which is done by two "pointers" to **size parameter** (which has an attribute explicitly giving the parameter's value). But it is known that someone (television manufacturers?) will wish to specify rectangles by toleranced width and diagonal. This would be modeled as follows:

1. The **size parameter** giving WIDTH would have an associated **tolerance range**, signalling that WIDTH is a functional dimension.
2. The **size parameter** giving LENGTH would not have an associated **tolerance range**. Thus WIDTH would be used to specify nominal shape, but would be known not to be a functional dimension.
3. The **derivable dimension** associated with DIAGONAL would have an associated **tolerance range**, signalling that DIAGONAL is a functional dimension whose nominal value can be calculated.

Note that the above (1) works, but (2) may seem contrived. Handling of the situation is being addressed as an integration issue between the STIM and FFIM. The final solution may differ from that above.

```
*)
ENTITY derivable_dimension
  SUBTYPE OF (size_parameter_dimension);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:
PROPOSITIONS:

4.10.4 Tolerancing FUNCTION Definitions

4.10.4.1 IS CIRCULAR

This function returns TRUE if its input argument, or members if it is aggregate, has a circular cross-section in a plane normal to an axis.

```
*)
FUNCTION is_circular(arg: GENERIC) : LOGICAL;
END_FUNCTION;
(*
```

4.10.4.2 IS DISK

This function returns TRUE if its argument, or members if it is an aggregate, is normal to an axis, planar and has a circular boundary.

```
*)
FUNCTION is_disk(arg: GENERIC): LOGICAL;
  LOCAL
    result : LOGICAL := TRUE;
  END_LOCAL;
  IF NOT is_circular(arg) THEN
    result := FALSE;
  END_IF;
  IF NOT planar(arg) THEN
    result := FALSE;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.10.4.3 IS PARALLEL

This function returns TRUE if its arguments are parallel.

```
*)
FUNCTION is_parallel(arg1,arg2: GENERIC): LOGICAL;
END_FUNCTION;
(*
```

4.10.4.4 LINEAR SECTIONS

```
*)
FUNCTION linear_sections(dir: direction; arg: generic): LOGICAL;
END_FUNCTION;
(*
```

This function returns TRUE if all cross-sections of ARG taken in planes tangent to DIR are straight lines.

4.10.4.5 VALID MLSN

This function returns TRUE if the value of TOL_MLSN is appropriate for the toleranced entity.

The constraint states that if a toleranced "object" is symmetric about a point, curve or surface, then the associated MATERIAL CONDITION modifier must be *maximum material condition*, *least material condition* or *regardless of feature size*. If the object is not symmetric, then the modifier value is *not applicable*.

```
*)
FUNCTION valid_mlsn(mat : tol_mlsn; arg : GENERIC): LOGICAL;
  IF is_symmetric(arg) THEN
    CASE mat  OF
      maxmc      : RETURN(TRUE);
      leastmc    : RETURN(TRUE);
      regardless : RETURN(TRUE);
      OTHERWISE  : RETURN(FALSE);
    END_CASE;
  ELSE
    BEGIN
      IF (mat = none) THEN
        RETURN(TRUE);
      ELSE
        RETURN(FALSE);
      END_IF;
    END;
  END_IF;
END_FUNCTION;
(*
```

4.10.5 Tolerancing Classification Structure

The following indented list provides the classification structure for the Tolerancing Integrated Product Information Model.

```
COORDINATE DIMENSION
  ANGLE DIMENSION
    ANGLE PARAMETER DIMENSION
      ANGLE PARAMETER
      DERIVABLE ANGLE DIMENSION
    BIDIRECTIONAL ANGLE DIMENSION
    DIRECTED ANGLE DIMENSION
  LOCATION DIMENSION
  SIZE DIMENSION
    SIZE CHARACTERISTIC DIMENSION
    SIZE PARAMETER DIMENSION
      DERIVABLE DIMENSION
      SIZE PARAMETER
DATUM
  CONDITIONED DATUM
  UNCONDITIONED DATUM
FEATURE OF SIZE
  SIZE FEATURE
  FORM FEATURE OF SIZE
GEOMETRIC DERIVATION
PROJECTED TOLERANCE ZONE
TOLERANCE
  SHAPE TOLERANCE
    COORDINATE TOLERANCE RANGE
    GEOMETRIC TOLERANCE
      ANGULARITY
      CIRCULAR RUNOUT
      CIRCULARITY
      CONCENTRICITY
      CYLINDRICITY
      FLATNESS
      PARALLELISM
      PERPENDICULARITY
      POSITION
      PROFILE LINE
      PROFILE SURFACE
      STRAIGHTNESS
      TOTAL RUNOUT
TOLERANCE MAGNITUDE
TOLERANCE RANGE
  COORDINATE TOLERANCE RANGE
  NULL
TOLERANCED REAL

*)
END_SCHEMA; -- end of TOLERANCING schema
(*
```

4.11 Materials

4.11.1 Introduction

```
*)
SCHEMA ipim_material_schema;

  EXPORT EVERYTHING;

(*
```

4.11.2 MATERIAL PROPERTY

This entity contains data that describes a material constitutive matrix and other material properties. A material constitutive matrix is the relationship between stress and strain. For mechanical problems it is called an elasticity matrix.

In the general case, the thermoelastic behavior of real materials is relatively complex and is influenced by factors including the material properties, magnitude and type of the loads, temperature, time, the rate of loading or deformation, and the previous history of the material. At any point of the material, however, at any given time, temperature, and rate of loading or deformation, the behavior can be characterized by a constitutive law for the material. The constitutive laws express the relationships between the material internal forces, or stresses, and material deformations, or strains. At any point within a material, this incremental relationship between stresses and strains is characterized by a set of elastic moduli, thermal expansion coefficients, and structural damping coefficients which characterize the material behavior. These constants therefore define the material's properties.

```
*)
ENTITY material_property
  SUPERTYPE OF (homogeneous_material XOR
                composite_material XOR
                material_table);
  material_name  : OPTIONAL STRING;
  material_units : units;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

material_name: Descriptive name for the material (e.g steel).

material_units: The fundamental units of measure in which the material properties are defined.

4.11.2.1 HOMOGENEOUS MATERIAL

This entity collects together material properties for materials that may be considered to be uniform in constituency.

```
*)
ENTITY homogeneous_material
  SUPERTYPE OF (isotropic_material XOR
```

```
                orthotropic_material XOR
                anisotropic_material)
  SUBTYPE OF (material_property);
  mass_density                    : OPTIONAL REAL;
  structural_damping_coefficient : OPTIONAL REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

mass_density: The mass density of a material is its mass per unit volume.

structural_damping_coefficient: The structural damping coefficient is the measure of the percentage dissipation of energy as the material undergoes elastic deformations.

4.11.2.2 ISOTROPIC MATERIAL

This entity represents a material whose constitutive properties are the same in all directions.

```
*)
ENTITY isotropic_material
  SUBTYPE OF (homogeneous_material);
  thermal_property     : OPTIONAL isotropic_thermal_property;
  structural_property : OPTIONAL isotropic_structural_property;
  expansion_property   : OPTIONAL isotropic_thermal_expansion;
WHERE
  EXISTS(thermal_property) OR
  EXISTS(structural_property) OR
  EXISTS(expansion_property);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

thermal_property: The material thermal properties.

structural_property: The material structural properties.

expansion_property: The material expansive properties.

PROPOSITIONS:

1. At least one attribute must be present.

4.11.2.3 ISOTROPIC THERMAL PROPERTY

This entity contains isotropic thermal material information.

```
*)
ENTITY isotropic_thermal_property;
  thermal_conductivity_coefficient : REAL;
```

```
  specific_heat_capacity              : REAL;
  convective_film_coefficient         : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

specific_heat_capacity: The heat capacity of a material is the total quantity of heat required to produce a one unit change in temperature for a unit mass of the material at constant pressure. The heat capacity is a scalar quantity independent of direction for all material types, whether isotropic, orthotropic, or anisotropic.

thermal_conductivity_coefficient: Fourier's law of heat conduction relates the rate of heat flow through any surface area within the material to the temperature gradient normal to that surface. The constant of proportionality is the material thermal conductivity, which for isotropic materials is independent of direction.

convective_film_coefficient: The convective film coefficient is the constant of proportionality that relates the magnitude of the heat flux normal to an external material boundary to the temperature difference occurring across the boundary.

4.11.2.4 ISOTROPIC STRUCTURAL PROPERTY

This entity contains isotropic structural material information.

```
*)
ENTITY isotropic_structural_property;
  youngs_modulus   : REAL;
  poissons_ratio   : REAL;
  shear_modulus    : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

youngs_modulus: Young's modulus for an isotropic elastic material is the ratio of stress to strain observed in a simple uniaxial tension test with traction-free lateral surfaces. Young's modulus is denoted by E.

poissons_ratio: Poisson's ratio represents the ratio of lateral contraction to longitudinal extension in the simple uniaxial tension test.

shear_modulus: The shear modulus, G, of an isotropic material represents the ratio of shearing stress to shearing strain. Whatever the stress system may be, the ratio is measured by relating the shearing strain for any pair of rectangular axes and the shearing stress on a pair of planes orthogonal to these axes.

4.11.2.5 ISOTROPIC THERMAL EXPANSION

This data entity contains isotropic thermal expansion information.

```
*)
ENTITY isotropic_thermal_expansion;
  thermal_expansion_coefficient     : REAL;
  thermal_expansion_reference_temp  : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

thermal_expansion_coefficient: When the temperature of a small portion of an unrestrained isotropic material is changed by an amount, ΔT, a change in dilation proportional to ΔT is produced without any corresponding change in stress. The ratio of the induced extensional strain to the temperature change, ΔT, is the thermal expansion coefficient for the material.

thermal_expansion_reference_temp: The thermal expansion reference temperature is the temperature about which all thermal expansion phenomena are related.

4.11.2.6 ORTHOTROPIC MATERIAL

This entity represents a material whose constitutive properties are orthogonal, but not the same in the three principal directions.

```
*)
ENTITY orthotropic_material
  SUBTYPE OF (homogeneous_material);
  thermal_property    : OPTIONAL orthotropic_thermal_property;
  structural_proerty  : OPTIONAL orthotropic_structural_property;
  expansion_property  : OPTIONAL orthotropic_thermal_expansion;
WHERE
  EXISTS(thermal_property) OR
  EXISTS(structural_property) OR
  EXISTS(expansion_property);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

thermal_property: The material thermal properties.

structural_property: The material structural properties.

expansion_property: The material expansive properties.

PROPOSITIONS:

1. At least one attribute must be present.

4.11.2.7 ORTHOTROPIC THERMAL PROPERTY

This data entity contains orthotropic thermal material information.

```
*)
ENTITY orthotropic_thermal_property
  SUPERTYPE OF (orthotropic_thermal_2d_property XOR
                orthotropic_thermal_3d_property);
  in_plane_thermal_conductivities : ARRAY [1:2] OF REAL;
  specific_heat_capacity          : REAL;
  convective_film_coefficient     : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

in_plane_thermal_conductivities: K_1 and K_2 are the in-plane thermal conductivities.

specific_heat_capacity: The heat capacity of a material is the total quantity of heat required to produce a one unit of change of temperature for a unit mass of the material at constant pressure. The heat capacity is a scalar quantity independent of direction for all material types, whether isotropic, orthotropic, or anisotropic.

convective_film_coefficient: The convective film coefficient is the constant of proportionality that relates the magnitude of the heat flux normal to an external material boundary to the temperature difference occurring across the boundary.

4.11.2.8 ORTHOTROPIC THERMAL 2D PROPERTY

This entity represents orthotropic thermal 2D material property information.

```
*)
ENTITY orthotropic_thermal_2d_property
  SUBTYPE OF (orthotropic_thermal_property);
END_ENTITY;
(*
```

4.11.2.9 ORTHOTROPIC THERMAL 3D PROPERTY

This entity represents orthotropic thermal 3D material property information.

```
*)
ENTITY orthotropic_thermal_3d_property
 SUBTYPE OF (orthotropic_thermal_property);
  out_of_plane_thermal_conductivity : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

out_of_plane_thermal_conductivity: K_3 is the out of plane thermal conductivity.

4.11.2.10 ORTHOTROPIC STRUCTURAL PROPERTY

This entity contains orthotropic structural material information.

```
*)
ENTITY orthotropic_structural_property
  SUPERTYPE OF (orthotropic_structural_2d_property XOR
                orthotropic_structural_3d_property);
  inplane_youngs_moduli   : ARRAY [1:2] OF REAL;
  inplane_poissons_ratio : REAL;
  inplane_shear_modulus   : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

inplane_youngs_moduli: Young's modulus in the longitudinal direction for an orthotropic elastic material is the ratio of longitudinal stress to longitudinal strain observed in a simple longitudinal tension test with traction free transverse and lateral surfaces. Young's modulus for the longitudinal direction is denoted by E_{11}.

Young's modulus in the lateral direction for an orthotropic elastic material is the ratio of stress to strain observed in a simple lateral tension test with traction free longitudinal and transverse surfaces. Young's modulus for the lateral direction is denoted by E_{22}.

inplane_poissons_ratio: The in-plane Poisson's ratio, v_{12}, for an orthotropic material represents the ratio of lateral contraction to longitudinal extension in a simple in-plane longitudinal tension test.

inplane_shear_modulus: The in-plane shear modulus, G_{12}, of an orthotropic material represents the ratio of shearing stress to shearing strain for in-plane shearing deformations.

4.11.2.11 ORTHOTROPIC STRUCTURAL 2D PROPERTY

This entity represents orthotropic structural 2D material information.

```
*)
ENTITY orthotropic_structural_2d_property
  SUBTYPE OF (orthotropic_structural_property);
END_ENTITY;
(*
```

4.11.2.12 ORTHOTROPIC STRUCTURAL 3D PROPERTY

This entity contains orthotropic structural 3D material information.

```
*)
ENTITY orthotropic_structural_3d_property
  SUBTYPE OF (orthotropic_structural_property);
  out_of_plane_youngs_modulus : REAL;
  out_of_plane_poissons_ratio : ARRAY [1:2] OF REAL;
```

```
  out_of_plane_shear_moduli    : ARRAY [1:2] OF REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

out_of_plane_youngs_modulus: G_{23} and G_{31} are the out-of-plane shear moduli.

out_of_plane_poissons_ratio: V_{23} and V_{31} are the out-of-plane Poisson's ratios.

out_of_plane_shear_moduli: E_{33} is the out-of-plane Young's modulus.

4.11.2.13 ORTHOTROPIC THERMAL EXPANSION

This entity contains orthotropic thermal expansion information.

```
*)
ENTITY orthotropic_thermal_expansion
  SUPERTYPE OF (orthotropic_thermal_2d_expansion XOR
                orthotropic_thermal_3d_expansion);
  inplane_thermal_expansion_coefficients  : ARRAY [1:2] OF REAL;
  thermal_expansion_reference_temperature : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

inplane_thermal_expansion_coefficients: The thermal expansion coefficients, A_1, A_2 and A_3, are the ratios of induced extensional strains in one of the principal material directions due to an applied temperature change, ΔT. The coefficient A_1 is for the longitudinal direction, A_2 applies to expansion in the lateral direction, and A_3 is the out-of-plane thermal expansion coefficient.

thermal_expansion_reference_temperature: The thermal expansion reference temperature is the temperature about which all thermal expansion phenomena are related to.

4.11.2.14 ORTHOTROPIC THERMAL 2D EXPANSION

This entity defines orthotropic thermal 2D expansion material properties.

```
*)
ENTITY orthotropic_thermal_2d_expansion
  SUBTYPE OF (orthotropic_thermal_expansion);
END_ENTITY;
(*
```

4.11.2.15 ORTHOTROPIC THERMAL 3D EXPANSION

This entity defines orthotropic thermal 3D expansion material properties.

```
*)
ENTITY orthotropic_thermal_3d_expansion
  SUBTYPE OF (orthotropic_thermal_expansion);
  out_of_plane_thermal_expansion_coefficient : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

out_of_plane_thermal_expansion_coefficient: A_3 is the out of plane thermal expansion coefficient.

4.11.2.16 ANISOTROPIC MATERIAL

This entity represents a material whose constitutive properties are not orthogonal.

```
*)
ENTITY anisotropic_material
  SUBTYPE OF (homogeneous_material);
  thermal_property     : OPTIONAL anisotropic_thermal_property;
  structural_property  : OPTIONAL anisotropic_structural_property;
  expansion_property   : OPTIONAL anisotropic_thermal_expansion;
WHERE
  EXISTS(thermal_property) OR
  EXISTS(structural_property) OR
  EXISTS(expansion_property);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

thermal_property: The material thermal properties.

structural_property: The material structural properties.

expansion_property: The material expansive properties.

PROPOSITIONS:

1. At least one attribute must be present.

4.11.2.17 ANISOTROPIC STRUCTURAL PROPERTY

This entity defines anisotropic structural properties.

```
*)
ENTITY anisotropic_structural_property
  SUPERTYPE OF (anisotropic_structural_2d_property XOR
                anisotropic_structural_3d_property);
END_ENTITY;
(*
```

4.11.2.18 ANISOTROPIC STRUCTURAL 2D PROPERTY

The structural properties of a 2D anisotropic material.

```
*)
ENTITY anisotropic_structural_2d_property
  SUBTYPE OF (anisotropic_structural_property);
  material_elasticity_half_3x3_matrix : ARRAY [1 : 6] OF REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

material_elasticity_half_3x3_matrix: The 3x3 material matrix of elastic coefficients relates stress to strain by the relationship (insert equation here???????).

4.11.2.19 ANISOTROPIC STRUCTURAL 3D PROPERTY

The structural properties of a 3D anisotropic material.

```
*)
ENTITY anisotropic_structural_3d_property
  SUBTYPE OF (anisotropic_structural_property);
  material_elasticity_half_6x6_matrix : ARRAY [1 : 21] OF REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

material_elasticity_half_6x6_matrix: The 6x6 material matrix of elastic coefficients relates stress to strain by the relationship (insert equation here???????).

4.11.2.20 ANISOTROPIC THERMAL PROPERTY

The thermal properties of an anisotropic material.

```
*)
ENTITY anisotropic_thermal_property
  SUPERTYPE OF (anisotropic_thermal_2d_property XOR
                anisotropic_thermal_3d_property);
  specific_heat_capacity : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

specific_heat_capacity: The heat capacity of a material is the total quantity of heat required to produce a one unit of change of temperature for a unit mass of the material at constant pressure. The heat capacity is a scalar quantity independent of direction for all material types, whether isotropic, orthotropic, or anisotropic.

4.11.2.21 ANISOTROPIC THERMAL 2D PROPERTY

This data entity contains anisotropic thermal 2D material information.

```
*)
ENTITY anisotropic_thermal_2d_property
  SUBTYPE OF (anisotropic_thermal_property);
  thermal_conductivity    : ARRAY [1 : 2] OF REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

thermal_conductivity: For a 2D anisotropic thermal material, the thermal conductivities are the two unequal constants of proportionality that relate the rate of heat flow through a surface to the temperature gradient.

4.11.2.22 ANISOTROPIC THERMAL 3D PROPERTY

This data entity contains anisotropic thermal 3D material information.

```
*)
ENTITY anisotropic_thermal_3d_property
  SUBTYPE OF (anisotropic_thermal_property);
  thermal_conductivity    : ARRAY [1 : 3] OF REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

thermal_conductivity: For a 3D anisotropic thermal material, the thermal conductivities are the three unequal constants of proportionality that relate the rate of heat flow through a surface to the temperature gradient, denoted as K_{xx}, K_{yy} and K_{zz}.

4.11.2.23 ANISOTROPIC THERMAL EXPANSION

The thermal expansion properties of an anisotropic material.

```
*)
ENTITY anisotropic_thermal_expansion
  SUPERTYPE OF (anisotropic_thermal_2d_expansion XOR
                anisotropic_thermal_3d_expansion);
  thermal_expansion_reference_temp : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

thermal_expansion_reference_temp: The temperature about which the thermal expansion phenomena are related.

4.11.2.24 ANISOTROPIC THERMAL 2D EXPANSION

This entity contains anisotropic 2D thermal expansion information.

```
*)
ENTITY anisotropic_thermal_2d_expansion
  SUBTYPE OF (anisotropic_thermal_expansion);
  thermal_expansion_coefficients    : ARRAY [1:3] OF REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

thermal_expansion_coefficients: (A_1, A_2, A_{12}). The ratios of the induced extensional strains to the temperature change, ΔT, for an element with 2D anisotropic material are the thermal expansion coefficients for the respective material directions.

4.11.2.25 ANISOTROPIC THERMAL 3D EXPANSION

This entity contains anisotropic 3D thermal expansion information.

```
*)
ENTITY anisotropic_thermal_3d_expansion
  SUBTYPE OF (anisotropic_thermal_expansion);
  thermal_expansion_coefficients    : ARRAY [1:6] OF REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

thermal_expansion_coefficients: $(A_1, A_2, A_3, A_4, A_5, A_6)$???? The ratios of the induced extensional strains to the temperature change, ΔT, for an anisotropic material are the thermal expansion coefficients A_1, A_2 and A_3 for the material.

4.11.2.26 COMPOSITE MATERIAL

This data entity represents the concept that some materials are defined as the combination of several materials combined together to form a composite material. Presently, there are three different types of composite materials defined. These are **mixture composite material**, **halpin-tsai material**, and **laminate composite material**. Each type of composite material represents a different computational theory with which to calculate the material's constitutive properties.

```
*)
ENTITY composite_material
  SUPERTYPE OF (halpin_tsai_material XOR
                laminate_composite_material XOR
                mixture_composite_material)
  SUBTYPE OF (material_property);
END_ENTITY;
(*
```

4.11.2.27 MIXTURE COMPOSITE MATERIAL

This data entity indicates a mixture composite material.

```
*)
ENTITY mixture_composite_material
  SUBTYPE OF (composite_material);
  mixture_material : LIST [1 : #] OF mixture_material_property;
WHERE
  sum(mixture_material.mixture_fraction) = 1.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

mixture_material: Three mixture material attributes are defined in the mixture material property. These attributes are related. For each mixture material, a volume material number will exist and point to material properties for that type of material. For each volume material number, there will exist a volume fraction which is a fractional amount of this type of material in this element. The sum of the volume fractions for all volume material numbers in this element must be 1. There will also be a corresponding volume orientation for each volume material number which defines the direction of the volume material with respect to the local element coordinate system.

PROPOSITIONS:

1. The material fractions must sum to 1.

4.11.2.28 MIXTURE MATERIAL PROPERTY

This entity represents a composite material whose material properties are determined by the rule of mixtures.

```
*)
ENTITY mixture_material_property;
  volume_fraction     : REAL;
  volume_orientation  : REAL;
  volume_material     : material_property;
WHERE
  (0.0 < volume_fraction < 1.0);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

volume_fraction: A fractional amount of the volume material in this element.

volume_orientation: Defines the direction of the volume material with respect to the local element coordinate system.

volume_material: Material properties for this type of material.

PROPOSITIONS:

1. The volume fraction must be greater than zero and less than one.

4.11.2.29 HALPIN TSAI MATERIAL

This data entity represents a composite material whose material properties are determined by the theory of Halpin-Tsai.

```
*)
ENTITY halpin_tsai_material
  SUBTYPE OF (composite_material);
  fiber_volume_fraction  : REAL;
  matrix_volume_fraction : REAL;
  coefficients           : LIST [1 : #] OF REAL;
  fiber_material         : material_property;
  matrix_material        : material_property;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

fiber_volume_fraction, matrix_volume_fraction: The fiber volume fraction and matrix volume fraction are two fractional numbers that sum to 1 describing the proportions of the mix of the fiber and matrix materials.

coefficients: The Halpin-Tsai coefficients are (see PDA manual).

fiber_material, matrix_material: The fiber material and matrix material are pointers to material property entities.

4.11.2.30 LAMINATE COMPOSITE MATERIAL

This entity represents a composite material whose material properties are determined by laminates (layers) stacked together. Each laminate is a single material of a constant thickness.

```
*)
ENTITY laminate_composite_material
  SUBTYPE OF (composite_material);
  laminate_z_offset : REAL;
  laminate_material : LIST [1 : #] OF laminate_material_property;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

laminate_z_offset: The laminate Z-offset is the distance from nodal plane (also mid-plane of laminate) to the top of the laminate.

laminate_material:

4.11.2.31 LAMINATE MATERIAL PROPERTY

This entity represents data about a laminate's material properties. Each laminate is defined by a material, in-plane orientation, and thickness.

```
*)
ENTITY laminate_material_property;
  ply_thickness    : REAL;
  ply_orientation  : REAL;
  ply_material     : material_property;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

ply_thickness: The laminate ply thickness is the thickness (in appropriate units) for the ply.

ply_orientation: The laminate ply orientation is the in-plane angle between the material's major direction and the local element coordinate system.

ply_material: The material property for the ply.

4.11.2.32 MATERIAL TABLE

This data entity represents the concept that a material's properties sometimes vary with respect to independent variables. This data entity implements this concept in a very general manner. Typically, in structural mechanics problems where there is a wide variation of temperature among a FEM's nodes, material property data is input as a tabular set of information with the independent variable being temperature and the dependent variables being the material's Young's Modulus, Poisson's Ratio, etc. These data are interpolated with respect to temperature to determine what material property value to use.

Because this entity (along with its companion entity, **material table instance**) recursively points to a **material property**, any number of independent variables may be defined.

```
*)
ENTITY material_table
  SUBTYPE OF (material_property);
  independent_variable_descriptor : STRING;
  material_table_instance_ref     : LIST [1 : #] OF
                                      material_table_instance;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

independent_variable_descriptor: This attribute represents the physical significance of the data contained within the **independent variable value** of the **material table instances**. For example, the value of this attribute might be 'Temperature'; i.e., 'Temperature' is the physical quantity for which other material properties pointed to by the **material table instance** data entity are measured with respect to. (See FEM/Issue-21 for a more complete discussion of the **material table** concept.)

material_table_instance_ref:

4.11.2.33 MATERIAL TABLE INSTANCE

Data pertaining to an independent variable value and material number is contained within this data entity. It is used in conjunction with a material table data entity to complete the specification of tabular material property by associating an independent variable with a pointer to another material property data entity.

```
*)
ENTITY material_table_instance;
  material_table_material_ref : material_property;
  independent_variable_value  : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

material_table_material_ref: The material property for the table; i.e., a pointer to the dependent material property that has been measured with respect to the value of the material table independent variable value.

independent_variable_value: Contains the value of the independent variable used as a reference in the material table instance data entity.

4.11.3 Material Classification Scheme

The following indented list provides the classification structure for the Material Schema.

```
ANISOTROPIC STRUCTURAL PROPERTY
  ANISOTROPIC STRUCTURAL 2D PROPERTY
  ANISOTROPIC STRUCTURAL 3D PROPERTY
ANISOTROPIC THERMAL EXPANSION
  ANISOTROPIC THERMAL 2D EXPANSION
  ANISOTROPIC THERMAL 3D EXPANSION
ANISOTROPIC THERMAL PROPERTY
  ANISOTROPIC THERMAL 2D THERMAL PROPERTY
  ANISOTROPIC THERMAL 3D THERMAL PROPERTY
ISOTROPIC STRUCTURAL PROPERTY
ISOTROPIC THERMAL EXPANSION
ISOTROPIC THERMAL PROPERTY
LAMINATE MATERIAL PROPERTY
MATERIAL PROPERTY
  COMPOSITE MATERIAL
    HALPIN TSAI MATERIAL
    LAMINATE COMPOSITE MATERIAL
    MIXTURE COMPOSITE MATERIAL
  MATERIAL TABLE
  HOMOGENEOUS MATERIAL
    ANISOTROPIC MATERIAL
    ISOTROPIC MATERIAL
    ORTHOTROPIC MATERIAL
```

```
MATERIAL TABLE INSTANCE
MIXTURE MATERIAL PROPERTY
ORTHOTROPIC STRUCTURAL PROPERTY
  ORTHOTROPIC STRUCTURAL 2D PROPERTY
  ORTHOTROPIC STRUCTURAL 3D PROPERTY
ORTHOTROPIC THERMAL EXPANSION
  ORTHOTROPIC THERMAL 2D EXPANSION
  ORTHOTROPIC THERMAL 3D EXPANSION
ORTHOTROPIC THERMAL PROPERTY
  ORTHOTROPIC THERMAL 2D PROPERTY
  ORTHOTROPIC THERMAL 3D PROPERTY

*)
  END_SCHEMA; -- end of MATERIAL schema
(*
```

4.12 Presentation

4.12.1 Introduction to Presentation

4.12.1.1 Scope of Presentation

Presentation assumes the task of specifying the structures which support the visualisation of product data.

The following is a pragmatic attempt at a definition:

> Presentation shall provide the entities which support the visualisation of data contained in a file, augmented by instructions of the sender governing the appearance of product model data as well as supplementary annotations. The presentation entities shall form a common basis for the needs of any application.

Ideally, this Standard must be capable of transmitting all the elements of a drawing or picture produced by the sender in such a way, that the receiver will be in a position to produce an accurate copy. The practical goal must be to get as close as possible to this ideal. In particular, any differences which may occur must be limited to such an extent that they do not change either appearance in global or meaning in detail. In many cases this accuracy is of legal importance, since the sender's drawing is the reference in case of dispute.

The commonalities of different applications relate to the way in which presentation attributes are associated with product model data or application specific data and to their final appearance, as well as to basic entities which should be defined in a single place with sufficient generality rather than separately in multiple variations.

```
*)
SCHEMA ipim_presentation_schema;

  EXPORT EVERYTHING;
  ASSUME(ipim_geometry_schema,
         ipim_drafting_schema);

(*
```

This version represents the owners' ideas on the content of presentation. It is obvious that several rounds of coordinating discussions will be required in order to arrive at an optimal content.

Specifically, the following parts are either incomplete or missing:

- Use of escape sequences
- Definition of composite symbols
- Instance of composite symbols
- Higher level entities
- Definition of "Object"
- Light sources

Auxiliary constructs were mentioned in the previous version. They have been left out since they belong either to drafting or geometry.

4.12.1.2 Status and Important Principles

As an international standard, the presentation entities must assure that the requirements for accurate reproduction based on the product data hold independent of language, character and text appearance (e.g Kanji characters or writing backwards) or local drafting standards. In international exchanges of product model data with the associated drawings it is a requirement that the drawings produced by the receiver conform to the rules and standards of the sender.

This should hold for line oriented presentations as well as for coloured surface pictures, or for any combination of the two.

Since, in many CAD systems, associativities exist between the product model and visual elements of a presentation it is highly desirable to be able to preserve such associativities, most importantly in the area of dimensioning and property related annotations. If this requirement did not exist, presentations might be transmitted by using the existing ISO standard for computer graphics metafiles (CGM, ISO 8632), pre- and post- processors for which would be much simpler. It may turn out that, for certain applications, this would provide an adequate solution. It is an open question whether CGM should appear as an embedded standard within this Standard.

As a principle, any existing ISO standards which overlap with presentation (or in fact with any other issue) should be used in the terminology as well as in the formal definition of the corresponding entities. This may allow the full or partial use of existing software.

The following overlaps are known and shall be used:

- ISO charactersets. The existance of multiple standards (ISO 8859, ISO 646, ISO 6937 and ISO 2022) represents a question of choice which is to be decided by the Physical File Group.
- ISO characterfont as part of the ISO drawing standard (ISO ...).
- ISO drawing standard (ISO ...) with its requirements for the appearance of drawings, dimensioning, annotations and symbols.
- ISO standard for computer graphics metafiles (ISO 8632) as mentioned above.

In the area of graphics systems, GKS-3D and PHIGS are the two main competing candidates, neither of which is currently an ISO standard. It appears that PHIGS with its development towards PHIGS+ is the preferable choice, since the overlap with this Standard, and hence the number of immediately usable features is larger than for GKS-3D. On the ISO front, PHIGS was advanced to DIS stage (SC24/WG2, last stop before coming an international standard) in December 1987 (*Computer Graphics, Vol 22, no 2, April 1988*).

Although it cannot fully meet the requirements of presentation, its terminology and a number of its concepts are of immediate use in the definition of prtesentation entities. PHIGS processors available at the receiver's end may be used but will require some preparatory computations, especially for proper handling of annotations. User defined fonts and mirroring are not available in PHIGS at this time.

In several cases, separate entities for 2D and 3D instances are defined, particularly where the 2D case will cover the majority of uses. This separation will facilitate the building of subsets.

The present specifications defined here for presentation items represent a first attempt towards these highly ambitious goals. They are not complete but should present a sound, well structured foundation allowing for improvement and extensions. Considerable generality is available and, in most cases, it is fairly obvious where and how extensions may be included.

4.12.1.3 Interface to Drafting and Applications

Drafting represents a single major area for the use of presentation entities. It requires flexible possibilities for the presentation of product model data and extensive possibilities for the addition of annotations, symbols, and area pattern styles.

Contacts with the working group on drafting have brought forth the majority of the requirements but details are still outstanding. It is expected, that other application areas will have further specific requirements of their own.

The interface questions relate to the method by which presentation attributes are defined and associated and to the basic entities provided for either direct use or composition of higher level entities, e.g., for dimensioning or other annotations.

Entities requiring coordination are present in the Electrical Applications Schema. No other requirements have been announced for Version 1 of this Standard but further needs are expected from areas such as FEM, Technical Publishing and others.

For many drafting aspects, the use of layers is essential, and furthermore, they are state of the art in most CAD systems. An infinite number of layers will be allowed for assignment to elements of a drawing.

In the past, it has not been stated or decided, that transformations may also apply to annotations, however, this is clearly available in IGES. The solution proposed is different, but quite general and corresponds to presentation methods available in some CAD systems.

A fair number of entities are considered to be the task of the drafting working group. Their definition is not given here. The following entities or groups of entities belong to this class:

- Auxiliary constructs (e.g., center lines)
- All entities for dimensional annotation
- All entities for non-dimensional annotation
- Sheet and Drawing administrative data

4.12.1.4 Organization of the Model

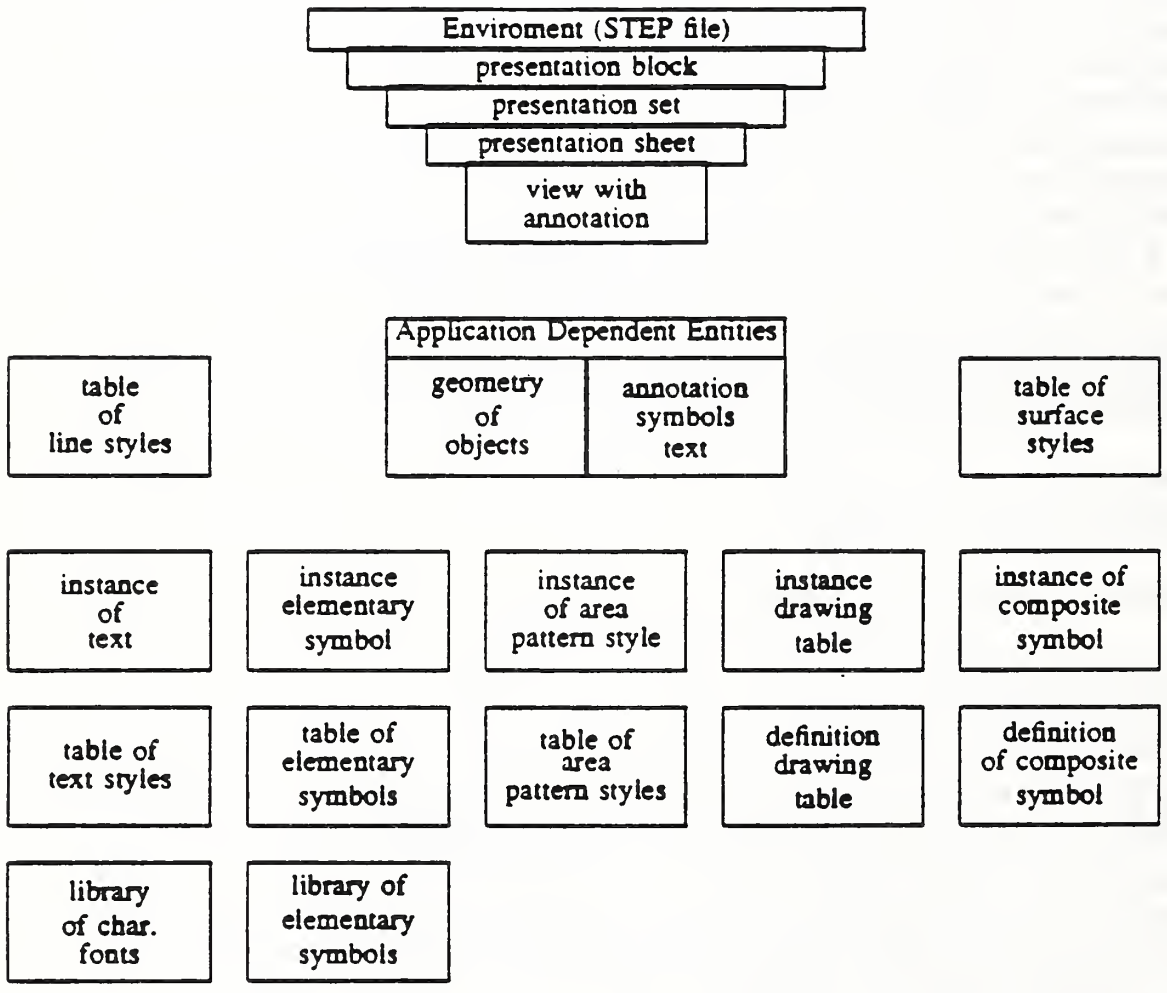
The majority of the data to be presented are usually derived from the geometry of the product model or from applications. Attributes associated with the geometrical items at different levels of the tree structure determine the intended visual appearance. A hierachy of presentation entities compatible with the needs for drafting allow for changes of attributes at each level, the lowest being the level of a single view. One part of presentation deals with this hierarchical structure and with the components of which a view with annotations is composed.

Another part of presentation defines the low-end entities which may serve as components for more complex composite structures or directly to determine the appearance of lines, text, symbols or area pattern styles. Between these two parts, any application may define its application specific entities as depicted by Figure 1.

This Figure shows how presentation requirements for applications are embedded in the presentation entities and makes it obvious that coordination of requirements is necessary on all sides.

The entities are defined below in a sequence such that any entity is defined prior to its use in the definition of another entity, with a few exceptions.

Figure 1: Low-end and high-end entities of presentation.



4.12.2 Basic Presentation Entities

4.12.2.1 RGB COLOR

Colors need to be defined for lines, text or surfaces and, therefor, appear at several places. Only an RGB definition shall be given, since transformations from CMY (cyan, magenta, yellow) or HSL (hue, saturation, lightness) are available (see *IGES 3.0, Appendix F*).

```
*)
ENTITY rgb_color;
  red   : REAL;
  green : REAL;
  blue  : REAL;
WHERE
  {0.0 <= red <= 1.0};
  {0.0 <= green <= 1.0};
  {0.0 <= blue <= 1.0};
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

red: First color value as an intensity.

green: Second color value as an intensity.

blue: Third color value as an intensity.

PROPOSITIONS:

1. Each color value must be in the range zero to one, inclusive.

4.12.2.2 SYMBOL 2D CURVE

Characters, elementary and composite symbols and tables on drawings require simple geometry. While characters are often described as polygons, many symbols contain circular segments. The following entity allows for interrupted sequences for both linear and circular elements.

The sequence is defined by a list of **points**, zeroes and **vectors**. Each **point** is a point on the curve. The value zero indicates an interruption of the line (e.g pen-up between the previous and next points). A **vector** is needed for a circular segment, in which case the vector points from the first point on the circle, which is the preceeding point in the list, to the second point, the third being the one which follows the vector.

If no vectors are used, the character is defined solely with straight line segments and, if no zero appears, a single uninterrupted sequence will define it.

```
*)
ENTITY symbol_2d_curve;
  linear_circular : LIST [1 : #] OF curve_interrupt;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

linear_circular: A list of **points**, zeroes and **vectors**.

4.12.2.2.1 ZERO

The number value zero.

```
*)
TYPE zero = 0;
END_TYPE;
(*
```

4.12.2.2.2 CURVE INTERRUPT

Required by symbol 2d curve.

```
*)
TYPE curve_interrupt = SELECT
     (cartesian_two_coordinate,
      zero,
      two_space_direction);
END_TYPE;
(*
```

4.12.2.3 PATTERN FILL AREA

A pattern fill area is used as the boundary for hatching or other area patterns. In the majority of cases this curve lies in a plane parallel to the projection plane. However, this is not a requirement, and the closed curve may even be an arbitrary spatial curve as long as its projection is not self intersecting.

```
*)
ENTITY pattern_fill_area;
  boundary : composite_curve;
WHERE
  boundary.closed_curve = TRUE;
END_ENTITY;
(*
```

boundary: A composite curve defining the boundary of the area.

PROPOSITIONS:

1. The boundary curve must be closed (and not self intersecting in the projections applied to it).

4.12.3 Libraries

4.12.3.1 Organization of Libraries

A library is a collection of data the use of which is not limited to the environment of a single file. Libraries may contain data from international standards, from national standards, or from privately agreed nonstandard definitions.

Libraries shall be organized to contain a list of lists of library elements, i.e., they are two-dimensional structures.

The inner list shall contain a sequence of library elements, defined as an entity whose first attribute is an identifying number.

Each element of the outer list shall consist of an identifying string, a header element and the inner list.

As an example, a character font library would contain multiple fonts, such identified by a string followed by information on the font and containing in its inner list a sequence of character definitions identified by numbers.

On the outer level a mechanism is needed which will search for a string, while at the inner level, a search for an integer value is required.

Libraries may be defined in the same files as the references to them. Since they often contain information which rarely changes, it is reasonable to have files containing only libraries. References to such libraries will then necessarily be external to the calling file.

The following libraries are defined below:

- character font library
- elementary symbol font library.

4.12.3.2 Character Font Libraries

A character font library describes the graphical form of a number of characters of the character set, but not necessarily all of them. For characters not present in a font library, a backup font is indicated.

Characters are defined inside a box. Within a font library, the character boxes all have the same height. The width is also the same unless the font defines proportional spacing. Characters should leave sufficient room on all four sides of the box to provide for aesthetic spacing if boxes are placed next to each other. Kerning, i.e., extending parts of a character beyond the box may lead to difficulties.

The definition of characters may occur in boxes of arbitrary sizes, provided that the same box height (bottom line to top line) is used within a font library.

4.12.3.3 SINGLE CHARACTER

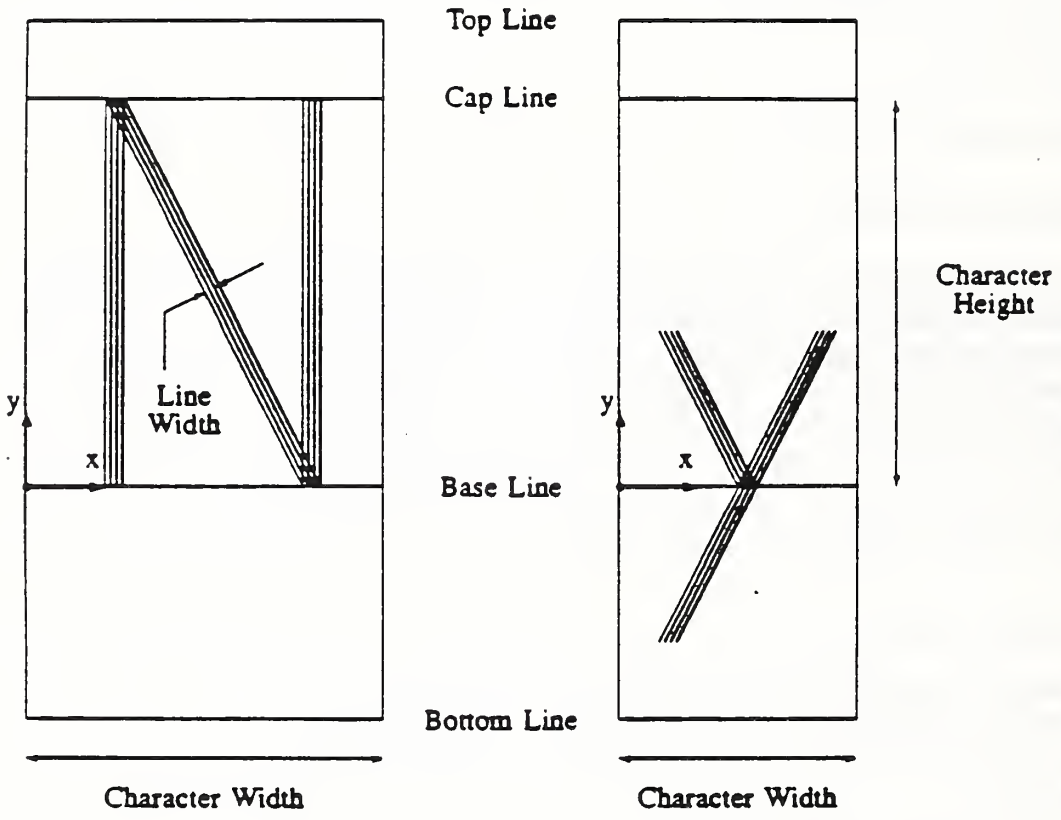
A single character is identified by its ISO code, i.e an integer in the range from xxx to yyy. Each character is defined in its own local coordinate system. Figure 2 shows the terminology used and the geometric relationship of the different lines and dimensions. This entity defines the entry of a single character into a font.

The character box defines a coordinate system whose origin is the intersection of the base line with the left border line of the box.

The position of the character in the box must allow for sufficient space symmetrically on both sides to guarantee the minimal spacing required between characters when boxes are joined next to each other. Likewise, sufficient space is required from the bottom line for descending characters, and on the top for accents or umlauts (see Figure 2).

```
*)
ENTITY single_character;
  iso_code             : INTEGER;
  width_of_character   : OPTIONAL REAL;
  graphic_definition   : symbol_2d_curve;
END_ENTITY;
(*
```

Figure 2: Definition of characters for fonts. Terminology and coordinate system.



ATTRIBUTE DEFINITIONS:

iso_code: Awaiting decision on which ISO standard will be used; the exact meaning of the code values is still open.

width_of_character: The width of the box containing the character. To be indicated only in the case of a proportional font. If an ISO standard is used which contains the accent characters of western European languages as separate characters, it is advisable to include them in the font with zero width, in which case they should precede the character to which they apply.

graphic_definition: Defines the shape of the charcter within the box.

4.12.3.4 CHARACTER FONT

A **character font** describes the shape of a number of characters of the characterset. For characters not present in the font, a backup font is indicated. This allows, for example, to have two sets of numbers of the same style — one for proprtional spacing and one with monospaced boxes for use in tables, whereby one of the sets need only contain the numbers. Likewise, some fonts will not contain the special symbols, and refer to a standard font for those.

The vertical layout, which is the same for all characters of the font is also defined here (see Figure 2).

It is not necessary if all the characters of the set are defined in the backup font. If it is missing or in the case of an error situation, the receiving system must default to the ISO characterset for drafting (ISO ...).

```
*)
ENTITY character_font;
  backup_font_library : OPTIONAL character_font_library;
  backup_font         : OPTIONAL character_font;
  line_width          : REAL;
  character_height    : REAL;
  top_line            : REAL;
  bottom_line         : REAL;
  character_width     : OPTIONAL REAL;
  font_content        : SET [1 : #] OF single_character;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

backup_font_library: A reference to a character font library. If this attribute is not present, the backup library is identical to the one in which this font is inserted.

backup_font_identification: The identification of the backup font within the backup library.

line_width: The linewidth which corresponds to the character height of this font.

character_height: The distance of the cap line from the base line, which is identical for the definition of all characters in the font.

top_line: The distance of the top line from the base line, i.e the character height plus a vertical spacing margin.

bottom_line: The distance of the bottom line of the characters from the base line, i.e the height of descenders plus a spacing margin.

character_width: The width of the box in the case of characters of fixed width, missing in the case of proportional spacing.

font_content: The characters for which the font information is provided.

```
*)
RULE font_rule FOR (single_character,
                    character_font);
  LOCAL
    i        : INTEGER;
    int_list : list_of_integer;
    int_set  : set_of_integer;
  END_LOCAL;
  REPEAT i := 1 TO SIZEOF(character_font.font_content);
    int_list := int_list + font_content[i].iso_code;
    int_set := int_set + font_content[i].iso_code;
    IF ((character_font.character_width = NULL) AND
        (font_content[i].width_of_character = NULL)) THEN
      VIOLATION;
    END_IF;
    IF ((character_font.character_width <> NULL) AND
        (font_content[i].width_of_character <> NULL)) THEN
      VIOLATION;
    END_IF;
  END_REPEAT;
  IF SIZEOF(int_list) <> SIZEOF(int_set) THEN
    VIOLATION;
  END_IF;
END_RULE;
(*
```

PROPOSITIONS:

1. If **character width** is not NULL then each character composing the font must have a NULL value for its **width of character** attribute.

2. If **character width** is NULL then each character composing the font must have a declared value for its **width of character** attribute.

3. The **iso code** for each character in a font must be UNIQUE within the font.

4.12.3.5 CHARACTER FONT LIBRARY

A **character font library** is a collection of an arbitrary number of **character fonts**. The library is accessible either directly through entries in the table of text styles or indirectly through the backup mechanism of the **character font**.

```
*)
```

```
ENTITY character_font_library;
  collection : SET [1 : #] OF character_font;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

collection: The collection of character fonts composing the library.

4.12.3.6 Elementary Symbol Libraries

Elementary symbols are symbols of fixed appearance, which may change in size and may possibly be slanted or mirrored.

Arrowheads may be defined as elementary symbols, in which case the line leading to the arrowhead is assumed to be the negative X axis. The line will terminate at the origin unless a reference point on the negative X axis is given.

Elementary symbols have one or multiple reference points. The first reference point is by default at the origin of the defining coordinate system.

4.12.3.7 FILLED SYMBOL AREA

A filled symbol area is a closed symbol 2d curve filled with the color specified for the symbol to which it belongs.

```
*)
ENTITY filled_symbol_area;
  filled_area : symbol_2d_curve;
WHERE
  closed(filled_area);
  NOT self_intersect(filled_area);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

filled_area: A symbol 2d curve defining the boundary of the area to be filled.

PROPOSITIONS:

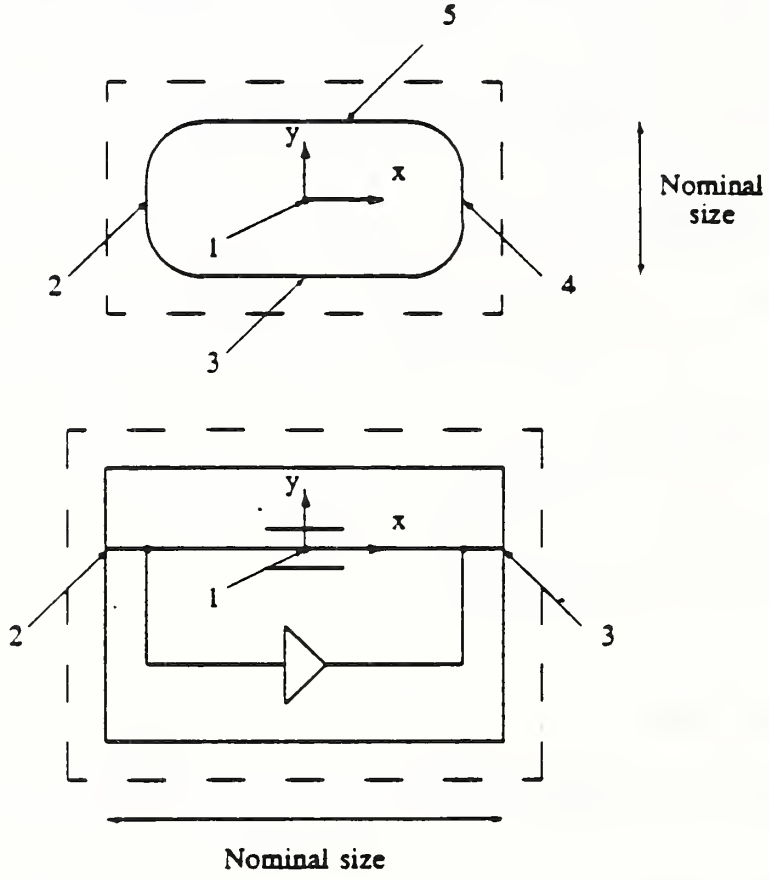
1. The symbol 2d curve must be closed.
2. The symbol 2d curve must not self intersect.

4.12.3.8 ELEMENTARY SYMBOL

Elementary symbols are defined in a two dimensional coordinate system which may be arbitrarily chosen with respect to the symbol. The origin of the coordinate system assumes the function of the first reference point, to which more may be added. See Figure 3.

The symbol shape may consist of lines and/or filled areas. Each symbol has a nominal size which may be its diameter, width or height. In an instance of a symbol the desired size is indicated whereafter the symbol is scaled by a factor equal to the quotient of the desired size and the nominal size.

Figure 3: Examples of elementary symbols with indications of nominal size, reference points (numbered) and optional blanking area (dashed).



```
*)
ENTITY elementary_symbol;
  code_number       : INTEGER;
  symbol_shape      : LIST [1:#] OF symbol_curve_or_area;
  reference_points  : LIST [1:#] OF UNIQUE cartesian_two_coordinate;
  blanking_box      : OPTIONAL LIST [2:2] OF
                              UNIQUE cartesian_two_coordinate;
  nominal_size      : REAL;
UNIQUE
  code_number;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

code_number: A unique integer value assigned to the symbol.

symbol_shape: The symbol shape may consist of solid lines and/or filled areas.

reference_points: The first reference point provides the local origin for the symbol. Additional reference points are needed when multiple lines are to be connected to different connection points of the symbol. Reference points are implicitly numbered.

blanking_box: Optionally, a blanking box may be indicated by giving first the left lower point of the rectangle and then the right upper point.

nominal_size: The nominal size of the symbol.

PROPOSITIONS:

1. Code number must be UNIQUE.

4.12.3.8.1 SYMBOL CURVE OR AREA

A choice between a symbol 2d curve and a filled symbol area.

```
*)
TYPE symbol_curve_or_area = SELECT
     (symbol_2d_curve,
      filled_symbol_area);
END_TYPE;
(*
```

4.12.3.9 ELEMENTARY SYMBOL FONT

An elementary symbol font is a grouping of the elementary symbols which usually belong to the same application area (e.g hydraulic, welding, electrical).

```
*)
ENTITY elementary_symbol_font;
  backup_font_library : OPTIONAL elementary_symbol_font_library;
```

```
  backup_font           : OPTIONAL elementary_symbol_font;
  line_width            : REAL;
  font_definition       : SET [1 : #] OF elementary_symbol;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

backup_font_library: A reference to the backup elementary symbol font library. If this attribute is not present, the backup library is identical to the one in which the font is inserted.

backup_font: The backup font within the backup library.

line_width: The linewidth corresponding to the nominal sizes of all the symbols in the font.

font_definition: The list of **elementary symbols** which form the font.

4.12.3.10 ELEMENTARY SYMBOL FONT LIBRARY

A **elementary symbol font library** is a collection of an arbitrary number of **elementary symbol fonts**. The library is accessible either through entries in the table of **elementary symbols** or through the backup mechanism of the **elementary symbol font**.

```
*)
ENTITY elementary_symbol_font_library;
  collection : SET [1 : #] OF elementary_symbol_font;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

collection: The **elementary symbol fonts** composing the library.

4.12.4 Tables

4.12.4.1 Organisation of Tables

Presentation attributes define the appearance of items (e.g the appearance of text). Tables are used to bundle presentation attributes. Although theoretically an infinite number of different combinations of attributes are possible, only a very small number of such combinations appear on a drawing. It is therefore more efficient to store the combinations in tables and then to use indices as pointers to attribute bundles.

The bundles proposed in the following may be somewhat extreme in their extensiveness, leaving only the really variable attributes to the instance entities.

Tables are defined as fixed length lists outside of presentation blocks. A single or multiple tables may exist of each type, but within a presentation block only one of each type may be activated.

Attribute bundles may be filled into tables inside or outside of presentation blocks, but each entry of the table may be filled only once. Obviously, an entry must have been filled prior to being used.

The following tables have been defined:

- Line styles

- Surface styles
- Geometric aspects of text
- Text styles
- Elementary symbol styles
- Area pattern fill styles

4.12.4.2 BUNDLE TABLE

A table of presentation attribute bundles.
The definition of a **bundle table** defines the size of an empty table and its reference. Multiple tables may be created. The tables to become active will be selected in the **presentation block.**
Any position in a table may be filled only once.
Entries for a **bundle table** can occur only at places where it is obvious which table is meant i.e in the definition of a table and within a presentation block after a specific table has been selected.

```
*)
ENTITY bundle_table
  SUPERTYPE OF (table_of_line_styles XOR
                table_of_surface_styles XOR
                table_of_geometric_aspects_of_text XOR
                table_of_text_styles XOR
                table_of_elementary_symbol_styles XOR
                table_of_area_pattern_styles);
  table_size    : INTEGER;
  open_table    : LIST [1 : table_size] OF NULL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

table_size: Maximum number of entries forseen.

open_table: Initializes the empty table.

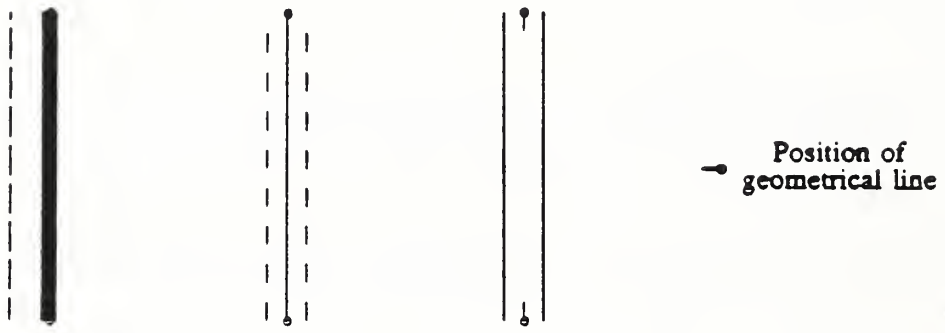
4.12.5 Table of Line Styles

Each entry of the active table of line styles defines a graphical appearance which may be attributed to any line. The line styles are grouped into categories, of which two are currently defined:

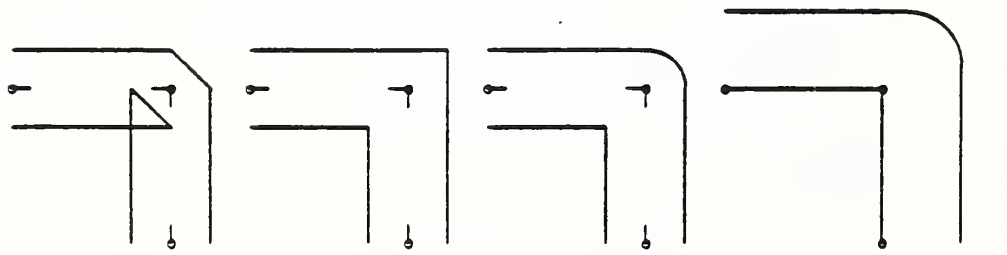
- Cat 1: Interrupted patterns
- Cat 2: Multiple offset lines

Figure 4 shows several aspects of line styles.

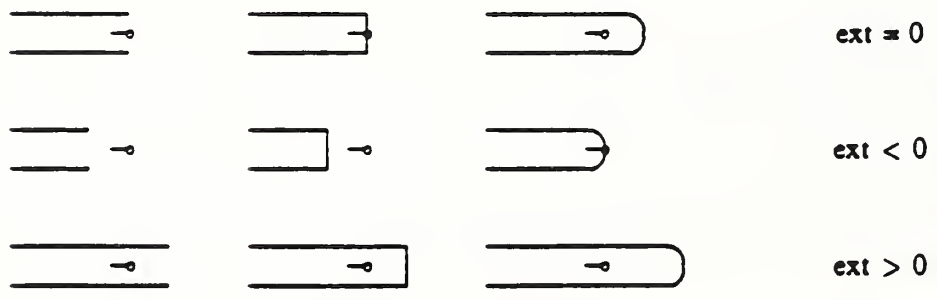
Figure 4: Variational possibilities of line styles.



Category 2 Line Styles (Multiple Lines)



Corner Handling (Category 2)



Line End Style and Extension

4.12.5.1 CATEGORY 1 LINE STYLE

This category covers the most frequently appearing simple line styles (Figure 4).
The line pattern is defined by an ordered list of number pairs. Each pair denotes the length of a visible line segment and the length of an invisible segment.
Examples are:

0, 1000000	invisible line
1000000, 0	visible line
2, 2	dashed line
3, 1.5, 0.5, 1.5	dashed-dotted line

```
*)
ENTITY category1_line_style;
  line_width   : REAL;
  line_pattern : LIST [2 : #] OF REAL;
  line_color   : rgb_color;
WHERE
  NOT ODD(SIZEOF(line_pattern));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

line_width: The width of the line in presentation units.

line_pattern: The definition of the line pattern.

line_color: The color.

PROPOSITIONS:

1. **Line pattern** must have an even number of entries.

4.12.5.2 CATAGORY 2 LINE STYLE

Category 2 covers the case of multiple offset lines and wide lines with line end options. It refers to **category1 line styles** for color, linewidth and line pattern (Figure 4).
The same line end style applies at both ends of a line. In case of different line widths of outer lines, the smaller of the two is used for the line end.

```
*)
ENTITY category2_line_style;
  multi_style  : LIST [1 : #] OF INTEGER;
  multi_offset : LIST [1 : #] OF REAL;
  corners      : OPTIONAL line_handling;
  ends         : OPTIONAL line_handling;
  extension    : OPTIONAL REAL;
WHERE
  coordinated_lists(multi_style, multi_offset);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

multi_style: List of references to category 1 entries in the same table of line styles.

multi_offset: List of offset values for each of the parallel lines. A positive value offsets to the right of the directed line, a negative value to the left.

corners: Control of the appearance of lines at corners. The values correspond to:

- **default**: No rules, implementation dependent.
- **squared**: Linear extension on convex side, intersecting parts removed on concave side.
- **rounded**: Circular rounding on convex side, intersecting parts removed on concave side.

ends: Control of the appearance of the ends of lines. The values correspond to:

- **default**: No line end handling, implementation dependent.
- **squared**: Squared line end.
- **rounded**: Rounded end line (semi circle). The circle extends beyond the line end.

extension: A negative value will shorten the line prior to applying the line end style, a positive value will extend it in the tangential direction (default 0.0).

4.12.5.2.1 LINE HANDLING

```
*)
TYPE line_handling = ENUMERATION OF
     (default,
      squared,
      rounded);
END_TYPE;
(*
```

4.12.5.3 ENTRY LINE STYLES

Line styles apply to lines derived from the product model as well as to lines appearing in annotations and tables.

Entries for a **table of line styles** can only occur at places where it is obvious which table is meant, i.e in the definition of a table and within a presentation block after a specific table has been selected.

```
*)
ENTITY entry_line_styles;
  table_index  : INTEGER;
  entry_choice : line_category_choice;
UNIQUE
  table_index;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

table_index: Position in table where entry shall be placed. This position must be unoccupied, i.e the value must be NULL.

entry_choice: Entering one attribute bundle of a particular category.

PROPOSITIONS:

1. Table index must be UNIQUE.

4.12.5.3.1 LINE CATEGORY CHOICE

```
*)
TYPE line_category_choice = SELECT
     (category1_line_style,
      category2_line_style);
END_TYPE;
(*
```

4.12.5.4 TABLE OF LINE STYLES

Each entry in a table of line styles defines a graphical appearance which may be attributed to any line.

```
*)
ENTITY table_of_line_styles
  SUBTYPE OF (bundle_table);
  initial_fill : OPTIONAL LIST [1 : #] OF entry_line_styles;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

initial_fill: Allows for filling in entries. The index value of the entry determines the position of the table to be filled.

4.12.6 Table of Surface Styles

The entries in the table of surface styles determine which lines of a surface are to be drawn and what the graphical appearance of each line shall be. These table entries obviously apply only to surface elements, whereby a distinction is required between CSG generated surfaces and Brep surfaces. For the latter a mapping exists from parameter space to 3D space and isoparametric lines may be of use in rendering the surface.

For version 1 of this Standard a simple form of shading is forseen in such a way to allow future extensions along the line of PHIGS+.

4.12.6.1 STYLE AND LAYER

Sinc the many optional entries for the table of surface styles each consist of a line style and a layer number increment, a separate entity is here defined for such a pair.

```
*)
ENTITY style_and_layer;
  style           : INTEGER;
  layer_increment : INTEGER;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

style: Refers to the table of line styles for the style of the corresponding class of lines.

layer_increment: This allows for the different classes of lines on a surface to be associated to different layers. A layer index is associated in the formation of the tree of geometry. The layer increments are added to this layer index to obtain the final layer number of the lines.

4.12.6.2 ENTRY SURFACE STYLES

Surface styles apply exclusively to surfaces derived from the product model.
Entries for a table of surface styles can only occur at places where it is obvious which table is meant. i.e in the definition of a table and within a presentation block after a specific table has been selected.
The OPTIONAL entries for styles and layer increments indicate which classes of lines are to be drawn. A missing attribute indicates that the corresponding class of lines shall not appear. The following abbreviations indicate to which syrface types a style may apply:

CSG: CSG surfaces.

Bez: Bezier surfaces (including rational).

Bsp: B-spline surfaces (including rational).

```
*)
ENTITY entry_surface_styles;
  table_index           : INTEGER;
  style_surface_bnd     : OPTIONAL style_and_layer;
  style_silhouette      : OPTIONAL style_and_layer;
  style_mult_knots      : OPTIONAL style_and_layer;
  style_patch_bnd       : OPTIONAL style_and_layer;
  style_isoparametric   : OPTIONAL style_and_layer;
  spacing_of_par_lines  : OPTIONAL REAL;
  style_patch_bnd_grid  : OPTIONAL style_and_layer;
  style_full_grid       : OPTIONAL style_and_layer;
  hidden_line_removal   : LOGICAL;
  shading_color         : OPTIONAL rgb_color;
  transparency          : OPTIONAL REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

table_index: Position in table of surface styles where the entry shall be placed. This index must not be greater than the table size and the position must be unoccupied i.e the value must be NULL.

style_surface_bnd: For the boundary lines of surfaces (CSG, Bez, Bsp).

style_silhouette: For silhouette lines within surfaces (CSG, Bez, Bsp).

style_mult_knots: For multiple knot isoparametric lines (Bsp).

style_patch_bnd: For single knot (Bsp) or patch boundary (Bez) isoparametric lines.

spacing_of_par_lines: Spacing of parametric lines in 3D space. The value indicates roughly the spacing between lines which should not be exceeded where their spacing is widest.

style_patch_bnd_grid: Grid polygons of patch boundaries only (Bez).

style_full_grid: The full grid (Bez, Bsp).

hidden_line_removal: Hidden lines for all the line styles present other than grid lines to be computed if TRUE.

shading_color: The value indicates the relative color proportions to be used from bright to dark for the presentation of this surface. Shading of a surface will apply only if this attribute is present.

transparency: The degree of transparency is indicated by a value between 0 and 1 giving the fraction of light which traverses the surface. Default (if value not present) is nontransparent. The value is irrelevant if no shading color is given.

4.12.6.3 TABLE OF SURFACE STYLES

The entries in the **table of surface styles** determine which lines of a surface are to be drawn and what the graphical appearance of each line shall be.

```
*)
ENTITY table_of_surface_styles
  SUBTYPE OF (bundle_table);
  initial_fill : OPTIONAL LIST [1 : #] OF entry_surface_styles;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

initial_fill: Allows for filling in entries. The index value of the entry determines the position of the table to be filled.

4.12.6.4 Table of Text Styles

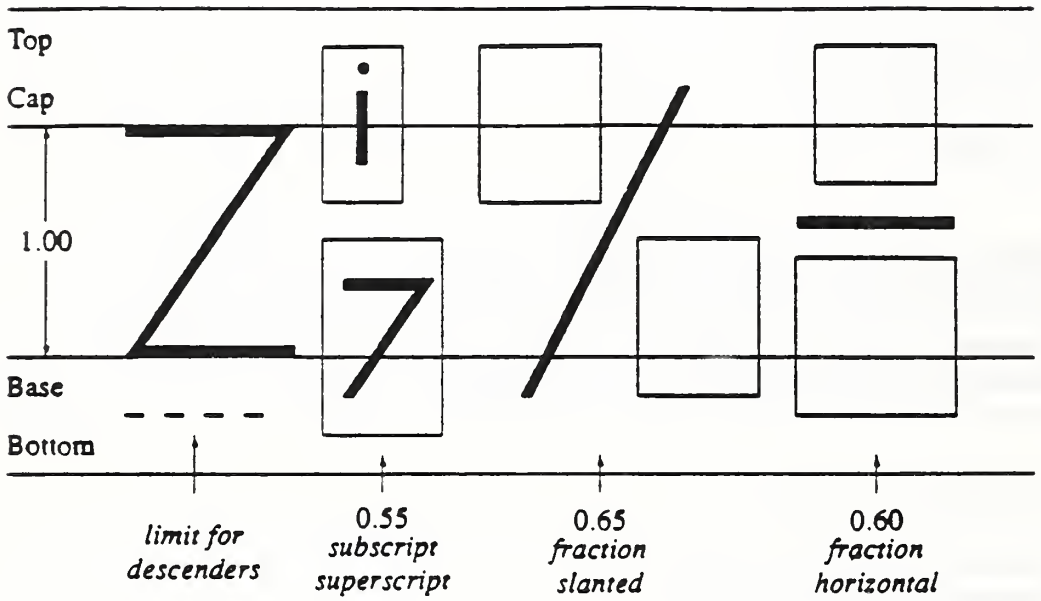
The **table of text styles** makes itself use of another table which allows for the control of many details of the geometry for the more sophisticared uses of text.

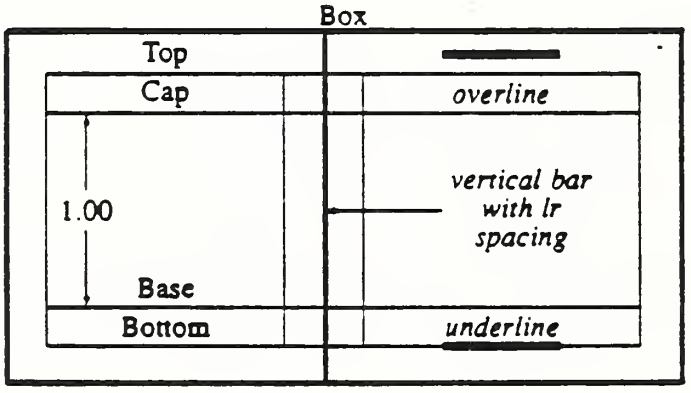
4.12.6.5 ENTRY GEOMETRIC ASPECTS OF TEXT

The **table of geometric aspects of text** controls aspects of the appearance of indices, exponents, fractions, underlines, overlines, boxes around the text and others. All the values of this entity are dimensionless, since they are given as proportional factors relative to the character height.

Figure 5 is helpful for the undestanding of the attributes. Approximate values corresponding to the figure are given as comments at the end of each attribute line in the definition below.

Figure 5: Geometric aspects of text





```
*)
ENTITY entry_geometric_aspects_of_text;
  size_super_subscript        : REAL;     --  0.55
  offset_superscript          : REAL;     --  0.70
  offset_midheight            : REAL;     --  0.20
  offset_subscript            : REAL;     --  -0.20
  size_fraction_slanted       : REAL;     --  0.65
  offset_numerator_slanted    : REAL;     --  0.60
  offset_denominator_slanted  : REAL;     --  -0.20
  size_fraction_horiz         : REAL;     --  0.60
  offset_numerator_horiz      : REAL;     --  0.65
  offset_denominator_horiz    : REAL;     --  -0.25
  position_of_underline       : REAL;     --  -0.40
  position_of_overline        : REAL;     --  1.40
  spacing_lr_of_vertical_bar  : REAL;     --  0.10
  box_margin_lr               : REAL;     --  0.25
  box_margin_bt               : REAL;     --  0.20
  blanking_margin_lr          : REAL;     --  0.20
  blanking_margin_bt          : REAL;     --  0.15
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

size_super_subscript: The size of the sub- and super- scripts depends on the selection of characters to be used. If the characters used in sub- and super- scripts include descenders the size must be smaller (e.g 0.55 as in Figure 5) than otherwise. If only numbers occur 0.60 may be possible as for horizontal fractions.

offset_superscript: Sets the position of the baseline for superscript characters.

offset_midheight: Sets the position of the baseline for midheight characters.

offset_subscript: Sets the position of the baseline for subscript characters.

size_fraction_slanted: In slanted fractions only digits occur and little or no gap is necessary. Therefore, the size may be larger than for horizontal fractions.

offset_numerator_slanted: Sets the baseline for the numerator.

offset_denominator_slanted: Sets the baseline for the denominator.

size_fraction_horiz: This size may correspond to the one for superscripts for digits only.

offset_numerator_horiz: Sets the baseline for the denominator.

offset_denominator_horiz: Sets the baseline for the demoninator.

position_of_underline: Defines the vertical position of an underline.

position_of_overline: Defines the vertical position of an overline.

spacing_lr_of_vertical_bar: If the vertical bar were treated as a character its character width would be twice this value.

box_margin_lr: The horizontal distance from the first or last character box to a vertical line of the surrounding box.

box_margin_bt: The vertical distance from the top resp. the bottom line to the surrounding box.

blanking_margin_lr: The margin for the blanking rectangle depends on whether a box exists or not. If no box is present, the distances are determined from the character box, otherwise the margins apply from the box outwards. This defines the left – right horizontal distances.

blanking_margin_bt: The margin for the blanking rectangle depends on whether a box exists or not. If no box is present, the distances are determined from the character box, otherwise the margins apply from the box outwards. This defines the bottom – top vertical distances.

4.12.6.6 TABLE OF GEOMETRIC ASPECTS OF TEXT

The **table of geometric aspects of text** controls aspects of the appearance of indices, exponents, fractions, underlines, overlines, boxes around the text and others.

```
*)
ENTITY table_of_geometric_aspects_of_text
  SUBTYPE OF (bundle_table);
  initial_fill : OPTIONAL LIST [1 : #] OF
                                  entry_geometric_aspects_of_text;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

initial_fill: Allows for filling in entries. The index value of the entry determines the position of the table to be filled.

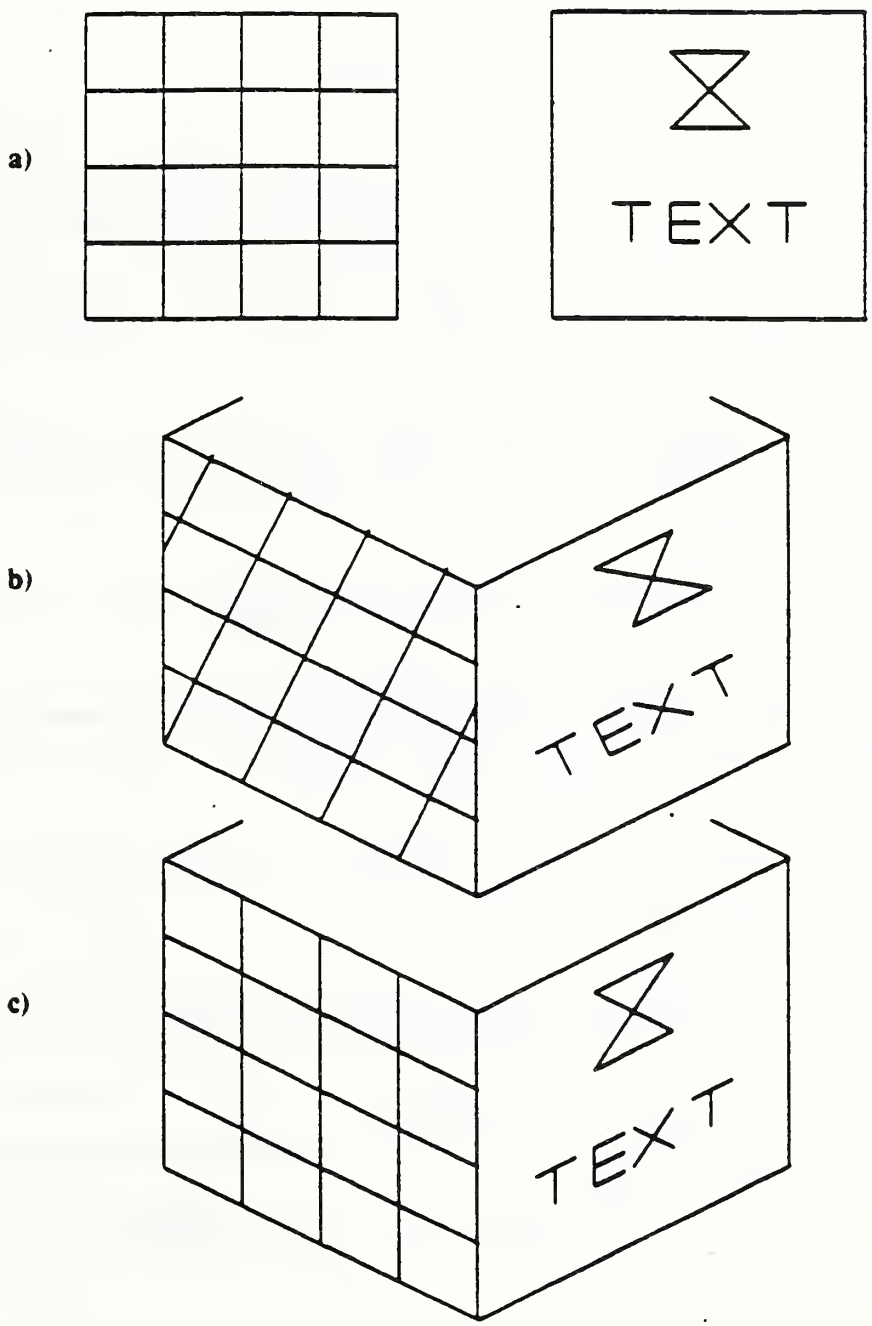
4.12.6.7 ENTRY TEXT STYLES

Each entry in the **table of text styles** describes a specific appearance of text by specifying its font, size, color and other characteristics.

Three dimensional text, symbols and patterns may be transformed (projected) as illustrated in Figure 6.

```
*)
ENTITY entry_text_styles;
  font_library                : character_font_library;
  font_identification         : character_font;
  line_width                  : OPTIONAL REAL;
  text_color                  : rgb_color;
  size_of_character           : REAL;
  angle_of_charup_vector      : REAL;
  char_expansion_factor       : REAL;
  additional_spacing_base     : REAL;
  text_path                   : simple_four_direction;
  additional_spacing_up       : OPTIONAL REAL;
  mirroring                   : mirroring_plane;
```

Figure 6: Text, elementary symbols and area pattern transformations: a) 2D versions of instances, b) 3D version with default transformation, c) 3D version with all geometry transformed.



```
  text_projection              : font_transformation;
  index_to_geometric_aspects   : INTEGER;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

font_library: Reference to a font library.

font_identification: Identifies a font within the library.

line_width: If present, overides the line width value in the library.

text_color: The color to be used.

size_of_character: Specifies the height of upper case characters. In an instance of text the characters are scaled from the size they have in the font definition to the size here defined.

angle_of_charup_vector: Specifies the angle in degrees between the baseline of the text and the up direction of characters. The values is 90 degrees for rectangular character boxes and less than 90 for slanted characters (italics).

char_expansion_factor: Controls the deviation of the character width from its normal value. A value larger than 1 signals expansion, a factor of less than 1 compression. This factor enlarges or reduces all dimensions in the direction of the base line by the same factor. This includes additional spacing base.

additional_spacing_base: Controls the spacing of characters. The value is given as a fraction of the font's nominal character height. This value is normally set to zero.

text_path: Specifies the writing direction for strings. Independent of the text path the first character is always at the same position but the following characters follow in the direction indicated. For the up and down values, the characters are arranged so that the centers of the character bodies are on a straight line in the direction of the up vector.

additional_spacing_up: This is of importance only for multiline text. If missing the default value is zero which means that the character boxes are vertically adjacent to each other.

mirroring: Specifies a mirroring plane.

text_projection: Of importance only for an instance of 3D text. It specifies the application of text transformation. The values have the following meaning, as illustrated in Figure 6:

- default: Only the position and base line direction are subject to transformation.
- all: The entire component is generated prior to a transformation i.e the composing geometry is subject to transformation.

index_to_geometric_aspects: Refers to the bundle of geometric specifications (see entry of geometric aspects of text).

4.12.6.7.1 SIMPLE FOUR DIRECTION

```
*)
TYPE simple_four_direction = ENUMERATION OF
      (right,
       up,
       left,
       down);
END_TYPE;
(*
```

4.12.6.7.2 MIRRORING PLANE

Defines planes about which an entity is mirrored.

```
*)
TYPE mirroring_plane = ENUMERATION OF
      (no_mirroring,
       yz_plane,
       xz_plane,
       xy_plane);
END_TYPE;
(*
```

4.12.6.7.3 FONT TRANSFORMATION

```
*)
TYPE font_transformation = ENUMERATION OF
      (default,
       all);
END_TYPE;
(*
```

4.12.6.8 TABLE OF TEXT STYLES

Each entry in the **table of text styles** describes a specific appearance of text by specifying its font, size, color and other characteristics.

```
*)
ENTITY table_of_text_styles
  SUBTYPE OF (bundle_table);
  initial_fill : OPTIONAL LIST [1 : #] OF entry_text_styles;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

initial_fill: Allows for filling in entries. The index value of the entry determines the position of the table to be filled.

4.12.6.9 ENTRY ELEMENTARY SYMBOL STYLES

Each entry in a **table of elementary symbol styles** describes the appearance of a class of symbols by specifying its font, size, color and other characteristics.

```
*)
ENTITY entry_elementary_symbol_styles;
  font_library                  : elementary_symbol_font_library;
  font_identification           : elementary_symbol_font;
  line_width                    : OPTIONAL REAL;
  symbol_color                  : rgb_color;
  size_of_symbol                : REAL;
  angle_of_symbol_up_direction  : REAL;
  mirroring                     : mirroring_plane;
  symbol_projection             : font_transformation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

font_library: Reference to a font library.

font_identification: Identifies a font within the library.

line_width: If present, overides the line width value in the library.

symbol_color: The color to be used.

size_of_symbol: Specifies the size of the symbol dependent dimension.

angle_of_symbol_up_direction: Specifies the angle in degrees between the baseline of the symbol and the up direction. The value is 90 degrees if no slant is required.

mirroring: Specifies a mirroring plane.

text_projection: Of importance only for an instance of 3D text. It specifies the application of text transformation.

4.12.6.10 TABLE OF ELEMENTARY SYMBOL STYLES

Each entry in a **table of elementary symbol styles** describes the appearance of a class of symbols by specifying its font, size, color and other characteristics.

```
*)
ENTITY table_of_elementary_symbol_styles
  SUBTYPE OF (bundle_table);
  initial_fill : OPTIONAL LIST [1:#] OF
                          entry_elementary_symbol_styles;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

initial_fill: Allows for filling in entries. The index value of the entry determines the position of the table to be filled.

4.12.6.11 Table of Area Pattern Styles

Each entry of the active **table of area pattern styles** defines a particular graphical appearance. The table is organized in a way to be open ended for future extensions. In particular, the introduction of categories of patterns (of which only two are used at this time) is a hint to future possibilities.

4.12.6.12 HATCHING

This category covers hatching with any line style available in the active line table, e.g with dotted lines.

```
*)
ENTITY hatching;
  line_style        : INTEGER;
  spacing_of_lines  : REAL;
  angle_of_lines    : REAL;
  relative_offset   : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

line_style: Refers to the active table of line styles.

spacing_of_lines: Distance between hatch lines in units of the **presentation sheet.**

angle_of_lines: Angle counterclockwise from the X axis in degrees.

relative_offset: Shifting of the hatch lines by a fraction of the spacing of lines. Relative offset is used when neighbouring areas are hatched with the same spacing and angle.

4.12.6.13 CATEGORY 2 AREA PATTERN STYLES

This category covers any area pattern style which may be generated by applying (category 1) hatchings. The definition occurs in a local coordinate system, in which the X axis becomes the base line.

```
*)
ENTITY category2_area_pattern_styles;
  line_styles     : LIST [1:#] OF INTEGER;
  start_of_lines  : LIST [1:#] OF cartesian_two_coordinate;
  shift_vectors   : LIST [1:#] OF two_space_direction;
  angle_of_lines  : LIST [1:#] OF REAL;
WHERE
  coordinated_lists(line_styles, start_of_lines);
  coordinated_lists(line_styles, shift_vectors);
  coordinated_lists(line_styles, angle_of_lines);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

line_styles: Each entry refers to the active table of line styles.

start_of_lines: The starting point of each line is also the initialisation of line patterns, which must be maintained even beyond gaps.

shift_vectors: The shift vectors control the spacing of lines as well as the initialisation of parallel lines.

angle_of_lines: Specifies the counterclockwise angles of the lines in degrees.

PROPOSITIONS:

1. All lists must be coordinated.

4.12.6.14 ENTRY AREA PATTERN STYLES

```
*)
ENTITY entry_area_pattern_styles;
  table_index          : INTEGER;
  entry_choice         : hatching_or_category2_pattern;
  pattern_projection : OPTIONAL font_transformation;
UNIQUE
  table_index;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

table_index: Position in table where entry shall be placed.

entry_choice: The type of area pattern style.

pattern_projection: Of importance only for an instance of a 3D pattern where it specifies the application of a transformation.

PROPOSITIONS:

1. Table index must be UNIQUE.

4.12.6.14.1 HATCHING OR CATEGORY 2 PATTERN

```
*)
TYPE hatching_or_category2_pattern = SELECT
      (hatching,
       category2_area_pattern_styles);
END_TYPE;
(*
```

4.12.6.15 TABLE OF AREA PATTERN STYLES

Each entry of a table of area pattern styles defines a particular graphical appearance.

```
*)
ENTITY table_of_area_pattern_styles
  SUBTYPE OF (bundle_table);
  initial_fill : OPTIONAL LIST [1:#] OF entry_area_pattern_styles;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

initial_fill: Allows for filling in entries. The index value of the entry determines the position of the table to be filled.

4.12.7 Instances

It is not obvious what kind of generality will be required for the placement of text, symbols etc. The following is a proposal (organised differently from IGES) which has a good logical structure and provides fair generality. Discussions will have to show to what extent these features are sufficient.

Although the data requirements for the 2D and 3D cases are closely related, separate entities are defined below. This will facilitate the building of subsets.

The 2D case is definitely the one encountered in the vast majority of annotations. Its data requirements are simpler to describe and to implement. It is therefore probable that a subset will be formed which would include only this case.

In the 3D case, the plane in which the annotation lies must be defined. This plane is normally defined by the point of the placement, the spatial direction of the base line and a second spatial direction in the plane, different from the first. The up Direction is determined by the angle given from the base line towards the second spatial direction. Futhermore, the geometry of the characters or symbols or of the area patterns may be formed at two different levels, either prior to entering the viewing pipeline or after, resulting in a different appearance, as shown in Figure 6.

Annotation items may be individually assigned a layer index. This is a positive integer number.

4.12.7.1 TEXT INSTANCE

Instances of text rely heavily on the table of text styles where the majority of the parameters are determined.

```
*)
ENTITY text_instance
  SUPERTYPE OF (instance_of_2d_text XOR
                instance_of_3d_text);
  text_string     : STRING;
  index_to_table  : INTEGER;
  layer           : INTEGER;
  text_length     : OPTIONAL REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

text_string: The text to be written, including escape sequences.

index_to_table: Specifies which entry of the table of text styles controls the appearance of the text.

layer: Indicates the layer index.

text_length: The maximum length on the base line which the text may assume. If the text (according to the specifications in the table) would be longer than this value, an overriding character expansion value less than 1 is applied. If the text length value is negative, the text will be right adjusted within the length if it is shorter than the absolute value.

4.12.7.2 INSTANCE OF 2D TEXT

An instance of 2D text.

```
*)
ENTITY instance_of_2d_text
  SUBTYPE OF (text_instance);
  position             : cartesian_two_coordinate;
  base_line_direction  : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

position: The placement of the left baseline point of the first character of the text.

base_line_direction: Counterclockwise angle in degrees from X axis to the base line.

4.12.7.3 INSTANCE OF 3D TEXT

An instance of 3D text.

```
*)
ENTITY instance_of_3d_text
  SUBTYPE OF (text_instance);
  position             : cartesian_three_coordinate;
  base_line_direction  : three_space_direction;
  second_direction     : three_space_direction;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

position: The placement of the left baseline point of the first character of the text.

base_line_direction: The direction of the base line.

second_direction: The second direction needed to define the plane of the text. The up direction will be determined by applying the angle of charup vector of the table of text styles from the base direction towards this direction.

4.12.7.4 ELEMENTARY SYMBOL INSTANCE

This entity places symbols from the elementary symbol font library on a view or sheet.

```
*)
ENTITY elementary_symbol_instance
  SUPERTYPE OF (instance_of_2d_elementary_symbol XOR
                instance_of_3d_elementary_symbol);
  symbol_ident   : elementary_symbol;
  index_to_table : INTEGER;
  layer          : INTEGER;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

symbol_ident: Specifies a specific symbol.

index_to_table: Specifies which entry of the table of elementary symbol styles shall control the appearance.

layer: The layer index.

4.12.7.5 INSTANCE OF 2D ELEMENTARY SYMBOL

An instance of a 2D elementary symbol.

```
*)
ENTITY instance_of_2d_elementary_symbol
  SUBTYPE OF (elementary_symbol_instance);
  position            : cartesian_two_coordinate;
  base_line_direction : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

position: The point where the origin of the symbol shall be located.

base_line_direction: Counterclockwise angle in degrees of the base line of the symbol with respect to the X axis.

4.12.7.6 INSTANCE OF 3D ELEMENTARY SYMBOL

An instance of a 3D elementary symbol.

```
*)
ENTITY instance_of_3d_elementary_symbol
  SUBTYPE OF (elementary_symbol_instance);
  position            : cartesian_three_coordinate;
  base_line_direction : three_space_direction;
  second_direction    : three_space_direction;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

position: The point where the origin of the symbol shall be located.

base_line_direction: A vector defining the base line in space.

second_direction: The second direction needed to define the plane of the symbol. The up direction will be determined by applying the angle of symbol up direction of the **table of elementary symbol styles** from the base direction towards this direction.

4.12.7.7 AREA PATTERN INSTANCE

Instances of area pattern styles rely heavily on the **table of area pattern styles** where most of the defining characteristics are stored.

```
*)
ENTITY area_pattern_instance
  SUPERTYPE OF (instance_of_2d_area_pattern_style XOR
                instance_of_3d_area_pattern_style);
  select_pattern : INTEGER;
  define_area    : pattern_fill_area;
  layer          : INTEGER;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

select_pattern: Index to the **table of area pattern styles**.

define_area: The area to be hatched or filled with the pattern. Blanking fields from annotation entities should be recognised automatically for subtraction from the area.

layer The layer index.

4.12.7.8 INSTANCE OF 2D AREA PATTERN STYLE

A 2D instance of an area pattern style.

```
*)
ENTITY instance_of_2d_area_pattern_style
  SUBTYPE OF (area_pattern_instance);
  ref_point_cat2 : OPTIONAL cartesian_two_coordinate;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

ref_point_cat2: Category 2 patterns require a reference point for alignment of the pattern with the boundary.

4.12.7.9 INSTANCE OF 3D AREA PATTERN STYLE

A 3D instance of an area pattern style.

```
*)
ENTITY instance_of_3d_area_pattern_style
  SUBTYPE OF (area_pattern_instance);
  ref_point            : cartesian_three_coordinate;
  base_line_direction  : three_space_direction;
  second_direction     : three_space_direction;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

ref_point: The point for the definition of the plane. At the same time it serves for the alignment of category 2 patterns.

base_line_direction: Part of the definition of the plane, and the base line for category 2 patterns. The pattern is rotated so that the X axis of the definition space corresponds to the base line direction.

second_direction: Part of the definition of the plane. The up direction, which is always 90 degrees from the base direction is determined in the half plane of this direction.

4.12.8 Composite Structures

4.12.8.1 Definition of Tables on Drawings

A table on a **presentation sheet** is generated in two steps, namely the table definition and the table instance. The table definition contains the static information which may remain unchanged over many uses of the table. The table instance calls for record formats which contain the information on the position and placement of the text to be filled into the table.

A table on a **presentation sheet** may be of variable width by having a repetitive part added repeatedly with an incremental shift. It may also be of variable length by adding a variable number of parts in another direction. In many cases a table is of fixed size, i.e all the variable parts are missing.

In the Western World, where normal text is written left to right, extyending the width of a table would normally mean an extension to the right and the variable part would in most cases add horizontal strips of fields at the bottom of the table. The mechanisms proposed leave the directions of growth entirely open in order that they should easily fit any requirement.

Text fields are specified to indicate where and how each text item shall appear in the table. Groups of fields form records. In the case of increments the number of fields per record increases with the number of increments added. The variable part of the table adds a number of records for each occurrence.

The definition of tables is fairly general and will allow the generation of a wide variety of tables including applications such as:

- title blocks (including logos),
- tables for bills of material,

- tables of changes,
- drawing frames.

4.12.8.2 GEOMETRY LINES

```
*)
ENTITY geometry_lines;
  line_style    : INTEGER;
  line_geometry : symbol_2d_curve;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

line_style: An index to the table of line styles defines the appearance of the lines described by line geometry.

line_geometry: Geometry consisting of a possibly interrupted sequence of straight line and circular segments.

4.12.8.3 FIELD DEFINITION

The field definition is a reservation of space for text to be filled in at an instance of the table.

```
*)
ENTITY field_definition;
  placement    : two_space_direction;
  text_style   : INTEGER;
  field_length : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

placement: The incremental placement of the left base line point of the first character of the field. The increments must be given relative to the previous field of the record. For the first field of a record, the increment is relative to the reference position of the record.

text_style: Specifies which entry of the table of text styles shall control the appearance of the text.

field_length: The field length indicates how much space is available for the text. If the text to be filled in would be longer than this, its character expansion factor must be reduced so that the text will fit. A negative value indicates that the text is to be right adjusted in the field.

4.12.8.4 RECORD DEFINITION

A record definition is a grouping of fields, defining either an absolute or relative reference point for the field definitions.

```
*)
ENTITY record_definition;
  reference_position : point_or_direction_2d;
  list_of_fields     : LIST [1:#] OF field_definition;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

reference_position: A position which serves as the starting point for the incremental positions of the fields. This position is normally specified through absolute coordinate values in the main part of the table, but must be given incrementally by specifying a vector in the variable part.

lisy_of_fields: If the table has variable width, the last field of the list of fields will be repeated as many times as needed.

4.12.8.4.1 POINT OR DIRECTION 2D

```
*)
TYPE point_or_direction_2d = SELECT
     (cartesian_two_coordinate,
      two_space_direction);
END_TYPE;
(*
```

4.12.8.5 TABLE ON DRAWING

The table on drawing has four parts labelled M, MH, V, VH in comments. Any allowable combination must contain the main part M. Out of the eight combinations remaining the following four are meaningful:

1. M;
2. M, V;
3. M, MH;
4. M, MH, V, VH;

Even in the main part all the lists are optional but at least one must appear. It is easy to imagine a table without the lines and header texts, consisting of the field definitions only.

```
*)
ENTITY table_on_drawing;
  -- Part M
  reference_point        : cartesian_two_coordinate;
  reference_mode         : table_extension;
  main_part_lines        : OPTIONAL LIST [1:#] OF geometry_lines;
  main_part_text         : OPTIONAL LIST [1:#] OF
                                    instance_of_2d_text;
  main_part_records      : OPTIONAL LIST [1:#] OF record_definition;
```

```
  -- Part MH
  main_increment          : OPTIONAL two_space_direction;
  main_extension_lines    : OPTIONAL LIST [1:#] OF geometry_lines;
  main_extension_text     : OPTIONAL LIST [1:#] OF
                                     instance_of_2d_text;

  -- Part V
  variable_increment      : OPTIONAL two_space_direction;
  variable_part_lines     : OPTIONAL LIST [1:#] OF geometry_lines;
  variable_part_text      : OPTIONAL LIST [1:#] OF
                                     instance_of_2d_text;
  variable_part_records   : OPTIONAL LIST [1:#] OF record_definition;

  -- Part VH
  variable_ext_increment : OPTIONAL two_space_direction;
  variable_ext_lines     : OPTIONAL LIST [1:#] OF geometry_lines;
  variable_ext_text      : OPTIONAL LIST [1:#] OF
                                     instance_of_2d_text;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

reference_point: The reference point which will later serve to place the table on the presentation sheet, possibly modified according to the reference mode.

reference_mode: This indicates whether and how the reference point shall be modified in case of variable width or length:

- None — No modification.
- Horizontal — Modification horizontal.
- Vertical — Modification vertical.
- Horizontal and vertical — Modification horizontal and vertical.

main_part_lines: The lines for the main part of the table.

main_part_text: Fixed text appearing in the main part, e.g as headers for columns.

main_part_records: The record definitions for the main part.

main_increment: Incremental change of local coordinate system for horizontal extensions.

main_extension_lines: The lines to be drawn for each extension.

main_extension_text: Identical text occuring in each of the extensions.

variable_increment: Incremental change of local coordinate system for vertical extensions.

variable_part_lines: The lines to be drawn for each vertical extension.

variable_part_text: Identical text occuring in each vertical extension.

variable_part_records: The records to be added for each vertical extension.

variable_ext.increment: Incremental change of the local coordinate system within a vertical extension. Should in almost all cases be equal to the main increment.

variable_ext_lines: The lines to be drawn for each horizontal extension within the vertical extension.

variable_ext_text: Identical text to be written into each of the horizontal extensions within the vertical extension.

4.12.8.5.1 TABLE EXTENSION

```
*)
TYPE table_extension = ENUMERATION OF
      (none,
       horizontal,
       vertical,
       horizontal_and_vertical);
END_TYPE;
(*
```

4.12.8.6 INSTANCE OF TABLE ON DRAWING

An **instance of table on drawing** consists of a previously defined table which is then positioned on a sheet and filled with data. All the data must be available as strings. In the two dimensional array, the inner list contains all the strings to fill one record, whereby the fields of the record are filled in the sequence of their definition. If fewer strings are available than corresponds to the record with the number of extensions, the last fields of the recortd may remain empty.

The outer list corresponds to the records defined for the table. The number of records must correspond to those of the main part of the table and, if applicable, must include the necessary records for the vertical extensions.

```
*)
ENTITY instance_of_table_on_drawing;
  choice_of_table        : table_on_drawing;
  position_on_sheet      : cartesian_two_coordinate;
  horizontal_extensions  : INTEGER;
  vertical_extensions    : INTEGER;
  texts                  : LIST [1:#] OF LIST [1:#] OF STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

choice_of_table: Reference to the definition of the table. The further data must correspond to the record specifications of the table on drawing.

position_on_sheet: Specifies where the table is to be placed.

horizontal_extensions: If horizontal extensions exist in the definition, this number indicates how many times they shall occur.

vertical_extensions: If vertical extensions exist in the definition, this number indicates how many times they shall occur.

texts: A two dimensional list of texts. The texts may use escape sequences as far as they appear meaningful. Mutiline texts are possible if space is available. The fields only describe text appearance and the horizontal width as the field length.

4.12.8.7 Definition of Composite Symbols

Composite symbols have many of the attributes of **elementary symbols**, but are more general with respect to the following characteristics:

- They may contain fixed texts as part of the symbol.
- They may contain any number of text fields defined exactly as in the definition of drawing tables.

4.12.9 Components of View with Annotation

The view with annotation is the basic unit of a presentation. It is complete with geometry and annotation and allows for an immense variety of the appearance of each of its many elements. Whereas all the annotation elements are generated within presentation or at least with the help of the basic entities of presentation, the situation is different for the geometry.

Although surfaces of objects have colors or even a very complex reflection behaviour of the surface, which might be stored somewhere in a file, different colors will generally be used for presentation purposes. Furthermore, in a sequence of pictures, colors or other attributes may change, e.g., in order to highlight specific portions for explanatory purposes.

4.12.9.1 The Tree Structure and Attributes

A view will generally show selected parts of a larger product model. These parts may have some structure, possibly hierarchical. In view of different application areas, e.g., finite elements, which have not specified their presentation needs yet, the ultimate list of entities to be presented remains open.

The geometry to be visualized is here specified in a tree structure (it was a linear list in the previous version) mainly for two reasons. First, a tree structure provides more flexibility in the association and change of attributes. Second, a growing number of systems (e.g., those based on PHIGS), have a segment structure which is a tree and are therefore capable to exploit the proposed entity structure directly.

4.12.9.2 ARBITRARY GEOMETRIC OBJECT

This entity (or type) which is currently left undefined, shall ultimately identify the geometric elements, i.e., curves, surfaces, CSG constructs or possibly other geometric items, which are to be presented in a picture, view or drawing. An arbitrary geometric object may here be anything from a circular segment to an assembly or a complete airplane, or any geometric items from applications areas (finite elements, schematic diagrams).

In IGES, a transformation is associated with every entity and visualization is not separated from the definition of the geometry. Here, definition and presentation are clearly separated. The need for visualization of objects separately defined in local coordinate systems hardly remains.

A transformation is nevertheless added below for those special cases where a transformation is desirable from a presentation point of view, e.g., for exploded views.

```
*)
ENTITY arbitrary_geometric_object;
  object     : undefined;
  placement  : OPTIONAL transformation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

object: Any entity defining geometry which may be visualized. Needs to be defined.

placement: A **transformation** which may move the **object** to a different place for presentation purposes only.

4.12.9.3 TREE OF GEOMETRY

This entity recursively defines a tree bottom up. It shall be a proper tree — each **arbitrary geometric object** or **tree of geometry** element shall occur only once in the entire tree structure.

Except for the leaves of the tree, each node has a reference which can be used to attach attributes. Attributes may be attached to:

- a node by giving its reference
- an element on the next lower level to a node (leaves are accessible only this way)
- a consequitive sequence of elements on the next lower level.

The association of attributes to a node implies that they become valid for the entire tree or subtree except for leaves or subtrees to which the corresponding attributes have been assigned at a lower level of the tree.

For objects which are represented by lines, the **table of line styles** is referenced. For surfaces it is the **surface related attributes** which references the **table of surface styles** which may further point to the **table of line styles** for the presentation of different types of surface lines.

If, at the time of the preparation of the visualization, no attribute exists at certain nodes or leaves of a tree, attributes are taken from presentation attributes for which default values do exist.

```
*)
ENTITY tree_of_geometry;
  node_of_tree          : LIST [1:#] OF UNIQUE
                               geometry_tree_component;
  layer_index           : OPTIONAL INTEGER;
  initial_line_attr     : OPTIONAL INTEGER;
  initial_surface_attr  : OPTIONAL surface_related_attributes;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

node_of_tree: This is an ordered list of the branches of this node, which may be leaves or subtrees. The sequence in the list is essential for subsequent changes of attributes.

layer_index: Indicates a layer number which shall be given to the geometry of the entire tree except for subtrees which have been given their own layer numbers. It should be observed that surfaces, depending on the entries in the **surface style table**, may occupy a range of layer numbers.

initial_line_attr: The same attributes will be assumed by all the curve elements of the tree except for subtrees which have been given their own line attributes. This is an index to the **table of line styles**, applicable only to the curve elements of the subtree.

initial_surface_attr: The same attributes will be assumed by all the surface elements of the tree except for subtrees which have been given their own surface attributes. **Surface related attributes** allows different styles for the exterior and interior of a surface.

4.12.9.3.1 GEOMETRY TREE COMPONENT

```
*)
TYPE geometry_tree_component = SELECT
      (arbitrary_geometric_object,
       tree_of_geometry);
END_TYPE;
(*
```

4.12.9.4 SURFACE RELATED ATTRIBUTES

The exterior and interior of a surface may appear in different colors or line styles or may be given different layer increments.

```
*)
ENTITY surface_related_attributes;
  exterior_style : INTEGER;
  interior_style : INTEGER;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

exterior_style: Index to the **table of surface styles** defining the intended appearance of the exterior of a surface.

interior_style: Index to the **table of surface styles** defining the intended appearance of the interior of a surface. Depending on the type of geometry, interior surfaces may be partially visible.

4.12.9.5 CHANGE GEOMETRY RELATED ATTRIBUTES

Within the hierarchy of presentation, trees of geometry may be defined at any level. Geometry related attributes may be assigned to each node of the tree at the same time. On subsequent higher levels, geometry related attributes may be changed. This change excludes the layer index, which is assumed to be stable.

If only the **geometry group** is given, the attributes are attached to the node else, if the **first index** is present but not the second, the attributes are attached to the branch indicated. If both indices are present, they specify an interval of branches to all of which the same attributes are attached.

Attributes may be attached at different levels (see *"The Hierarchy of Presentation"* or Figure 1). Any attributes changed at a higher level are valid for the entire subtree until control exits from this level to a lower one, when the previous values are unstacked and reappear. Redefinition at a higher level does not destroy the value at the lower level but covers it up to the end of the higher level.

```
*)
ENTITY change_geometry_related_attributes;
  geometry_group       : tree_of_geometry;
  first_index          : OPTIONAL INTEGER;
  second_index         : OPTIONAL INTEGER;
  line_attr_change     : OPTIONAL INTEGER;
  surface_attr_change  : OPTIONAL surface_related_attributes;
WHERE
  second_index > first_index;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

geometry_group: Defines a node of the **tree of geometry**. If the **first index** is NULL, the entire subtree is selected for the attribute change.

first_index: Selects a branch of this node by its ordinal number. If there is no **second index**, only the single branch is selected for an attribute change. This is meaningful only if the branch is a leaf, since otherwise it would be a node, which could have been addressed directly as a **geometry group**.

second_index: Determines another branch of this node by its ordinal number. An interval of consecutive branches from **first index** to **second index** is thus selected for attribute changes.

line_attr_change: An index to the **table of line styles** which is associated with the selected branch(es) or node. Applicable only to line items.

surface_attr_change: A reference to the surface style which is associated with the selected branch(es) or node. Applicable only to surface items.

4.12.9.6 LIGHT SOURCES

The entity for the definition of **light sources** shall allow for a lighting configuration consisting of

- ambient light,
- multiple spot light sources (at finite distance),
- multiple directional light sources (at infinity).

```
*)
ENTITY light_sources;
  attr : undefined;
END_ENTITY;
(*
```

4.12.9.7 VIEWING PIPELINE

In a transmissionof data between systems the human perception of the specification of facts is not of direct importance, instead, the equivalent system data, obtained after some preprocessing, may provide a better base for the specification in a file. An example of this is the indication of a frame size versus scale.

The distinction between a 3D and a 2D viewing pipeline is made because the latter is of near trivial simplicity, but is dominant in use, and is thus a candidate for a subset.

Figure 7 shows the sequence of operations of the viewing pipeline and its embedding in the process of producing annotated views. It may remain open whether scaling and clipping should be interchanged, which has an influence on the coordinate system applicable for the specification of the clipping planes.

```
*)
ENTITY viewing_pipeline
  SUPERTYPE OF (viewing_pipeline_2d XOR
                viewing_pipeline_3d);
  reference_point : cartesian_point;
  scale           : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

reference_point: This is a point in space, usually close to or within the geometry to be projected. The projection plane will pass through this point, which becomes the origin of the view coordinate system.

scale: The scale of the drawing as a multiplicative factor, i.e 0.01 means reduction by a factor of 100.

4.12.9.8 VIEWING PIPELINE 2D

A 2D projection ia a parallel projection in direction of the Z axis onto a projection plane lying in the XY plane. The X and Y axes will be projected onto parallels of the corresponding axes on the view plane.

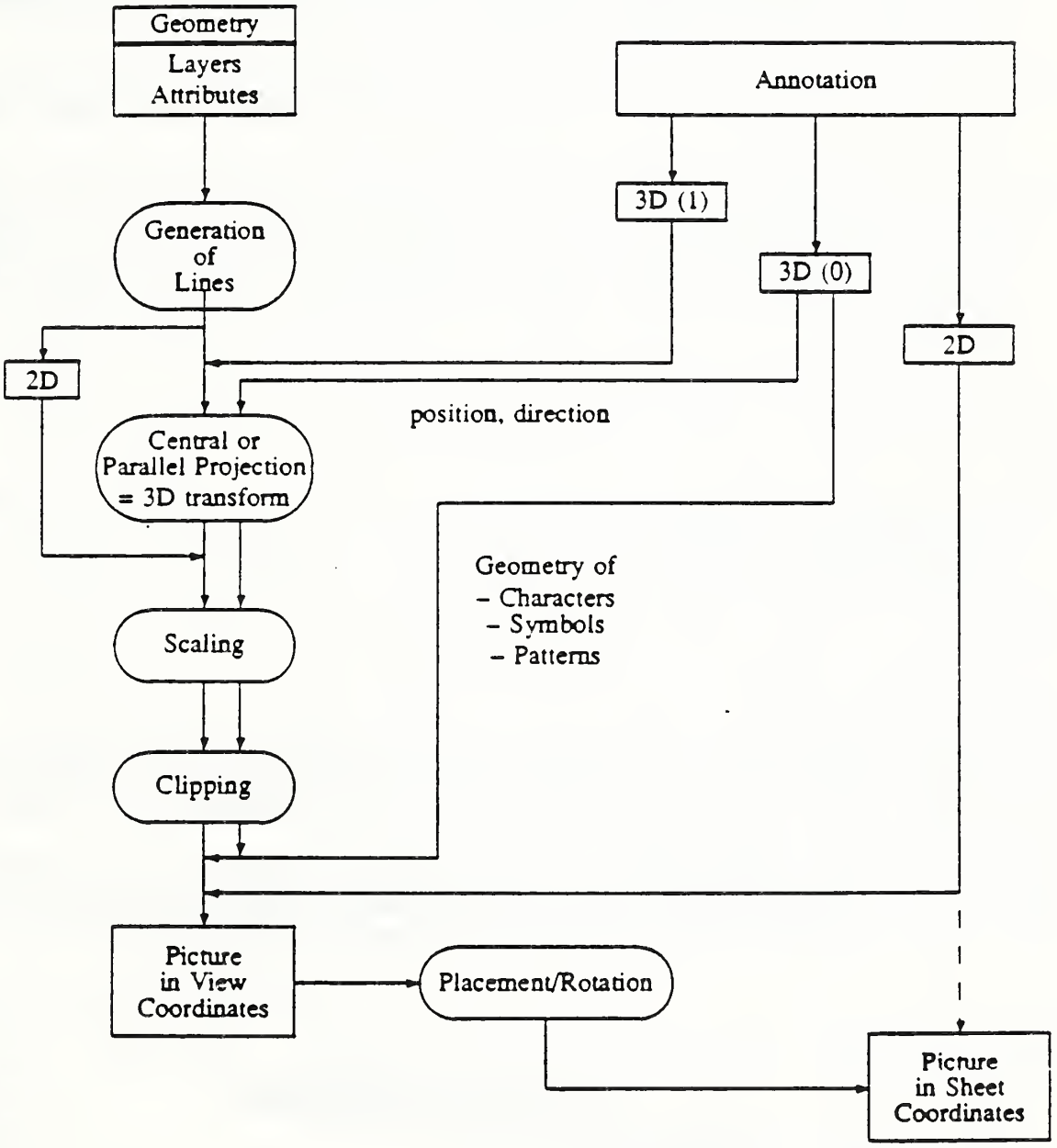
```
*)
ENTITY viewing_pipeline_2d
  SUBTYPE OF (viewing_pipeline);
END_ENTITY;
(*
```

4.12.9.9 VIEWING PIPELINE 3D

The **viewing pipeline 3d** defines the projection of geometrical items onto a viewplane. It allows for parallel or perspective projections and the optional tilting of the viewplane permits special effects, as, for example, keeping certain parallel lines in space parallel in their perspective projection.

```
*)
ENTITY viewing_pipeline_3d
```

Figure 7: Viewing pipeline for product model geometry and annotation.



```
  SUBTYPE OF (viewing_pipeline);
  projection         : point_or_direction_3d;
  view_up_vector     : three_space_direction;
  view_plane_normal  : OPTIONAL three_space_direction;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

projection: If the projection is given as a vector this implies parallel projection, the vector indicating the direction of the rays. Otherwise, perspective projection is assumed, whereby the point is the center from which the projecting rays depart.

view_up_vector: This vector is chosen such that a line in this direction from the reference point will be projected onto the Y axis of the view coordinate system for both parallel and perspective views.

view_plane_normal: If given, the view plane passing through the reference point will be normal to this vector. If NULL, the view plane normal is equal to the projection vector for parallel projection and equal to the vector pointing from the reference point to the projection point for perspective projection. The view plane normal need only ber given in order to achieve special effects.

4.12.9.9.1 POINT OR DIRECTION 3D

```
*)
TYPE point_or_direction_3d = SELECT
     (cartesian_three_coordinate,
      three_space_direction);
END_TYPE;
(*
```

4.12.9.10 CLIPPING

Clipping on the sides applies equally to 3D and 2D projections, but the front and back clipping planes are not needed for 2D projections. In accordance with the sequence of operations shown in Figure 7, clipping is indicated in view coordinates.

Many CAD systems have options for the user to provide a frame size instead of a scale, which is then computed such that all geometry after clipping will fit into the frame. Since the system then knows the scale it is not necessary to provide such a possibility within the viewing pipeline.

```
*)
ENTITY clipping;
  clipping_sides : OPTIONAL LIST [2:#] OF UNIQUE
                            cartesian_two_coordinate;
  clipping_front : OPTIONAL REAL;
  clipping_back  : OPTIONAL REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

clipping_sides: If two points are given, the rectangle formed parallel to the view coordinate axis is used for clipping, otherwise (more than two points) the closed polygon formed by these points, where the last point is connected to the first. The polygon must be convex. The coordinates refer to the view coordinate system. The clipping volume is formed by extending the rectangle or polygon to the front and back clipping planes.

clipping_front: The Z value in the view coordinate system of the front clipping plane. If NULL, the front clipping plane is assumed to lie immediately in front of the **projection point** for a perspective projection or in front of the closest part of geometry for a parallel projection.

clipping_back: The Z value in the view coordinate system of the back clipping plane. If NULL, the back clipping plane is assumed to lie far enough away that no clipping will occur.

4.12.10 The Hierarchy of Presentation

Figure 1 also shows the hierarchical structure of the high end entities of presentation.

presentation block It is asumed that drawings will not appear interspersed throughout a file, but will be collected as blocks which will follow the definition of the corresponding sections of geometry or of application entities. This will allow to set common property attributes at the level of the presentation block.

presentation set is a drawing which may consist of several sheets. Administrative data may be associated with drawings. In certain cases, when there is no need for this level in the hierarchy, it may be skipped.

presentation sheet A presentation sheet (or drawing sheet) is traditionally a piece of paper of a certain size which holds views tables and annotation. Views normally have well-defined scales and text and symbols appear in sizes determined by drafting standards to be easily legible.

It is suggested that the notion of presentation sheet be used even where presentations are to be viewed on screens of various sizes rather than on paper with the meaning that good viewing and legibility is ascertained if the screen size corresponds to the sheet size specified. In this way no distinction is necessary between views appearing in drawings and views to be seen on display screens (e.g. shadowing) while still allowing processing through existing graphics standards. A display then becomes a window with which we look at a sheet, and the size of the screen is unimportant from this point of view.

view with annotation This is the unit of presentation of geometry. It contains the collection of all the pieces which together form a drawing (selection of geometry, viewing transformation, all annotation and symbols and hatching.

These entities appear in the hierarchical order

```
level 0:  STEP File
level 1:          presentation block
level 2:                (presentation set)
level 3:                      presentation sheet
level 4:                            view with annotation
```

wherein the **presentation set** appears as an item which may optionally be skipped.

Geometry may be defined at all levels by recursively applying the entity **tree of geometry**

Attributes for presentation are split into general presentation attributes and geometry related attributes.

general presentation attributes is a collection of specifications which should usually remain unchanged for an entire presentation block and maybe for the entire file.

geometry related attributes are specifications attributed to items of the **tree of geometry**.

These two classes of attributes associate their specifications to the different levels of the hierarchy in a stack like fashion. At the lowest level, default attributes appear which become valid in the the absence of any redefinition. They may be redefined outside the **presentation block**, and on each of the levels shown above for the drafting hierarchy. A redefinition at a higher level never changes the one below it but hides it. If not redefined, the attribute is inherited from the lower level.

Ultimately, within the entities **presentation sheet** and **view with annotation** use will be made of the attribute values. For any attribute needed, the appropriate value is determined by checking from the higher to the lower levels until it is defined for the first time.

4.12.10.1 Changing Geometry Related Attributes at Different Levels

Geometry related attributes may be set or changed on the level on which the geometry is defined and changed at any higher level.

On each level, attributes which are set or changed for a node are valid for the entire subtree below that node except for assignments given to subtrees or leaves below that node. The sequence od assignments is of no importance. Since only the leaves of the tree carry geometrical information, attributes have meaning only for the leaves. The attribute for a leaf is determined by the following steps.

1. If an attribute has been assigned to the leaf, then exit, else goto point 2.
2. Move to the parent node of the current leaf or node.
3. If an attribute has been assigned to this node, then exit, else if this node is not the root goto point 2, else goto point 4.
4. Take attribute from the general presentation attributes.

Changes of attributes at a higher level define a new plane of attribute values on which exactly the same rules hold. At each level none, a few or all of the attributes may be redefined. From the point of view of a **view with annotation**, attribute values may appear on up to four different planes. In order to determine the valid attributes, the algorithm then needs a slight change.

- It shall start at the plane of attribute values of level 4 (**view with annotation**).
- Whenever a root node is reached without encountering the attribute, restart at the plane of the next lower level. The **general presentation attributes** would correspond to the plane at level 0.

This structure applies separately for each of the different attributes which may be assigned. Of course, this would be an inefficient way of implementing an algorithm, but it may help to understand the mechanism.

4.12.10.2 GENERAL PRESENTATION ATTRIBUTES

The general attributes related to presentation may be used at different levels of the entity hierarchy whereby attributes defined at a lower level act as default levels for the higher levels. Redefinition at a higher level does not invalidate the value at the lower level. This holds separately for each attribute.

General presentation attributes cover a number of parameters of presentation and drafting which usually remain constant over an entire **presentation block** or at least part of it. The list given in the entity below is probably far from complete. However, it establishes the principle, whereafter extensions are easily added.

ATTRIBUTE	DEFAULT	COMMENTS
sheet size	(1189, 841) mm	Format A0
curve viewing tolerance	0.1 mm	
surface viewing tolerance	0.1 mm	

Table 12: Default Presentation Values

General presentation attributes are not associated with any particular element but are values which hold for entire views drawings or all elements of a **presentation block**.

All attributes in this entity are optional, which allows the redefinition of selected attributes in the hierarchy of drafting entities.

All the attributes possess standard default values which are used individually for each attribute in the absence of redefinitions. Wherever possible, the default values are taken from the ISO Drafting Standard. Table 12 gives the defined defaults used here.

The parameters of type REAL and **cartesian two coordinate** specify values in units as defined by **sheet linear units**.

```
*)
ENTITY general_presentation_attributes;
  sheet_size                  : OPTIONAL coordinate_pair;
  curve_viewing_tolerance     : OPTIONAL REAL;
  surface_viewing_tolerance   : OPTIONAL REAL;
  default_curve_style         : OPTIONAL INTEGER;
  default_surface_appearance  : OPTIONAL surface_related_attributes;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sheet_size: Indicates the size of the paper sheet to be used in sheet linear units (Default: Format A0 = 1189 by 841 mm).

curve_viewing_tolerance: Indicates the maximum deviation allowable in case of approximation of projected curves by straight line segments, given in sheet linear units. (Default: 0.1mm).

surface_viewing_tolerance: Indicates the maximum deviation allowable in case of approximation of surfaces by facets for shadowed pictures, given in sheet linear units. (Default: 0.1mm).

default_curve_style: Index to the **table of line styles**.

default_surface_appearance: Reference to specification for the appearance of surfaces.

4.12.10.3 VIEW WITH ANNOTATION

The **view with annotation** is the unit of presentation. It collects all the elements which may occur together in a picture and defines projection, clipping and position on a sheet.

```
*)
ENTITY view_with_annotation;
  view_presentation_attr     : OPTIONAL
                                 general_presentation_attributes;
  selected_geometry          : tree_of_geometry;
  view_level_geometric_attr  : OPTIONAL LIST [1:#] OF
                               change_geometry_related_attributes;
  visible_layers             : OPTIONAL LIST [1:#] OF INTEGER;
  incident_light             : OPTIONAL light_sources;
  projection_geometry        : viewing_pipeline;
  view_limitation            : clipping;
  dimensions                 : OPTIONAL LIST [1:#] OF
                                        any_dimensional_annotation;
  other_annotation           : OPTIONAL LIST [1:#] OF
                                        any_other_annotation;
  area_fill                  : OPTIONAL LIST [1:#] OF
                                        area_pattern_styles;
  rotation_on_sheet          : OPTIONAL REAL;
  placement_on_sheet         : cartesian_two_coordinate;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

view_presentation_attr: Allows for changes of **presentation attributes** for this view only.

selected_geometry: Defines which tree of geometric objects may be visualized in this view if they are in the proper layer and are not lost in clipping.

view_level_geometric_attr: Allows changes of attributes assigned to geometry. Such changes are valid for this view only and are therefore restricted to the tree or subtrees of the **selected geometry**.

visible_layers: Each geometric object may be given a layer number at the time when the **tree of geometry** is composed. This attribute selects the layers such that only objects in these layers will be shown.

incident_light: The indication of **light sources** is necessary if, among the surfaces of the **selected geometry**, there is at least one for which a shading color is indicated in its associated entry in the **table of surface styles**.

projection_geometry: Selection of a **viewing pipeline**. The viewing pipeline applies to the geometrical items which, especially for the case of surfaces have to be extracted computationally from the format used for storage. It further applies to part of annotation.

view_limitation: Clipping with a rectangular region or with convex polygon. For 3D also clipping with front and back plane.

dimensions: The dimensional annotation entities appearing in this list are to be placed in relation to the geometric objects.

other_annotation: These annotation items may be related to geometry.

area_fill: Any regions to be hatched or filled with patterns.

rotation_on_sheet: Optionally, the placement of a view **with annotation** on a sheet may involve a rotation, the angle of which is given here in degrees. The rotation around the origin of the view coordinate system is executed prior to the placement and involves every single line of the view.

placement_on_sheet: This is a translation whereby the origin of the view coordinate system is moved to the sheet coordinates indicated. It places the entire view, annotations and everything at the desired position on the drawing sheet.

4.12.10.4 ANY DIMENSIONAL ANNOTATION

Required by **view with annotation.**

```
*)
ENTITY any_dimensional_annotation;
  attr : undefined;
END_ENTITY;
(*
```

4.12.10.5 ANY OTHER ANNOTATION

Required by **view with annotation.**

```
*)
ENTITY any_other_annotation;
  attr : undefined;
END_ENTITY;
(*
```

4.12.10.6 PRESENTATION SHEET

A **presentation sheet** (drawing sheet) may contain any number of **views with annotation** and/or **tables on drawing** with additional annotation in sheet coordinates. Attribute changes made on the sheet level (level 3) are valid for all the **views with annotation** appearing on the sheet.

```
*)
ENTITY presentation_sheet;
  sheet_presentation_attr     : OPTIONAL
                                 general_presentation_attributes;
  sheet_level_geometric_attr : OPTIONAL LIST [1:#] OF
                                 change_geometry_related_attributes;
  sheet_level_geometry        : OPTIONAL LIST [1:#] OF
                                 tree_of_geometry;
  positioned_views            : OPTIONAL LIST [1:#] OF
                                 view_with_annotation;
  positioned_tables           : OPTIONAL LIST [1:#] OF
                                 instance_of_table_on_drawing;
  annotation_on_sheet         : OPTIONAL LIST [1:#] OF
```

```
                                    any_other_annotation;
  sheet_admin_data              : undefined;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sheet_presentation_attr: Allows for changes of the **general presentation attributes** valid for this sheet only.

sheet_level_geometric_attr: Allows changes of attribute associations to items of any **tree of geometry** defined at a lower level. The changes are valid only for this sheet.

sheet_leve:_geometry: Definition of **trees of geometry** with associated attributes for use in views.

positioned_views: List of the views at their proper positions. In rare cases, a sheet may contain no views, i.e., only tables and annotation.

positioned_tables: The list of tables. It includes the title block and the frame around the sheet.

annotation_on_sheet: For annotation of this list, the sheet coordinates are relevant. Such annotation has no geometrical relationship to any of the views.

sheet_admin_data: may contain data of the following items (and is being defined by drafting)

- sheet number/title/classification
- sheet identification/status/date
- security classification
- sheet revision log
- sheet checking and approval

4.12.10.7 PRESENTATION SET

The **presentation set** is a logical grouping of **presentation sheets**, applicable when views and tables belong together but cannot fit on a single sheet. Changes of attributes are also possible on this level (level 2).

```
*)
ENTITY presentation_set;
  set_presentation_attr     : OPTIONAL
                                general_presentation_attributes;
  set_level_geometric_attr  : OPTIONAL LIST [1:#] OF
                                change_geometry_related_attributes;
  set_level_geometry        : OPTIONAL LIST [1:#] OF
                                tree_of_geometry;
  collection_of_sheets      : OPTIONAL LIST [1:#] OF
                                presentation_sheet;
  set_admin_data            : OPTIONAL undefined;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

set_presentation_attr: Allows for changes of **general presentation attributes** valid for this set only.

set_level_geometric_attr: Allows changes of attribute association to items of any **tree of geometry**. The changes are valid only for this set.

set_level_geometry: Definition of **trees of geometry** with associated attributes for use at the higher levels of the sheets and views.

collection_of_sheets: All the **presentation sheets** which together form the drawing.

drawing_admin_data: May contain data of the following types (and is being defined by drafting)

- drawing number/title/classification
- drawing identification/status/date
- security classification

4.12.10.8 PRESENTATION BLOCK

The **presentation block** is primarily a scoping entity which allows the setting of an environment of a number of sheets or sets. In particular, it determines the set of active tables.

```
*)
ENTITY presentation_block;
  sheet_linear_units            : length_unit;
  activate_line_styles          : table_of_line_styles;
  activate_surface_styles       : table_of_surface_styles;
  activate_geometric_aspects    : table_of_geometric_aspects_of_text;
  activate_test_styles          : table_of_text_styles;
  activate_symbol_styles        : table_of_elementary_symbol_styles;
  activate_area_pattern_styles  : table_of_area_pattern_styles;
  block_presentation_attr       : OPTIONAL
                                    general_presentation_attributes;
  block_level_geometric_attr    : OPTIONAL LIST [1:#] OF
                                  change_geometry_related_attributes;
  block_level_geometry          : OPTIONAL LIST [1:#] OF
                                    tree_of_geometry;
  sheets_and_sets               : OPTIONAL LIST [1:#] OF
                                    sheet_or_set;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sheet_linear_units: Indicates in which units dimensions on the drawing sheet and character sizes etc. are measured.

activate_line_styles: Select a **table of line styles**. Activation selects the one table to be used within this **presentation block**. Multiple tables of the same kind may exist and different choices may be made in different **presentation blocks**.

activate_surface_styles: Select a table of surface styles.

activate_geometric_aspects: Select a table of geometric aspects of text.

activate_text_styles: Select a table of text styles.

activate_symbol_styles: Select a table of elementary symbol styles.

activate_area_pattern_styles: Select a table of area pattern styles.

block_presentation_attr: Allows for changes of general presentation attributes valid for this block only.

block_level_geometric_attr: Allows for changes of attribute associations to items of any tree of geometry. The changes are valid for this block only.

block_level_geometry: Definition of trees of geometry with associated attributes for use at the higher levels of the sheets and views.

sheets_and_sets: Within a presentation block, presentation sets and presentation sheets may occur mixed.

4.12.10.8.1 SHEET OR SET

```
*)
TYPE sheet_or_set = SELECT
     (presentation_sheet,
      presentation_set);
END_TYPE;
(*
```

4.12.11 Presentation Entities Classification

The following indented list provides the entity classification structure for the Presentation Model.

```
ANY DIMENSIONAL ANNOTATION
ANY OTHER ANNOTATION
ARBITRARY GEOMETRIC OBJECT
AREA PATTERN INSTANCE
  INSTANCE OF 2D AREA PATTERN STYLE
  INSTANCE OF 3D AREA PATTERN STYLE
BUNDLE TABLE
  TABLE OF AREA PATTERN STYLES
  TABLE OF ELEMENTARY SYMBOL STYLES
  TABLE OF GEOMETRIC ASPECTS OF TEXT
  TABLE OF LINE STYLES
  TABLE OF SURFACE STYLES
  TABLE OF TEXT STYLES
CATEGORY1 LINE STYLE
CATEGORY2 AREA PATTERN STYLES
CATEGORY2 LINE STYLE
CHANGE GEOMETRY RELATED ATTRIBUTES
```

```
CHARACTER FONT
CHARACTER FONT LIBRARY
CLIPPING
ELEMENTARY SYMBOL
ELEMENTARY SYMBOL FONT
ELEMENTARY SYMBOL FONT LIBRARY
ELEMENTARY SYMBOL INSTANCE
  INSTANCE OF 2D ELEMENTARY SYMBOL
  INSTANCE OF 3D ELEMENTARY SYMBOL
ENTRY AREA PATTERN STYLES
ENTRY ELEMENTARY SYMBOL STYLES
ENTRY GEOMETRIC ASPECTS OF TEXT
ENTRY LINE STYLES
ENTRY SURFACE STYLES
ENTRY TEXT STYLES
FIELD DEFINITION
FILLED SYMBOL AREA
GENERAL PRESENTATION ATTRIBUTES
GEOMETRY LINES
HATCHING
INSTANCE OF TABLE ON DRAWING
LIGHT SOURCES
PATTERN FILL AREA
PRESENTATION BLOCK
PRESENTATION SET
PRESENTATION SHEET
RECORD DEFINITION
RGB COLOR
SINGLE CHARACTER
STYLE AND LAYER
SURFACE RELATED ATTRIBUTES
SYMBOL 2D CURVE
TABLE ON DRAWING
TEXT INSTANCE
  INSTANCE OF 2D TEXT
  INSTANCE OF 3D TEXT
TREE OF GEOMETRY
VIEWING PIPELINE
  VIEWING PIPELINE 2D
  VIEWING PIPELINE 3D
VIEW WITH ANNOTATION

*)
  END_SCHEMA;  -- end of PRESENTATION schema
(*
```

4.12.12 Resource Schema End

This ends the IPIM Resource schema.

```
*)
  END_SCHEMA; -- end RESOURCE schema
(*
```

4.13 Product Life Cycle

This is Part 2 of the Integrated Product Information Model and will contain the Product Life Cycle models.

```
*)
SCHEMA ipim_life_cycle_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_resources_schema,
         ipim_applications_schema);

END_SCHEMA; -- end LIFE CYCLE schema
(*
```

4.14 Applications

This third part defines the Application models for the Integrated Product Information Model. These are presently limited to Drafting, Product Structure Configuration Management, AEC, Ship Structures, Electical Functional, Electrical Schematics, Layered Electrical Product, FEM and Data Transfer Applications.

```
*)
SCHEMA ipim_applications_schema;

  EXPORT EVERYTHING;
  ASSUME(ipim_product_manifestation_schema,
         ipim_mechanical_product_schema,
         ipim_aec_schema,
         ipim_electrical_schema,
         ipim_analysis_schema,
         ipim_data_transfer_schema);
(*
```

4.15 Product Manifestation

Collected here are the Application Schemas for Product Manifestation.

```
*)
SCHEMA ipim_product_manifestation_schema;
  EXPORT EVERYTHING;
  ASSUME(ipim_resources_schema,
         ipim_drafting_schema);
(*
```

4.15.1 Drafting

```
*)
SCHEMA ipim_drafting_schema;

EXPORT EVERYTHING;

ASSUME (ipim_shape_interface_schema,
          ipim_features_schema,
          ipim_geometry_schema,
          ipim_tolerances_schema,
          ipim_resources_schema);
(*
```

4.15.1.1 Introduction

Traditionally Product Definition Data (PDD) is expressed via drawings created according to a drafting standard. Although many manufacturers now create drawings using Computer Aided Design (CAD), they still consider their product to be represented by its drawings rather than by the computer model which generated the drawing. This has been necessary since the computer models have not yet been able to support all of the product information conveyed on the drawing.

The Drafting Model defines the data relationships which enable exchange of a human interpretable representation of PDD, known as a drawing. The model must support a mix of two levels of exchange. The first level is exchange of the "physical" or "explicit" representation and the second level of exchange is "semantic content" or "logical" with parameters required to generate the representations. Drafting data exchange must take place at the highest level of information possible and evolutionary development from lower to higher levels of exchange is inevitable. Supporting a mixed range of capabilities complicates the model significantly but is felt to be the only way to support current data exchange and the evolution to a more informational level.

One simple example of the levels of data exchange is found in the exchange of area sectioning data. A physical level of exchange is included in the entity **section representation** as a set of line segments: however, the entity includes optional pointers to **section parameters** and **sectioned area association** to provide generative parameters and associations to the product shape, respectively. Exchange at the physical level provides no information, only a representation. Exchanging the generative parameters and associative information is a higher level of exchange which provides all of the information required to support regeneration of the representation. The difference is that the higher level of information is often more compact and provides the receiver with the information to support maintenance of the drawing.

The model incorporates much of the work of many participants in the Drafting effort. Modeling has developed from numerous international viewpoints on both general and specific aspects of the model. Integrating the work of all viewpoints has not been totally completed yet, but the model does accommodate the multiple viewpoints and areas of interest.

4.15.1.2 Known Shortcomings

The model has not yet been integrated with resource models to any significant extent. A few examples of how we feel we integrate with resource models are shown; the most complete is in the area of geometric tolerances. No integration with presentation has taken place. We felt that model development should not be constrained by attempting integration simultaneously with extension.

The area of dimension representations is very weak as we have a modeling subgroup doing extensive work in this area but weren't able to integrate in their work for this version. This area is essentially an incorporation of the corresponding IGES constructs.

Many of the "association" entities which provide the links between PDD representations and the resource information are undefined and serve merely as placeholders.

Many of the representations are still a mixture of generative information and pure representation. These have not yet been sorted into the general structures of representation, properties and association.

4.15.1.3 Detailed Example of Mapping

The details of high-level data exchange between the resource, logical, and physical levels of the model, and the associated mapping functions, can be demonstrated with geometric tolerance information. In the following paragraphs two examples of the details of the mappings and the response of the model to low-level data exchange will be presented.

The resource information in the TOLERANCE model is mapped to logical information in the **geometric tolerance association** via the **gtol map** mapping function. The logical information is mapped to the physical representaion of the information in **feature control frame** via the **feature control frame map** mapping fuction.

When a tolerance is specified in cylindrical form **gtol map** transfers the real number value of the **tol dia form mag** from the TOLERANCE model to **geometric tolerance association. Feature control frame map** then outputs both the maximum tolerance value, as a string, and the diameter symbol identifying string to the **tol specification compartment** in the **feature control frame. In feature control frame map**, the **real to char** mapping function converts the tolerance value from a real number value to a string; the diameter symbol identifying string is derived from the location of the value in the TOLERANCE model **cylindricity**, **position**, etc.).

When a material condition is applied to a tolerance the information is again mapped to the **tol specification compartment** in the **feature control frame** from the TOLERANCE model using the **gtol map** and **feature control frame map** mapping functions. The real number tolerance value will again be converted to a string using the **real to char** function; in this instance the identifying string for the material condition symbol is retrieved from the **tol mlsn** entity in the TOLERANCE model with the **mlsn map** mapping fuction.

To accommodate low level, representation based, drafting data the **feature control frame map** mapping function can be deleted by omitting the **feature control frame.assoc** link to **geometric tolerance association**. Since the **feature control frame** is a physical entity it can stand alone without a link to product data.

4.15.1.4 Drafting TYPE Definitions

4.15.1.4.1 DRAFTING STANDARD

The standards which control the interpretation of the **drawing**.

```
*)
TYPE drafting_standard = ENUMERATION OF
          (iso,
          afnor,
          ansi,
          bsi,
          csa,
```

```
          din,
          jis,
          user_convention);
END_TYPE;
(*
```

4.15.1.4.2 DRAWING STATUS

The status of the drawing.

```
*)
TYPE drawing_status = ENUMERATION OF
    (working,
     approved,
     issued,
     check);
END_TYPE;
(*
```

4.15.1.4.3 DRAWING TYPE

The type of drawing representation.

```
*)
TYPE drawing_type = ENUMERATION OF
    (hand_sketch,
     single_detail,
     multi_detail,
     assembly,
     layout);
END_TYPE;
(*
```

4.15.1.4.4 MODEL VIEW APPEARANCE

The high level conveyance of hidden edge representations.

```
*)
TYPE model_view_appearance = ENUMERATION OF
      (hidden_edges_dashed,
       hidden_edges_not_shown);
END_TYPE;
(*
```

4.15.1.4.5 STD SHEET SIZE

The standard sizes for a drawing sheet.

```
*)
```

```
TYPE std_sheet_size = ENUMERATION OF
     (A, B, C, D, E, F, G, H, J, K,
      A0, A1, A2, A3, A4);
END_TYPE;
(*
```

4.15.1.5 ANNOTATION

Any text or graphics which is not product geometry. **Annotation** is used for documenting product features, defining product characteristics, aiding drawing administration, and for facilitating the reading of the drawing.

```
*)
ENTITY annotation
  SUPERTYPE OF (annotation_geometry XOR
                annotation_subfigure XOR
                note XOR
                product_annotation XOR
                relation_annotation XOR
                symbol);
  origin           : OPTIONAL cartesian_two_coordinate; --resource
  depth            : OPTIONAL depth;
  view_association : OPTIONAL drawing_view;
  sheet            : OPTIONAL drawing_sheet;
  appearance       : OPTIONAL appearance_specification;
  box_height       : OPTIONAL REAL;
  box_width        : OPTIONAL REAL;
WHERE
  NOT (depth <> NULL and view_association = NULL);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

origin: The primary location of the **annotation**.

depth: An entity whose existence denotes definition in model space, and whose value provides the z coordinate for the **origin**. Without this entity, the **annotation** is defined in two-dimensional sheet space.

view_association: A view the **annotation** is associated with. For **annotation** defined in model space, this associated view determines the orientation of the text according to the view transform.

sheet: The sheet on which the **annotation** is defined.

appearance: The appearance of the **annotation**. This is OPTIONAL since default appearance is set by **drawing**, and optionally set by **drawing sheet** and **drawing view**.

box_height: The height of a box which bounds the **annotation**. The box is not necessarily filled.

box_width: The width of a box which bounds the **annotation**.

PROPOSITIONS:

1. The depth cannot exist and have the view association be null.

4.15.1.6 ANNOTATION GEOMETRY

A type of annotation which encompasses the lines and curves required to annotate the drawing. Typically excludes the entities used to describe the geometry of the part.

```
*)
ENTITY annotation_geometry
  SUBTYPE OF (annotation);
  curve      : SET [1 : ?] OF curve; --resource
  appearance : OPTIONAL curve_appearance;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

curve: The geometry used in the annotation.

appearance: The appearance parameters for the geometry.

4.15.1.7 ANNOTATION SUBFIGURE

An instance of an annotation aggregation as defined by a subfigure definition.

```
*)
ENTITY annotation_subfigure
  SUBTYPE OF (annotation);
  name                : STRING;
  transformation      : transformation; --resource
  org_display_symbol  : OPTIONAL symbol;
  definition          : subfigure_definition;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: An identifying name.

transformation: The transformation which orients the instance relative to its definition orientation.
this instance.

org_display_symbol: A symbol used to designate the origin of the annotation subfigure.

definition: The defining subfigure definition.

4.15.1.8 NOTE

A type of **annotation** used to convey textual information.

```
*)
ENTITY note
  SUBTYPE OF (annotation);
  char_string          : STRING;
  appearance           : OPTIONAL text_appearance;
  attached_notes       : OPTIONAL SET [1 : #] OF note;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

char_string: The text string representing the information.

appearance: The appearance of the **annotation**. This is OPTIONAL since default appearance is set by **drawing**, and optionally set by **drawing sheet** and **drawing view**.

attached_notes: Other notes linked to this one.

4.15.1.9 PRODUCT ANNOTATION

The physical representation of the product information concerning dimensions, geometric tolerance, datums, etc. on the drawing sheet.

```
*)
ENTITY product_annotation
  SUPERTYPE OF (datum_representation XOR
                dimension_representation XOR
                feature_control_frame XOR
                leader_directed_callout XOR
                section_representation)
  SUBTYPE OF (annotation);
END_ENTITY;
(*
```

4.15.1.10 DATUM REPRESENTATION

The physical representation of datum information on the drawing. **Datum representation** is a subtype of **product annotation**; **datum feature symbol** and **datum target symbol** are subtypes of **datum representation**.

```
*)
ENTITY datum_representation
  SUPERTYPE OF (datum_target_symbol XOR
                datum_feature_symbol)
  SUBTYPE OF (product_annotation);
  association     : OPTIONAL datum_association;
  relator         : SET [1 : #] OF shape_relator;
```

```
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

association: The link between datum representation and datum association.

relator: The link between datum representation and shape relator.

4.15.1.11 DATUM FEATURE SYMBOL

Identifies datum features on the drawing. The datum feature symbol consists of a frame box which contains the datum identifier.

```
*)
ENTITY datum_feature_symbol
  SUBTYPE OF (datum_representation);
  symbol           : symbol;
  datum_identifier : STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

symbol: The frame box used to enclose the datum feature symbol.

datum_identifier: The letter or letters used to identify a datum which are preceded by and followed by a dash.

4.15.1.12 DATUM TARGET SYMBOL

The datum target symbol identifies the datum target. It is a circle divided horizontally into two halves; the lower half contains the datum target identifier and the upper half contains the target area size, where applicable. When the datum target is not an area the upper half of the circle is left blank.

```
*)
ENTITY datum_target_symbol
  SUBTYPE OF (datum_representation);
  symbol                  : symbol;
  target_area_size        : OPTIONAL target_area_size;
  datum_target_identifier : datum_target_identifier;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

symbol: The circular perimeter and divider used to enclose the datum target symbol

target_area_size: The target area size gives the area of the datum target (where applicable).

datum_target_identifier: The datum target identifier includes the datum identifier and the datum target number; it is enclosed in the lower half of the datum target symbol and is used to identify the datum target.

4.15.1.13 DIMENSION REPRESENTATION

A physical representation of a dimension.

```
*)
ENTITY dimension_representation
  SUPERTYPE OF (angular_dimension_representation XOR
                linear_dimension_representation XOR
                ordinate_dimension_representation XOR
                radius_dimension_representation)
  SUBTYPE OF (product_annotation);
  association : OPTIONAL dimension_association;
  note : note;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

association: The link to the information content of the dimension.

note: The dimension note.

4.15.1.14 ANGULAR DIMENSION REPRESENTATION

A physical representation of an angular dimension.

```
*)
ENTITY angular_dimension_representation
  SUBTYPE OF (dimension_representation);
  radius                : REAL;
  end_extension         : OPTIONAL line_segment; --resource
  start_extension       : OPTIONAL line_segment; --resource
  second_leader         : OPTIONAL leader;
  first_leader          : OPTIONAL leader;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

radius: The radius for the dimension arc.

end_extension: The line for the terminating extension.

start_extension: The line for the beginning extension.

second_leader: The leader on the terminating side.

first_leader: The leader on the beginning side.

4.15.1.15 LINEAR DIMENSION REPRESENTATION

A physical representation of a linear dimension.

```
*)
ENTITY linear_dimension_representation
  SUBTYPE OF (dimension_representation);
  end_extension          : OPTIONAL line_segment; --resource
  start_extension        : OPTIONAL line_segment; --resource
  second_leader : OPTIONAL leader;
  first_leader  : OPTIONAL leader;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

end_extension: The line for the terminating extension.

start_extension: The line for the beginning extension.

second_leader: The leader on the terminating side.

first_leader: The leader on the beginning side.

4.15.1.16 ORDINATE DIMENSION REPRESENTATION

A physical representation of an ordinate dimension.

```
*)
ENTITY ordinate_dimension_representation
  SUBTYPE OF (dimension_representation);
  extension_line : OPTIONAL line_segment; --resource
  leader         : OPTIONAL leader;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

extension_line: An extension line.

leader: A leader.

4.15.1.17 RADIUS DIMENSION REPRESENTATION

A physical representation of a radius dimension.

```
*)
ENTITY radius_dimension_representation
  SUPERTYPE OF (diameter_dimension_representation)
  SUBTYPE OF (dimension_representation);
  arc_center : cartesian_two_coordinate; --resource
  leader     : leader;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

arc_center: The center of the arc whose radius is dimensioned.

leader: A leader.

4.15.1.18 DIAMETER DIMENSION REPRESENTATION

A physical representation of a diameter dimension.

```
*)
ENTITY diameter_dimension_representation
  SUBTYPE OF (radius_dimension_representation);
  second_leader : leader;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

second_leader: Leader on second end.

4.15.1.19 FEATURE CONTROL FRAME

A **feature control frame** specifies a geometric tolerance for an individual feature; it is divided into compartments containing the geometric characteristic **symbol** followed by the tolerance. Where applicable, the tolerance is preceded by the diameter **symbol** and followed by a material condition **symbol**. A composite **feature control frame** is used when more than one tolerance is specified for the same characteristic of a feature.

```
*)
ENTITY feature_control_frame
  SUBTYPE OF (product_annotation);
  association            : OPTIONAL geometric_tolerance_association;
  sym_group              : symbol_group_specification;
  geo_chrstc             : geometric_characteristic_compartment;
  divider                : SET [1 : ?] OF symbol;
  tol_specification      : tol_specification_compartment;
  max_tol_specification  : OPTIONAL
                             maximum_tolerance_value_specification;
  pri_datum              : OPTIONAL primary_datum_compartment;
  sec_datum              : OPTIONAL secondary_datum_compartment;
  ter_datum              : OPTIONAL tertiary_datum_compartment;
  relator                : SET [1 : ?] OF shape_relator;
WHERE
  association = NULL OR
   (feature_control_frame_map(feature_control_frame, association));
  sym_group.font_code_id = 'geometric tolerance';
  divider.symbol_id = 'divider';
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

association: The link between feature control frame and the information stored in geometric tolerance association.

sym_group: The link to the appearance parameters for a group of symbols.

geo_chrstc: The aspect of the part or feature that a tolerance is directed to, such as flatness or circularity for a form tolerance, or position for a position tolerance.

divider: The divider is a vertical line which separates the compartments in a feature control frame.

tol_specification: The enumeration of the required tolerance parameters of a feature in the feature control frame. It must contain the tolerance value, and may also contain the tolerance cylindrical form, the tolerance material condition, the tolerance unit area, and the tolerance unit length.

max_tol_specification: When specifying perpindicularity for an axis with zero tolerance at maximum material condition a maximum tolerance may be specified with a maximum tolerance value specification.

pri_datum: The datum of highest precedence in the datum reference.

sec_datum: The datum of second-highest precedence in the datum reference.

ter_datum: The datum of least precedence in the datum reference.

relator: The link between feature control frame and shape relator.

PROPOSITIONS:

1. Either association is null or the feature control frame map function returns TRUE for feature control frame and association as arguments.
2. The font code of sym group has the value 'geometric tolerance'.
3. The symbol id of divider has the value 'divider'.

4.15.1.20 LEADER DIRECTED CALLOUT

A common aggregation of a note and a leader.

```
*)
ENTITY leader_directed_callout
  SUBTYPE OF (product_annotation);
  note   : note;
  leader : leader;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

note: The note which conveys product information.

leader: The shape relator which links the annotation to the geometric representation.

4.15.1.21 SECTION REPRESENTATION

The elementary physical representation of sectioning.

```
*)
ENTITY section_representation
  SUBTYPE OF (product_annotation);
  segment      : OPTIONAL SET [1 : #] OF line_segment; --resource
  parameters   : OPTIONAL section_parameters;
  association : OPTIONAL sectioned_area_association;
WHERE
  (segment <> NULL OR (parameters <> NULL AND association <> NULL));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

segment: Lines which convey each individual segment of the **section representation**.

parameters: Parameters for generating the representation from the association.

association: The association to the product information required to regenerate the representation.

4.15.1.22 RELATION ANNOTATION

Annotation which represents an association between two instances of **Annotation** or between the shape representation and a instance of **Annotation**.

```
*)
ENTITY relation_annotation
  SUPERTYPE OF (note_relator XOR
                shape_relator)
  SUBTYPE OF (annotation);
END_ENTITY;
(*
```

4.15.1.23 NOTE RELATOR

A type of **relation annotation** which links notes to other notes or to features on a part.

```
*)
ENTITY note_relator
  SUPERTYPE OF (implicit_generalization XOR
                named_variable XOR
                note_flag)
  SUBTYPE OF (relation_annotation);
  association         : OPTIONAL note_relator_association;
  note                : note;
  related_annotation : annotation;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

association: The information content represented by the relator.

note: The note linked to.

related_annotation: Other annotation related to the relator.

4.15.1.24 IMPLICIT GENERALIZATION

A type of note relator where the relation is a generalization implied in the note, e.g., break all edges.

```
*)
ENTITY implicit_generalization
  SUBTYPE OF (note_relator);
  atts: undefined; --resource
END_ENTITY;
(*
```

4.15.1.25 NAMED VARIABLE

A type of note relator which links a variable name to a name in a note, e.g., for tabular dimensions.

```
*)
ENTITY named_variable
  SUBTYPE OF (note_relator);
  name  : STRING;
  note  : note;
  atts  : undefined; --resource
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: The variable label.

note: The note associated to the variable.

4.15.1.26 NOTE FLAG

A type of note relator which uses a flag symbol with a note inside to relate two notes.

```
*)
ENTITY note_flag
  SUBTYPE OF (note_relator);
  flag : symbol;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

flag: The symbol which represents the flag.

4.15.1.27 SHAPE RELATOR

A type of **relation annotation** which links a piece of **annotation** to features on a part.

```
*)
ENTITY shape_relator
  SUPERTYPE OF (center_line XOR
                dimension_line XOR
                extension_line XOR
                leader)
  SUBTYPE OF (relation_annotation);
  association  : OPTIONAL shape_relator_association;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

association: The information content represented by the relator.

4.15.1.28 CENTER LINE

A type of **shape relator** which may also be a feature representation.

```
*)
ENTITY center_line
  SUBTYPE OF (shape_relator);
END_ENTITY;
(*
```

4.15.1.29 DIMENSION LINE

The line which relates the dimension value to the feature extensions or to the feature itself.

```
*)
ENTITY dimension_line
  SUBTYPE OF (shape_relator);
END_ENTITY;
(*
```

4.15.1.30 EXTENSION LINE

A type of **shape relator** that extends from the geometric representation away from the part which be used by the **dimension lines.**

```
*)
ENTITY extension_line
  SUBTYPE OF (shape_relator);
END_ENTITY;
(*
```

4.15.1.31 LEADER

A type of **shape relator** that extends from the geometric representation away from the part. Dimensions, notes, and geometric tolerance information can be directed to the part via a **leader**.

```
*)
ENTITY leader
  SUBTYPE OF (shape_relator);
  arrow_width          : OPTIONAL REAL;
  arrow_height         : OPTIONAL REAL;
  terminator_start     : cartesian_two_coordinate; --resource
  segment_pt           : LIST [1 : #] OF
                          cartesian_two_coordinate; --resource
  terminator           : OPTIONAL symbol;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

arrow_width: The width of the terminator if an arrowhead.

arrow_height: The height or length of the terminator if an arrowhead.

terminator_start: The location of the locating point of the terminator.

segment_pt: The locations which form the segment points for the **leader**.

terminator: The symbol used at the end of the **leader**.

4.15.1.32 SYMBOL

A glyph with a common interpretation, typically included in a library.

```
*)
ENTITY symbol
  SUBTYPE OF (annotation);
  symbol_id : STRING;
  direction : OPTIONAL direction; --resource
  scale     : OPTIONAL REAL;
UNIQUE
  symbol_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

symbol_id: A unique identifier for the **symbol**.

direction: An orienting **direction** for the instance.

scale: The scale for the instance.

4.15.1.33 APPEARANCE ASSOCIATIVITY

The association between appearance parameters and one or more curves.

```
*)
ENTITY appearance_associativity;
  appearance : curve_appearance;
  curve : SET [1 : #] OF curve; --resource
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

appearance: The appearance parameters for the geometry.

curve: The geometry affected by the appearance parameters.

4.15.1.34 APPEARANCE SPECIFICATION

The supertype entity for the two appearance entities which control appearance of text and curves.

```
*)
ENTITY appearance_specification
  SUPERTYPE OF (curve_appearance OR
                text_appearance);
END_ENTITY;
(*
```

4.15.1.35 CURVE APPEARANCE

The parameters which control the appearance of a curve.

```
*)
ENTITY curve_appearance
  SUPERTYPE OF (full_entity_appearance XOR
                partial_entity_appearance)
  SUBTYPE OF (appearance_specification);
  line_style       : INTEGER;
  line_weight      : INTEGER;
  color            : INTEGER;
  blanked          : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

line_style: The style code for the curve. (predefined codes)

line_weight: The weight code for the curve. (predefined)

color: The color of the curve.

blanked: TRUE if the curve is to be blanked from the display.

4.15.1.36 FULL ENTITY APPEARANCE

A type of curve appearance where the full entity is affected.

```
*)
ENTITY full_entity_appearance
  SUBTYPE OF (curve_appearance);
END_ENTITY;
(*
```

4.15.1.37 PARTIAL ENTITY APPEARANCE

The appearance parameters applied to a portion of an entity.

```
*)
ENTITY partial_entity_appearance
  SUBTYPE OF (curve_appearance);
  from_intersecting_element   : shape_element_type; --resource
  to_intersecting_element     : OPTIONAL
                                shape_element_type; --resource
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

from_intersecting_element: The entity whose intersection with the referencing entity defines the start of the region controlled by the appearance parameters.

to_intersecting_element: The entity whose intersection with the referencing entity defines the end of the region controlled by the appearance parameters.

4.15.1.38 TEXT APPEARANCE

The parameters which govern the appearance of text. To be integrated with presentation. This list of attributes is incomplete.

```
*)
ENTITY text_appearance
  SUBTYPE OF (appearance_specification);
  string_angle : OPTIONAL REAL;
  char_angle   : OPTIONAL REAL;
  mirror_flag  : OPTIONAL INTEGER;
  slant_angle  : OPTIONAL REAL;
  font_code    : OPTIONAL INTEGER;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

string_angle: The slant angle for the baseline of the string.

char_angle: The individual rotation angles for characters with positive angles measured counterclockwise from the string angle. Rotation is about the lower left corner at the baseline.

mirror_flag: Mirroring options.

slant_angle: The slant angle for individual characters with positive values measured clockwise from the normal to the char angle.

font_code: The desired font identifier.

4.15.1.39 DATUM TARGET IDENTIFIER

The **datum target identifier** includes the datum identifier and the datum target number; it is enclosed in the lower half of the **datum target symbol** and is used to identify the datum target.

```
*)
ENTITY datum_target_identifier;
  datum_identifier     : STRING;
  datum_target_number : STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

datum_identifier: The letter or letters used to identify a **datum**.

datum_target_number: The **datum target number** forms part of the **datum target identifier**. Numbers are assigned sequentially, starting with 1, for each **datum**.

4.15.1.40 DEPTH

An entity whose existence denotes definition in model space, and whose.value provides the z coordinate for the referencing entity.

```
*)
ENTITY depth;
  z_value : real;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

z_value: Z coordinate value.

4.15.1.41 DATUM REFERENCE COMPARTMENT

The origin of the dimensional relationship between a toleranced feature and a designated feature on a part.

```
*)
ENTITY datum_reference_compartment
  SUPERTYPE OF (primary_datum_compartment XOR
                secondary_datum_compartment XOR
                tertiary_datum_compartment);
  datum_id  : STRING;
  matl_cond : OPTIONAL symbol;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

datum_id: The letter or letters used to identify a **datum**.

matl_cond: The relative limit of size condition of the designated feature that is to be used as the datum in the dimensional relationship.

4.15.1.42 PRIMARY DATUM COMPARMENT

The **primary datum** indicates the origin of a dimensional relationship of highest precedence between a toleranced feature and a designated feature on a part.

```
*)
ENTITY primary_datum_compartment
  SUBTYPE OF (datum_reference_compartment);
END_ENTITY;
(*
```

4.15.1.43 SECONDARY DATUM COMPARTMENT

The **secondary datum** indicates the origin of a dimensional relationship of second highest precedence between a toleranced feature and a designated feature on a part.

```
*)
ENTITY secondary_datum_compartment
  SUBTYPE OF (datum_reference_compartment);
END_ENTITY;
(*
```

4.15.1.44 TERTIARY DATUM COMPARTMENT

The **tertiary datum** indicates the origin of a dimensional relationship of lowest precedence between a toleranced feature and a designated feature on a part.

```
*)
ENTITY tertiary_datum_compartment
  SUBTYPE OF (datum_reference_compartment);
END_ENTITY;
(*
```

4.15.1.45 DRAWING

A human-interpretable representation of PDD conforming to specific standards. The drawing has both physical and informational aspects. Physical aspects are developed under **drawing sheet**, which acts as an organizing subdivision of the drawing. Informational aspects are divided into **product association**, which includes information about the product, and **non product association** which includes non-product aspects, such as drawing administration data.

```
*)
ENTITY drawing;
  sheet                    : LIST [1:#] OF drawing_sheet;
  type                     : drawing_type;
  product_association      : OPTIONAL SET [1 : #] OF
                                        product_association;
  non_product_association : OPTIONAL SET [1 : #] OF
                                        non_product_association;
  default_appearance       : appearance_specification;
END_ENTITY;

(*
```

ATTRIBUTE DEFINITIONS:

sheet: The physical subdivision of the drawing.

type: The type of drawing.

product_association: The category of information which provides drafting's view of, and association to, product data, such as dimensions or tolerances.

non_product_association: The category of information which provides drafting's view of, and association to, non product data, such as that governing the administration of the drawing.

default_appearance: The highest level for setting default appearance parameters for curves and text.

4.15.1.46 DRAWING REQUIREMENT

The predefined contractural or organizational list of requirements for drawings such as ANSI, ISO, or MIL standards or other referenced documents.

```
*)
ENTITY drawing_requirement;
  reference              : SET [1 : #] of STRING;
  format_specification : OPTIONAL sheet_format_specification;
  standard               : drafting_standard;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

reference: A requirements document or standard which controls the development and/or interpretation of the drawing.

format_specification: A required format specification.

standard: The standard controlling the drawing presentation.

4.15.1.47 DRAWING SHEET

The drawing sheet is the physical foundation for locating and subdividing representaions of PDD. The drawing sheet is administered according to information contained in sheet administration association. Characteristics of the first sheet may differ from subsequent sheets. The ordering of the sheets is accomplished by using the LIST aggregation type in the drawing reference to drawing sheet.

```
*)
ENTITY drawing_sheet;
  view                : OPTIONAL SET [1 : #] OF drawing_view;
  association         : OPTIONAL sheet_administration_association;
  format              : OPTIONAL sheet_format;
  size                : sheet_size;
  default_appearance  : OPTIONAL appearance_specification;
  annotation          : SET [1 : #] OF annotation;
  reference_point     : cartesian_two_coordinate; --resource
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

view: Views to appear on the sheet.

association: The association to information governing the administration of the sheet, such as the sheet revision.

format: The format to be placed on the sheet with the origin at the reference point.

size: The size of the sheet.

default_appearance: The default appearance parameters. This is OPTIONAL since default appearance is set by drawing, and optionally set by drawing view.

annotation: Annotation which is defined on the sheet.

reference_point: A point which specifies the origin of the sheet coordinate system relative to the bottom left corner of the sheet. Both x and y values must be positive.

4.15.1.48 DRAWING VIEW

The structure for presenting the product geometry with its appearance attributes. The geometry may be either a three-dimensional model or it may be an explicit projection of a model onto the view plane (with or without retaining associativity to an underlying model).

```
*)
ENTITY drawing_view;
  specification                : view_specification;
  origin_on_sheet              : cartesian_two_coordinate; --resource
```

```
  range_on_sheet              : two_dimensional_range;
  selected_model_geometry     : OPTIONAL selected_model_geometry;
  projected_model             : OPTIONAL projected_model;
  name                        : OPTIONAL STRING;
  default_model               : OPTIONAL model_view_appearance;
  default_appearance          : OPTIONAL appearance_specification;
  annotation                  : OPTIONAL SET [1 : #] OF annotation;
WHERE
  NOT (selected_model_geometry <> NULL AND projected_model <> NULL);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

specification: The parameters defining the view.

origin_on_sheet: The location of the view origin in terms of sheet coordinates.

range_on_sheet: The range of sheet coordinates which completely encompass the view geometry and any associated annotation. The range should be considered to potentially clip both the annotation and geometry, while the clip in view specification is only for geometry.

selected_model_geometry: The link to the entity which specifies the geometry and associated appearance parameters to be visible in a view.

projected_model: The link to the aggregation of elements and associated appearance parameters for geometry which has been projected into the view according to the view specification.

name: An identifying name for the view.

default_model: Defaultable parameter which governs the high level appearance of hidden edges in the models.

default_appearance: Defaultable parameters for the appearance of entities visible through the view.

annotation: Annotation which is defined in the view.

4.15.1.49 GEOMETRIC CHARACTERISTIC COMPARTMENT

The aspect of the part or feature that a tolerance is directed to, such as flatness or circularity for a form tolerance, or position for a position tolerance.

```
*)
ENTITY geometric_characteristic_compartment;
  chrstc_sym : symbol;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

chrstc sym: The symbol which represents the geometric characteristic.

4.15.1.50 INFORMATION FIELD

A rectangular region used in building formats and tables. Contains the description of the content of the field along with defining geometry, symbols, and fixed text.

```
*)
ENTITY information_field;
  use                    : undefined --resource;
  lower_left             : cartesian_two_coordinate; --resource
  direction_left_line    : two_space_direction; --resource
  block_height           : REAL;
  block_length           : REAL;
  top_line_visible       : LOGICAL;
  bottom_line_visible    : LOGICAL;
  left_line_visible      : LOGICAL;
  right_line_visible     : LOGICAL;
  information_content    : undefined; --resource
  further_info_fields    : OPTIONAL LIST [1 : #] OF information_field;
  note                   : OPTIONAL LIST [1 : #] OF note;
  symbol                 : OPTIONAL LIST [1 : #] OF symbol;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

use: The type of information in the block.

lower_left: The lower left corner.

direction_left_line: The direction of the left line of the rectangle. This provides the orientation.

block_height: The height of the block.

block_length: The length of the block.

top_line_visible: TRUE if the line is visible.

bottom_line_visible: TRUE if the line is visible.

left_line_visible: TRUE if the line is visible.

right_line_visible: TRUE if the line is visible.

information_content: The information in the block.

further_info_fields: Other linked blocks, i.e., sub-blocks.

note: Fixed textual information as for the block label.

symbol: Symbols used in the block.

4.15.1.51 MAXIMUM TOLERANCE VALUE SPECIFICATION

When specifying perpendicularity for an axis with zero tolerance at maximum material condition a maximum tolerance may be specified with a **maximum tolerance value specification.**

```
*)
ENTITY maximum_tolerance_value_specification;
  diameter_form : OPTIONAL symbol;
  max_tol_value : STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

diameter_form: The **symbol** used to indicate that the **maximum tolerance value specification** applies to a diameter.

max_tol_value: The value of the **maximum tolerance value specification.**

4.15.1.52 NON PRODUCT ASSOCIATION

The **Non-Product Association** entities convey the Drafting application's view of, and association to, the Non-Product Definition Data. It has appropriate structure and augmentation components but should be independent of physically representing the information. These entities may point out areas of information which are conveyed by the application but are not contained by any resource model. Examples of **Non-Product Association** entities are administrative information such as the sheet revision information and the drawing title.

```
*)
ENTITY non_product_association
  SUPERTYPE OF (drawing_administration_association XOR
                sheet_administration_association);
END_ENTITY;
(*
```

4.15.1.53 DRAWING ADMINISTRATION ASSOCIATION

Drafting's view of, and association to, the information which governs the administration of a **drawing.**

```
*)
ENTITY drawing_administration_association
  SUBTYPE OF (non_product_association);
  number                : STRING;
  title                 : STRING;
  revision              : STRING;
  owner                 : person_and_organization; --resource
  requirement           : drawing_requirement;
  status                : drawing_status;
  classification        : OPTIONAL security_classification;
  title_classification  : OPTIONAL security_classification;
```

```
  user_defined          : OPTIONAL SET [1 : #] OF
                            user_defined; --resource
  title_note            : note;
  number_note           : note;
  revision_note         : note;
WHERE
  title_note.char_string = title;
  number_note.char_string = number;
  revision_note.char_string = revision;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

number: The drawing number.

title: The drawing title.

revision: The drawing revision.

owner: The drawing owner.

requirement: Requirements associated with the drawing.

status: The status of the review and approval of the drawing.

classification: The drawing security classification.

title_classification: The drawing title security classification.

user_defined: Any user defined administrative information.

title_note: A note which conveys the title.

number_note: A note which conveys the drawing number.

revision_note: A note which conveys the revision.

PROPOSITIONS:

1. The character string in title must match that in title note.
2. The character string in number note must match that in number.
3. The character string in revision note must match that in revision.

4.15.1.54 SHEET ADMINISTRATION ASSOCIATION

Drafting's view of, and association to, the information which governs the administration of the sheet.

```
*)
ENTITY sheet_administration_association
  SUBTYPE OF (non_product_association);
  sheet_number      : STRING;
  security_class    : security_classification;
```

```
  revision_level    : STRING;
  revision_history : OPTIONAL LIST [1 : #] OF sheet_revision_log;
  approvals         : LIST [1 : #] OF sheet_approval;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sheet_number: The sheet number.

security_class: The sheet classification.

revision_level: The sheet revision level.

revision_history: The revision history.

approvals: The approvals for the sheet.

4.15.1.55 PRODUCT ASSOCIATION

Prodcuct Definition Data is the bulk of the information conveyed on a drawing. The product association entities convey the Drafting applications view of, and association to, the PDD. It has appropriate structure and augmentation but should be independent of physically representing the information. Additionally, these entities may point out areas of information which are conveyed by the application but which are not covered by any of the resource models.

```
*)
ENTITY product_association
  SUPERTYPE OF (non_shape_association XOR
                shape_association);
END_ENTITY;
(*
```

4.15.1.56 NON SHAPE ASSOCIATION

Drafting's view of, and association to, the information about the product which does not determine the shape representation.

```
*)
ENTITY non_shape_association
  SUPERTYPE OF (datum_association XOR
                dimension_association XOR
                feature_association XOR
                geometric_tolerance_association XOR
                manufacturing_process_association XOR
                mark_and_handle_association XOR
                material_association XOR
                note_relator_association XOR
                product_administration_association XOR
                shape_relator_association)
  SUBTYPE OF (product_association);
END_ENTITY;
(*
```

4.15.1.57 DATUM ASSOCIATION

The information required to completely specify a datum. **Datum association** is a subtype of non shape association.

```
*)
ENTITY datum_association
  SUBTYPE OF (non_shape_association);
  datum                  : datum; --tolerance
  datum_id               : STRING;
  datum_feature          : dt_feature; --tolerance
  datum_target_number    : OPTIONAL INTEGER;
  target_area_diameter   : OPTIONAL REAL;
  target_area_size       : OPTIONAL REAL;
  datum_target_position  : LOGICAL;
UNIQUE
  datum_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

datum: The associativity to the **datum** entity in the Tolerance Model.

datum_id: The letter or letters used to identify a **datum**.

datum_feature: The associativity to **d-t feature** in the Tolerance Model.

datum_target_number: The **datum target number** forms part of the **datum target identifier**. Numbers are assigned sequentially, starting with 1, for each datum.

target_area_diameter: When the area of a datum target is circular the diameter of the area is given in the target area size portion of the **datum target symbol**.

target_area_size: The area size, where applicable, gives the real number value of the datum target surface area.

datum_target_position: The specification of whether the datum target on a drawing is located on the visible or hidden surface of a part.

```
*)
RULE association_of_datum FOR (datum_association, datum);
  specifies_a_datum : datum_association;
  repesents_a_datum : datum;
WHERE
  specifies_a_datum.datum_id = repesents_a_datum.name;
  specifies_a_datum.datum_feature = repesets_a_datum.reference;
END_RULE;
(*
```

PROPOSITIONS:

1. Datum id and the name field of datum are identical.
2. Datum feature and the reference field of datum are identical.

4.15.1.58 DIMENSION ASSOCIATION

Drafting's view of, and association to, the information content of a dimension.

```
*)
ENTITY dimension_association
  SUBTYPE OF (non_shape_association);
  dimension       : coordinate_dimension; --tolerance
  target          : dt_feature; --tolerance
  origin          : OPTIONAL dt_feature; --tolerance
  max_tol         : OPTIONAL REAL;
  min_tol         : OPTIONAL REAL;
  directed        : LOGICAL;
  direction       : OPTIONAL direction; --resource
  parameter_value : OPTIONAL REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

dimension: The dimensionality being conveyed.

target: The target (or measured-to) of the dimension.

origin: The origin (or measured-from) of the dimension.

max_tol: The maximum tolerance value for the dimension.

min_tol: The minimum tolerance value for the dimension.

directed: TRUE for a directed dimension.

direction: The direction for a directed dimension.

parameter_value: A parameter value for a parametric dimension.

4.15.1.59 FEATURE ASSOCIATION

Drafting's view of, and association to, the information content of feature representations on the drawing.

```
*)
ENTITY feature_association
  SUBTYPE OF (non_shape_association);
  atts : undefined; --resource
END_ENTITY;
(*
```

4.15.1.60 GEOMETRIC TOLERANCE ASSOCIATION

The information required to specify a geometric tolerance; the information is displayed on the drawing in a feature control frame.

```
*)
ENTITY geometric_tolerance_association
  SUBTYPE OF (non_shape_association);
  gtol                    : geometric_tolerance; --tolerance
  toleranced_ents         : SET [1 : #] OF dt_feature; --tolerance
  gc_representation       : STRING;
  tol_dia_form_mag        : REAL;
  magnitude               : REAL;
  tol_mc                  : OPTIONAL tol_misn; --tolerance
  tol_unit_area           : OPTIONAL REAL;
  tol_unit_length         : OPTIONAL REAL;
  max_tol_dia_form_mag    : OPTIONAL REAL;
  max_tol_magnitude       : OPTIONAL REAL;
  pri_datum_id            : OPTIONAL STRING;
  pri_datum_mc            : OPTIONAL tol_misn; --tolerance
  sec_datum_id            : OPTIONAL STRING;
  sec_datum_mc            : OPTIONAL tol_misn; --tolerance
  ter_datum_id            : OPTIONAL STRING;
  ter_datum_mc            : OPTIONAL tol_misn; --tolerance
  proj_tol_magnitude      : OPTIONAL REAL;
WHERE
  gtol_map(geometric_tolerance_association, gtol);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

gtol: The associativity to geometric tolerance in the Tolerance Model.

toleranced_ents: The associativity from geometric tolerance association to a d-t feature in the Tolerance Model.

gc_representation: The identifying string for the Geometric Characteristic symbols which are contained in the feature control frame.

tol_dia_form_mag: The real number value of the tolerance in cylindrical form. When a diameter symbol is used in a feature control frame it indicates that the tolerance value following the symbol applies to a diameter.

magnitude: The real number value of the maximum allowed deviation from the specified ideal value.

tol_mc: The relative limit-of-size condition applied to a feature on the tolerance specification of the feature control frame.

tol_unit_area: The real number value of the unit of surface area on the part for which the tolerance value gives the maximum allowed deviation for a geometric characteristic.

tol_unit_length: The real number value of the length- wise limit of the part (or the length of the surface area) for which the tolerance value gives the maximum allowed deviation for a geometric characteristic.

max_tol_dia_form_mag: The real number value of the cylindrical form of the maximum tolerance value specification.

max_tol_magnitude:

pri_datum_id: The letter or letters used to identify the primary datum.

pri_datum_mc: The relative limit-of-size condition applied to the primary datum feature.

sec_datum_id: The letter or letters used to identify the secondary datum.

sec_datum_mc: The relative limit-of-size condition applied to the secondary datum feature.

ter_datum_id: The letter or letters used to identify the tertiary datum.

ter_datum_mc: The relative limit-of-size condition applied to the tertiary datum feature.

proj_tol_magnitude: The real number value of the height of a projected tolerance zone.

PROPOSITIONS:

1. The **gtol map** function returns TRUE for the arguments (**feature control frame association**, **gtol**).

4.15.1.61 MANUFACTURING PROCESS ASSOCIATION

A type of **non shape association** which contains information about the manufacturing process.

```
*)
ENTITY manufacturing_process_association
  SUBTYPE OF (non_shape_association);
  atts : undefined; --resource
END_ENTITY;
(*
```

4.15.1.62 MARK AND HANDLE ASSOCIATION

A type of **non shape association** which contains information about marking and handling the product.

```
*)
ENTITY mark_and_handle_association
  SUBTYPE OF (non_shape_association);
  atts : undefined; --resource
END_ENTITY;
(*
```

4.15.1.63 MATERIAL ASSOCIATION

A type of **non shape association** which contains information concerning the material(s) included in the product information.

```
*)
ENTITY material_association
  SUBTYPE OF (non_shape_association);
  atts : undefined; --resource
END_ENTITY;
(*
```

4.15.1.64 NOTE RELATOR ASSOCIATION

Drafting's view of, and association to, the information represented by the **note relator**.

```
*)
ENTITY note_relator_association
  SUBTYPE OF (non_shape_association);
  atts: undefined; --resource
END_ENTITY;
(*
```

4.15.1.65 PRODUCT ADMINISTRATON ASSOCIATION

Drafting's view of, and association to, the information which governs the administration of the product.

```
*)
ENTITY product_administration_association
  SUBTYPE OF (non_shape_association);
  part_number          : STRING;
  part_name            : STRING;
  part_classification : security_classification;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

part_number: The part number.

part_name: The part name.

part_classification: The part classification.

4.15.1.66 SHAPE RELATOR ASSOCIATION

Drafting's view of, and association to, the information represented by the **shape relator**.

```
*)
ENTITY shape_relator_association
  SUBTYPE OF (non_shape_association);
  atts: undefined; --resource
END_ENTITY;
(*
```

4.15.1.67 SHAPE ASSOCIATION

Drafting's view of, and association to, the information about the product which determines the shape representation.

```
*)
ENTITY shape_association
  SUPERTYPE OF (sectioned_area_association OR
```

```
                - view_specification)
  SUBTYPE OF (product_association);
END_ENTITY;
(*
```

4.15.1.68 SECTIONED AREA ASSOCIATION

Drafting's view of, and association to, the information required to support regeneration of the section representation.

```
*)
ENTITY sectioned_area_association
  SUBTYPE OF (shape_association);
  boundary_curve      : bounded_curve; --resource
  island_curve        : OPTIONAL SET [1:#] OF
                          bounded_curve; --resource
  material            : OPTIONAL material_association;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

boundary_curve: The bounded geometry which defines the outer boundary for the section.

island_curve: Bounded geometry which defines island areas within the boundary curve.

material: The association to the product material information.

4.15.1.69 VIEW SPECIFICATION

The parameters which define what part of the geometric representation is visible in a view, and how it is viewed.

```
*)
ENTITY view_specification
  SUBTYPE OF (shape_association);
  coordinate_system : transformation; --resource
  clip              : OPTIONAL view_clip;
  perspective_point : OPTIONAL
                        cartesian_three_coordinate; --resource
  base_angle_deg    : OPTIONAL REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

coordinate_system: The orientation and definition of the view coordinate system.

clip: The clipping boundaries for geometry.

perspective_point: A viewing perspective point.

base_angle_deg: The base rotation angle for placement of the view on the sheet. Positive angles are measured counterclockwise. Rotation is about the view origin.

4.15.1.70 PROJECTED MODEL

An explicit projection of a model onto the view plane (with or without retaining associativity to an underlying parent model).

```
*)
ENTITY projected_model;
  elements                    : SET [1 : #] OF annotation_geometry;
  parent_model                : OPTIONAL selected_model_geometry;
  appearance                  : OPTIONAL model_view_appearance;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

elements: Annotation geometry included in the **projected model**.

parent_model: The parent three dimensional model from which the geometry is projected.

appearance: Defaultable parameter which governs the high level appearance of hidden edges in the models.

4.15.1.71 SECTION PARAMETERS

The parameters for generating a **section representation** from a **sectioned area associativity**.

```
*)
ENTITY section_parameters;
  line_angle                  : REAL;
  line_spacing                : REAL;
  line_pattern_code           : INTEGER;
  line_thru                   : cartesian_two_coordinate; --resource
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

line_angle: The angle for the section lines.

line_spacing: The normal distance between the section lines.

line_pattern_code: A pattern code for the line style of the section lines.

line_thru: A location that one of the section lines passes through.

4.15.1.72 SECURITY CLASSIFICATION

The security classification structure.

```
*)
ENTITY security_classification;
  level                  : security_classification_level;
```

```
  category              : OPTIONAL user_defined; --resource
  access_restriction    : OPTIONAL user_defined; --resource
  date                  : date; --resource
  authority             : OPTIONAL person_and_organization;
  classifier_title      : OPTIONAL STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

level: The level of security classification.

category: A category modifier on the level.

access_restriction: A special restriction modifier.

date: The classification date.

authority: The authority which provided the classification.

classifier_title: The title of the person who provided the classification.

4.15.1.73 SELECTED MODEL GEOMETRY

The drafting aggregation of **shape** with appropriate appearance parameters for presentation in a view. Drafting may require shape geometry which augments the product model, as for the geometry which represents the product shape when cut by a cutting plane for a section view.

```
*)
ENTITY selected_model_geometry;
  pd_shape            : SET [1 : #] OF shape; --resource
  model_appearance    : OPTIONAL model_view_appearance;
  element_appearance  : OPTIONAL appearance_associativity;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

pd_shape: The aggregations of elements represented.

model_appearance: The high level parameters for hidden edge visualization or removal.

element_appearance: The link to provide a set of elements with a desired appearance, such as a dashed style on curves.

4.15.1.74 SHEET APPROVAL

The approval documentation for a sheet, as depicted on a drawing.

```
*)
ENTITY sheet_approval;
  prepared_by    : person_name; --resource
```

```
  prepared_date : date; --resource
  checked_by    : person_name; --resource
  checked_date  : date; --resource
  approval      : LIST [1 : #] OF approval; --resource
  issued_by     : person_and_organization; --resource
  issue_date    : date; --resource
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

prepared_by: Person who prepared the sheet.

prepared_date: Date the sheet was prepared.

checked_by: Person who checked the sheet.

checked_date: Date the sheet was checked.

approval: Approvals for the sheet.

issued_by: Issuing authority.

issue_date: Date of issue.

4.15.1.75 SHEET FORMAT

The aggregation of annotation which locates other blocks of administrative annotation.

```
*)
ENTITY sheet_format;
  name          : OPTIONAL STRING;
  specification : OPTIONAL sheet_format_specification;
  subfigure     : OPTIONAL annotation_subfigure;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: An identifier for the format.

specification: The specification associated with the format.

subfigure: A **annotation subfigure** which contains all of the fixed text and **annotation geometry** for the format.

4.15.1.76 SHEET FORMAT SPECIFICATION

The specification which determines the definition of the format.

```
*)
ENTITY sheet_format_specification;
  information_field : LIST [1 : #] OF information_field;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

information_field: The description of the content of the field along with defining geometry, symbols, and fixed text.

4.15.1.77 SHEET REVISION LOG

The structure for documenting sheet revisions on a drawing.

```
*)
ENTITY sheet_revision_log;
  revision      : STRING;
  date          : date; --resource
  explaination  : STRING;
  reference     : STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

revision: The revision level for the sheet.

date: The date of the revision.

explaination: An explaination of the revision.

reference: A reference to the affected zone(s).

4.15.1.78 SHEET SIZE

The size of the sheet.

```
*)
ENTITY sheet_size;
  designation : std_sheet_size;
  size        : OPTIONAL cartesian_two_coordinate; --resource
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

designation: A standard sheet size.

size: A point at the opposite diagonal from the sheet origin which specifies the size explicitly.

4.15.1.79 SUBFIGURE DEFINITION

An aggregation of **annotation**.

```
*)
ENTITY subfigure_definition;
  annotation : OPTIONAL SET [1:#] OF annotation;
  instance   : OPTIONAL SET [1:#] OF annotation_subfigure;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

annotation: Entities included.

instance: Instances where used.

4.15.1.80 SYMBOL GROUP SPECIFICATION

The specification which controls the appearance of a group of characters and symbols.

```
*)
ENTITY symbol_group_specification;
  font_code_id        : STRING;
  char_height_factor  : REAL;
  characters          : LIST [1 : #] OF symbol;
  underbar            : LOGICAL;
  overbar             : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

font_code_id: Identifier for desired font.

char_height_factor: Factor for scaling characters and symbols.

characters: Included characters/symbols.

underbar: TRUE if characters are to be underscored.

overbar: TRUE if characters are to be overscored.

4.15.1.81 TARGET AREA SIZE

The target area size gives the area of the datum target (where applicable).

```
*)
ENTITY target_area_size;
  area_diameter : OPTIONAL STRING;
  diameter_form : OPTIONAL symbol;
  area_size     : OPTIONAL STRING;
WHERE
  (area_diameter <> NULL and diameter_form <> NULL) OR
    area_size <> NULL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

area_diameter: When the area of a datum target is circular the diameter of the area is given in the target area size portion of the **datum target symbol.**

diameter_form: The **symbol** used to indicate diameter or circularity.

area_size: The area size, where applicable, gives the numerical value of the datum target surface area.

PROPOSITIONS:

1. Either **area diameter** and **diameter form** both have values or **area size** has a value.

4.15.1.82 TOL SPECIFICATION COMPARTMENT

The enumeration of the required tolerance parameters of a feature in the **feature control frame.** It must contain the tolerance value, and may also contain the tolerance cylindrical form, the tolerance material condition, the tolerance unit area, and the tolerance unit length.

```
*)
ENTITY tol_specification_compartment;
  diameter_form    : OPTIONAL symbol;
  tol_value        : STRING;
  matl_cond        : OPTIONAL symbol;
  proj_tol_form    : OPTIONAL symbol;
  tol_unit_area    : OPTIONAL STRING;
  tol_unit_length  : OPTIONAL STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

diameter_form: The **symbol** used to indicate diameter or circularity.

tol_value: The real number value of the maximum allowed deviation from the specified ideal value.

matl_cond: The relative limit of size condition of the designated feature that is to be used as the datum in the dimensional relationship.

proj_tol_form: The **symbol** which indicates that the tolerance applies to a projected tolerance zone.

tol_unit_area: The real number value of the unit of surface area on the part for which the tolerance value gives the maximum allowed deviation for a geometric characteristic.

tol_unit_length: The real number value of the length- wise limit of the part (or the length of the surface area) for which the tolerance value gives the maximum allowed deviation for a geometric characteristic.

4.15.1.83 TWO DIMENSIONAL RANGE

A range on the two dimensional drawing sheet.

```
*)
ENTITY two_dimensional_range;
  xmin_ymin : cartesian_two_coordinate; --resource
  xmax_ymax : cartesian_two_coordinate; --resource
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

xmin_ymin: The minimum x, minimum y coordinate, e.g. lower left.

xmax_ymax: The maximum x, maximum y coordinate, e.g. upper right.

4.15.1.84 VIEW CLIP

The region outside of which the geometric representation in a view is clipped.

```
*)
ENTITY view_clip;
  umin : REAL;
  umax : REAL;
  vmin : REAL;
  vmax : REAL;
  nmin : OPTIONAL REAL;
  nmax : OPTIONAL REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

umin: Minimum u value, view coordinate system (left).

umax: Maximum u value, view coordinate system (right).

vmin: Minimum v value, view coordinate system (bottom).

vmax: Maximum v value, view coordinate system (top).

nmin: Minimum n value, view coordinate system (back).

nmax: Maximum n value, view coordinate system (front).

4.15.1.85 Drafting FUNCTION Definitions

4.15.1.85.1 FEATURE CONTROL FRAME MAP

Function **feature control frame map** maps the feature control frame **feature control frame** from the drafting view of the tolerance information **feature control frame association**.

```
*)
FUNCTION feature_control_frame_map (fcf : feature_control_frame;
 association : geometric_tolerance_association) : LOGICAL;
 fcf.geo_chrstc.chrstc_sym.symbol_id :=
  association.gc_representation;
 IF association.tol_dia_form_mag <> NULL THEN
   fcf.tol_specification.diameter_form.symbol_id := 'diameter';
   fcf.tol_specification.tol_value :=
     real_to_char(association.tol_dia_form_mag,fcf.sym_group);
 END_IF;
 IF association.tol_mc < regardless THEN
   fcf.tol_specification.matl_cond.symbol_id :=
    mlsn_map(association.tol_mc);
   fcf.tol_specification.tol_val :=
     real_to_char(association.magnitude,fcf.sym_group);
 END_IF;
 IF association.max_tol_dia_form_mag <> NULL THEN
   fcf.max_tol_specification.diameter_form.symbol_id := 'diameter';
   fcf.max_tol_specification.max_tol_val :=
      real_to_char(association.max_tol_dia_form_mag,fcf.sym_group);
 END_IF;
 IF association.max_tol_val = NULL THEN
   fcf.max_tol_specification.max_tol_val :=
   real_to_char(association.max_tol_magnitude,
               fcf.sym_group);
 END_IF;
 IF association.pri_datum_id = NULL THEN
   fcf.pri_datum.datum_id := association.pri_datum_id;
 END_IF;
 IF association.pri_datum_mc < regardless THEN
   fcf.pri_datum.matl_cond.symbol_id :=
    mlsn_map(association.tol_mc);
 END_IF;
 IF association.sec_datum_id = NULL THEN
   fcf.sec_datum.datum_id := association.sec_datum_id;
 END_IF;
 IF association.sec_datum_mc < regardless THEN
   fcf.sec_datum.matl_cond.symbol_id :=
     mlsn_map(association.tol_mc);
 END_IF;
 IF association.ter_datum_id = NULL THEN
   fcf.ter_datum.datum_id := association.ter_datum_id;
 END_IF;
 IF association.ter_datum_mc < regardless THEN
   fcf.ter_datum.matl_cond.symbol_id :=
                            mlsn_map(association.tol_mc);
 END_IF;
 IF association.proj_tol_magnitude <> NULL THEN
   fcf.tol_specification.proj_tol_form.symbol_id :=
```

```
    'projected tolerance zone';
   fcf.tol_specification.tol_value :=
   real_to_char(association.proj_tol_magnitude, fcf.sym_group);
 END_IF;
RETURN(TRUE);
END_FUNCTION;
(*
```

4.15.1.85.2 GTOL MAP

Function **gtol map** generates the population of the **feature control frame** association based on the particular geometric tolerance. The function is intended to address the information off of the subtypes of tolerance (as it is divided in the Tolerance model, but we want access through a single connection to the supertype - hence the questionable use of 'typeof'. There is only enough here to show how we think it should be.

```
*)
FUNCTION gtol_map(association : geometric_tolerance_association;
 gtol : geometric_tolerance) :
   LOGICAL;
  CASE typeof(gtol) OF
       angularity :
      BEGIN
    association.gc_representation := 'angularity';
    association.toleranced_ents := gtol.toleranced_ents;
    association.magnitude := gtol.magnitude;
    association.tol_dia_form := NULL;
    association.tol_mc := gtol.material_condition;
    association.tol_unit_area := NULL;
    association.tol_unit_length := NULL;
    association.pri_datum_id := gtol.primary_datum.name;
    association.pri_datum_mc :=
     gtol_primary_datum.material_condition;
  IF gtol.sec_datum = NULL THEN
    association.sec_datum_id := gtol.sec_datum.name;
    association.sec_datum_mc := gtol.sec_datum.material_condition;
  END_IF;
  IF gtol.tertiary_datum = NULL THEN
    association.ter_datum_id := gtol.tertiary_datum.name;
    association.ter_datum_mc :=
     gtol.tertiary_datum.material_condition;
       END_IF;
      END;
   flatness :
      BEGIN
    association.gc_representation := 'flatness';
   -- etc for each appropriate attribute for flatness
      END;
   --etc for each geometric characteristic
```

```
    END_CASE;
RETURN(TRUE);
END_FUNCTION;
(*
```

4.15.1.85.3 MLSN MAP

Function mlsn map returns a string which represents a material condition.

```
*)
FUNCTION mlsn_map(matl_cond : tol_mlsn ) : STRING;
  CASE matl_cond OF
    maxmc : RETURN('maxmc');
    leastmc : RETURN('leastmc');
  END_CASE;
 END_FUNCTION;
(*
```

4.15.1.85.4 REAL TO CHAR

Function real to char converts a real number to a string according to a association.

```
*)
FUNCTION real_to_char(realnum : REAL;
 specification : symbol_group_specification) : STRING ;
END_FUNCTION;
(*
```

4.15.1.86 Drafting Entity Classification

The following indented listing provides the Classification Structure for the entities in the Drafting Schema.

```
ANNOTATION
  ANNOTATION GEOMETRY
  ANNOTATION SUBFIGURE
  NOTE
  PRODUCT ANNOTATION
    DATUM REPRESENTATION
      DATUM FEATURE SYMBOL
      DATUM TARGET SYMBOL
    DIMENSION REPRESENTATION
      ANGULAR DIMENSION REPRESENTATION
      LINEAR DIMENSION REPRESENTATION
      ORDINATE DIMENSION REPRESENTATION
      RADIUS DIMENSION REPRESENTATION
        DIAMETER DIMENSION REPRESENTATION
    FEATURE CONTROL FRAME
    LEADER DIRECTED CALLOUT
```

SECTION REPRESENTATION
RELATION ANNOTATION
NOTE RELATOR
IMPLICIT GENERALIZATION
NAMED VARIABLE
NOTE FLAG
SHAPE RELATOR
CENTER LINE
DIMENSION LINE
EXTENSION LINE
LEADER
SYMBOL
APPEARANCE ASSOCIATIVITY
APPEARANCE SPECIFICATION
CURVE APPEARANCE
FULL ENTITY APPEARANCE
PARTIAL ENTITY APPEARANCE
TEXT APPEARANCE
DATUM TARGET IDENTIFIER
DEPTH
DATUM REFERENCE COMPARTMENT
PRIMARY DATUM COMPARTMENT
SECONDARY DATUM COMPARTMENT
TERTIARY DATUM COMPARTMENT
DRAWING
DRAWING REQUIREMENT
DRAWING SHEET
DRAWING VIEW
INFORMATION FIELD
GEOMETRIC CHARACTERISTIC COMPARTMENT
MAXIMUM TOLERANCE VALUE SPECIFICATION
NON PRODUCT ASSOCIATION
DRAWING ADMINISTRATION ASSOCIATION
SHEET ADMINISTRATION ASSOCIATION
PRODUCT ASSOCIATION
NON SHAPE ASSOCIATION
DATUM ASSOCIATION
DIMENSION ASSOCIATION
FEATURE ASSOCIATION
GEOMETRIC TOLERANCE ASSOCIATION
MANUFACTURING PROCESS ASSOCIATION
MARK AND HANDLE ASSOCIATION
MATERIAL ASSOCIATION
NOTE RELATOR ASSOCIATION
PRODUCT ADMINISTRATION ASSOCIATION
SHAPE RELATOR ASSOCIATION
SHAPE ASSOCIATION
SECTIONED AREA ASSOCIATION
VIEW SPECIFICATION

```
PROJECTED MODEL
SECTION PARAMETERS
SECURITY CLASSIFICATION
SELECTED MODEL GEOMETRY
SHEET APPROVAL
SHEET FORMAT
SHEET FORMAT SPECIFICATION
SHEET REVISION LOG
SHEET SIZE
SUBFIGURE DEFINITION
SYMBOL GROUP SPECIFICATION
TARGET AREA SIZE
TOL SPECIFICATION COMPARTMENT
TWO DIMENSIONAL RANGE
VIEW CLIP

*)
END_SCHEMA;  -- end of DRAFTING schema
(*
```

4.16 Product Structure

4.16.1 Introduction

Collected here are the Mechanical Products Applications.

```
*)
SCHEMA ipim_mechanical_product_schema;

  EXPORT EVERYTHING;
  ASSUME(ipim_pscm_schema);
(*
```

4.16.2 Product Structure Configuration Management

4.16.2.1 Introduction

```
*)
SCHEMA ipim_pscm_schema;

  EXPORT EVERYTHING;
  ASSUME(ipim_shape_interface_schema,
         ipim_material_schema,
         ipim_geometry_schema);
(*
```

4.16.2.1.1 Purpose

The purpose of the information model described herein is to portray the significant information required to manage the configuration (or structure) of an industrial product.

The information is independent of the type of product. Whether it be strictly mechanical, electrical, architectural, or of some other type, the information contained in the model remains unchanged.

The information portrayed in this model tends to portray the point in the product development life cycle near the completion of engineering detail design. This is often referred to as *engineering release*. The model is therefore independent of any particular design or analysis function.

4.16.2.1.2 Scope

The Product Structure Configuration Management information model is limited primarily to engineering data. It is assumed, however, that the product described will be manufactured (and probably supported by a logistics organization throughout its life).

The short term scope of this model is restricted by the following:

1. Product Structure is restricted to the relationship between physical components and assemblies.
2. Configuration Management applies only to Product Structure, i.e., manages the way assemblies and components are configured.
3. Product Structure will support more than one Bill-Of-Material view.

4. Product Structure and Configuration Management will support material information only with respect to stock items and make-from issues. This means that the model will not include material property data.

5. The primary goal is to capture the as-designed information. This is initially limited to mechanical product issues. There has been no attempt to integrate this model with work done by other committees such as AEC, FEM, or Electrical.

The model contained herein is intended to be a "conceptual" model, in that it attempts to record the minimal set of entities required to capture the desired information. It does NOT include those entities and relationships whose sole purpose is to make the model easier to use or more efficient at data storage, i.e. those desired at the "implementation" level, but unnecessary at the conceptual level. This was done in order to minimize the labor involved in the future integration of the model with other committees. Implementation, although important, can only be satisfactorily achieved after this integration process has occurred.

4.16.2.1.3 Viewpoint

The viewpoint represented in this data model is primarily that of configuration management of product structures. Engineering design and management are also strongly represented, however. The particular terms may vary depending on the nature of the enterprise, but the function is that of managing the way in which deliverable products are designed and configured.

4.16.2.1.4 Abbreviations and Acronyms

The following abbreviations and acronyms are used throughout this document.

CI — Configuration Item

CM — Configuration Management

CR — Change Request

PDD — Product Definition Data

PMD — Product Management Data

PS — Product Structure

PSCM — Product Structure Configuration Management

4.16.2.1.5 Fundamental Concepts and Assumptions

This section of the Product Structure Configuration Management Data Model contains underlying ideas found in the "real world" of configuration management. The ideas are important to the focus of the model, and they are represented either in the entities or the relationships of the model.

1. CM is interested in product (items) which are intended to be built and which may or may not be sold. This implies that the configuration of parts supplied by a vendor is not usually managed by CM. For example, if a certain motor is supplied by a vendor, the individual parts of the motor are not usually under CM.

2. The enterprise and the customer agree on the identity of certain configuration items (CI's). These are usually high level assemblies which act as focal points for managing the effectivity of lower level parts and assemblies. In other words, effectivity is managed by the configuration item. In some businesses, these configuration items are equated to Product Models.

3. An enterprise often makes minor changes to a product for marketing purposes. General "product lines" which include one or more (model) variations of a product are therefore maintained, often simultaneously.

4.16.2.2 PRODUCT ITEM

The generic description of one or more versions of a (thing) that is, or is intended to be, produced, or is consumed in a production process.

This entity identifies those **product item versions** which fulfill the generic description of the **product item**. Although one **product item version** may fulfill the generic requirements of more than one **product item**, common practice dictates that each is restricted so as to belong to one and only one **product item**.

This entity represents a prevalent concept that there are deviations in the defined characteristics of **product item versions** that are such that they represent variants of the same **product item**.

Every **product item** has at least one **product item version** (the "initial" version), and may have more than one.

It is important to note that the rules for determining if variations to a design are "small" enough to constitute membership within the same **product item** are company specific. A common rule is that small variations do not effect the fit, form, and function of the **product item** (all versions of a **product item** are interchangeable throughout all usages of it within higher assemblies). This determination is subjective, however, and varies greatly among companies and departments. It is also highly life-cycle dependent, as the creator normally does not know at the time of creation all of the higher assembly usages in which the item may be applied.

As such, this entity only captures information that at the time of creation the variations were deemed small, not that they are still considered to be small.

```
*)
ENTITY product_item;
  name         : STRING;
  description  : STRING;
  item_id      : STRING;
  versions     : LIST [1 : #] OF product_item_version;
UNIQUE
  item_id;
WHERE
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: The name of the **product item**.

description: A description of the **product item**.

item_id: The unique identification of the **product item**, which may take the form of of a part number, stock item number, etc.

versions: The list of versions of the product item.

PROPOSITIONS:

1. Item id must be UNIQUE.

4.16.2.3 PRODUCT ITEM VERSION

A specific description of an existing concept which defines the characteristics of a product item.
This description is commonly referred to as the "design", and aggregates information for a (thing) that is, or is intended to be, produced, or is consumed in a production process (e.g. a part, stock material item, etc.).
Although product item versions may share common information (especially those referred to by the same product item), each is a unique description for achieving a particular concept, be it small or large; each, by itself, is considered a separate design.
A product item version may be the result of a design change applied to a prior version. The prior design may or may not be a version of the same product item.

```
*)
ENTITY product_item_version;
  name            : STRING;
  description     : STRING;
  creator         : person_and_organization;
  creation_date   : date_time;
  security_class  : OPTIONAL security_classification_history;
  approvals       : OPTIONAL approval_history;
  prior_version   : OPTIONAL product_item_version;
  change_reason   : OPTIONAL STRING;
  version_id      : STRING;
  contracts       : OPTIONAL SET [1 : #] OF STRING;
WHERE
  ((prior_version = NULL) AND (change_reason = NULL)) OR
  ((prior_version <> NULL) AND (change_reason <> NULL));
  prior_version :<>: product_item_version;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: The name of the product item version.

description: The description of the product item version.

creator: The name of the creating organization (company, department, section, person, and/or project) for the product item version.

creation_date: The date and time of the creation of the product item version.

security_class: The Security Classification (if appropriate).

approvals: Approval history (if appropriate).

prior_version: The immediately prior design from which this version was derived.

change_reason: A description of the reason for the design change.

version_id: The identification of the product item version, such as a part version number.

contracts: The identifiers of contracts in which the product item version is called for.

```
*)
RULE product_item_and_version FOR (product_item,
                                   product_item_version);
  IF (instantiation(product_item_version, product_item) <> 1) THEN
    VIOLATION;
  (* A PRODUCT_ITEM_VERSION is associated with one PRODUCT_ITEM *)
  END_IF;
  REPEAT FOR EACH product_item_version IN product_item.versions;
    IF NOT (UNIQUE version_id) THEN
      VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. The prior version cannot be the same as the "current" version.
2. A product item version must be associated with a single product item.
3. If there is a prior design, then the reason for the design change must be given.
4. A version id must be unique within the context of a product item.

4.16.2.4 PRODUCT ITEM VERSION FUNCTIONAL DEFINITION

A product item version functional definition, or functional definition for short, is a unique description of the underlying relationships and information defining product item versions.

Functional definitions may be defined for many reasons, but probably the most common is for different organizational definitions of a product item version's parts list (bill of material).

As an example, the component parts list of a multi-level assembly may vary considerably depending on the particular purpose or organization for which it is intended, such as between as-designed, as-planned, or as-serviced.

Functional definitions may also be defined to provide for restricted or expanded definitions. Consider a primary design definition of an assembly which includes assembly component usage traversal Substitutes. The use of any particular substitute is considered to have no appreciable effect on the final product; the substitutes are interchangeable. However, different applications (analysis, inspection, process planning, etc.) may have greatly different results based on which substitute is used. em Functional definition allows (but does not require) each application to construct its own definitions for the assembly, which may have fewer or more assembly component usage occurrence substitutes allowed, and to relate the results of the application to those restricted or expanded definitions.

```
*)
ENTITY product_item_version_functional_definition;
  version          : product_item_version;
  primary_shape    : OPTIONAL shape;
  materials        : OPTIONAL SET [1 : #] OF material_property;
  owner            : person_and_organization;
  name             : STRING;
  description      : STRING;
  creation_date    : date_time;
WHERE
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

version: A product item version.

primary_shape: The primary geometric shape definition associated with with the product item version functional definition.

materials: The set of materials used in the product.

owner: The name of the owning organization (company, department, section, person, and/or project) for the product item version functional definition.

name: The textual name of the product item version functional definition.

description: A textual description of the product item version functional definition.

creation_date: The date and time of the creation of the product item version functional definition.

```
*)
RULE version_and_definition FOR (product_item_version,
                  product_item_version_functional_definition);
  IF instantiation(product_item_version,
                   product_item_version_functional_definition) < 1
    THEN VIOLATION;
  (* A Product Item Version must be associated with at least
     one Product Item Version Functional Definition *)
  END_IF;
END_RULE;
(*
```

PROPOSITIONS:

1. A product item version must be associated with at least one product item version functional definition.

4.16.2.5 PRODUCT ITEM USAGE TRAVERSAL

A product item usage traversal is the use of an instance of a product item version in the context of a higher level product item version (i.e. assembly component usage traversal, or in the context of an output product item version (i.e. make from usage option.

Product item usage-traversals are identified for a variety of reasons, the most predominant being for locating instances of a component within some higher level of assembly. They may also be established for functional reasons, when locations may or may not be known. A **product item usage traversal** is not constrained to always have a geometric relationship (location and orientation) relative to its context.

```
*)
ENTITY product_item_usage_traversal
  SUPERTYPE OF (make_from_usage_option XOR
                assembly_component_usage_traversal);
END_ENTITY;
(*
```

4.16.2.6 MAKE FROM USAGE OPTION

Establishes a relationship that a **product item version** (the input) within a given **functional definition** can be physically transformed into another **product item version** (the output).

A make-from relationship in reality states that any physical manifestation of one design (the output) can be manufactured from any physical manifestation of another design (the input). This relationship is independent of any particular instances of the physical manifestations, however, and so is established between the designs (**product item version functional definitions**).

The **functional definition** for the **product item versions** is necessary in that a single **make from usage option** within one **functional definition** may actually be seen as several **make from usage options** or sequences of options within another.

The input **product item versions** are those typically called "stock" items, although they are not restricted to this. The resulting **product item version** can be either a version of a new **product item** or a version of the input **product item**.

As an example, consider the case of a shaft which can be machined from either a casting or a forging. All three (the shaft, the forging, and the casting) are separate instances of **product item versions**, and two input attributes of **make from usage option** exist, one between the output shaft and the input forging, the other between the output shaft and the input casting.

```
*)
ENTITY make_from_usage_option
  SUBTYPE OF (product_item_usage_traversal);
  output       : product_item_version_functional_definition;
  input        : product_item_version_functional_definition;
  quantity     : INTEGER;
  criteria     : ranking_rationale;
  occ_location : OPTIONAL axis_placement;
WHERE
  output.version <> input.version;
  quantity > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

output: The output **functional definition**.

input: The input functional definition from which the output can be made.

quantity: The quantity of physical manifestations of the output which will be produced from a single physical manifestation of the input. This is typically one, although it may be more (Such as cutting one forging into two shafts).

criteria: A ranking and rationale of the preference for use of the input.

occ_location: The spatial location of OUTPUT with respect to the INPUT.

PROPOSITIONS:

1. Input and output Versions must be different.
2. Quantity must be greater than zero.

4.16.2.7 RANKING RATIONALE

A ranking and ranking rationale. The ranking is expressed as a positive integer, with the preference being inversely related to the integer value (i.e if one entity has a ranking of 2 and another has a ranking of 4, then the one ranked 2 is preferred to the one ranked 4).

```
*)
ENTITY ranking_rationale;
  ranking   : INTEGER;
  rationale : STRING;
WHERE
  ranking > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

ranking: A ranking of the preference for use

rationale: A description of the rationale used for ranking, such as "Cost" or "High Lead Time".

PROPOSITIONS:

1. Ranking must be greater than zero.

4.16.2.8 MAKE FROM USAGE OPTION GROUP

A collection of make from usage option instances which delimits one possible combination of the outputs which can be made from a single physical manifestation of an input. For instance, an input bar stock item D can be cut twice so as to create three output items x, y, and z. make from usage option instances Dx_1, Dy_1, and Dz_1 (subscripts denote the Traversal-ID) would all exist and be gathered into one make from usage option group. Multiple groups are possible for the same input item. For example, the input bar stock D may also be cut twice to produce two output items x and an output item$t$, which would be collected into a group of Dx_1, Dx_2, and Dt_1. Make from usage option instances may be shared between multiple groups.

```
*)
ENTITY make_from_usage_option_group;
  members : LIST [1 : #] OF make_from_usage_option;
WHERE
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

members: The list of members of the Group.

```
*)
RULE make_from_group_rule FOR (make_from_usage_option_group);
  REPEAT i := 2 TO SIZEOF(members);
    IF (members[i].input <> members[i-1].input) THEN
      VIOLATION;
    END_IF;
  END_REPEAT;
  REPEAT i := 1 TO SIZEOF(members) - 1;
    REPEAT j := i + 1 TO SIZEOF(members);
      IF (members[i].output = members[j].output) THEN
        VIOLATION;
      END_IF;
    END_REPEAT;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. All members of the Group must be produced from the same item.
2. All outputs from the members of the group must be different.

4.16.2.9 ASSEMBLY COMPONENT USAGE TRAVERSAL

A **product item usage traversal** in which the context **product item version** is any higher level (parent) node within the assembly hierarchy.

This entity is a supertype of both single level occurrences (normal assembly component relationships) and multi level occurrences (ancestor descendent relationships).

```
*)
ENTITY assembly_component_usage_traversal
  SUPERTYPE OF (next_assembly_usage_occurrence XOR
                higher_assembly_usage_traversal)
  SUBTYPE OF (product_item_usage_traversal);
  ancestor     : product_item_version_functional_definition;
  descendent   : product_item_version_functional_definition;
  occ_location : OPTIONAL axis_placement;
WHERE
```

```
  descendent.version <> ancestor.version;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

ancestor: A high level assembly.

descendent: A lower level assembly or component.

occ_location: The spatial location of **descendent** with respect to **ancestor**.

PROPOSITIONS:

1. ANCESTOR must not be the same as DESCENDENT.

4.16.2.10 ASSEMBLY COMPONENT USAGE TRAVERSAL SUBSTITUTE

A ranked substitute in an assembly component usage traversal.

```
*)
ENTITY assembly_component_usage_traversal_substitute;
  reference  : assembly_component_usage_traversal;
  substitute : assembly_component_usage_traversal;
  criteria   : ranking_rationale;
WHERE
  reference <> substitute;
  reference.ancestor = substitute.ancestor;
  reference.descendant <> substitute.descendant;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

reference: An assembly component usage traversal.

substitute: An assembly component usage traversal that may be used in place of **reference**.

criteria: The **ranking rationale**.

PROPOSITIONS:

1. **Reference** and **substitute** must be different.
2. **Reference** and **substitute** must have the same ancestor.
3. **Reference** and **substitute** must have different descendents.

4.16.2.11 NEXT ASSEMBLY USAGE OCCURRENCE

The representation of an organization of a product which relates a **product item version** to its immediate parent within an assembly hierarchy. That is, ANCESTOR and DESCENDENT are parent and child. This entity is used to derive the typical indented parts list for a product.

By sequentially tracing through the context of these occurrences (D is a child of B, B is a child of G, etc.), or in the other direction (G is the parent of B, B is the parent of D, etc.), it is possible to derive the usual indented parts list for a product. It is also possible to determine both where an item is used and what is used within it.

For explicit locations of each instance of a component relative to its context assembly, an instance of this entity must exist for each located instance of the component.

For simple parts list information the quantity is the count of components within the given context. Because each instance of the component is not normally separately identified, location information may not exist.

This model does not prohibit the inclusion of redundant information; both non-geometric, quantified occurrences for typical indented parts list information, and explicit occurrences for each instance of a component may exist, although the former can be derived from the latter.

```
*)
ENTITY next_assembly_usage_occurrence
  SUBTYPE OF (assembly_component_usage_traversal);
  quantity : INTEGER;
WHERE
  quantity > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

quantity: The count of sub-assemblies of this type within its parent assembly.

PROPOSITIONS:

1. QUANTITY must be at least one.

4.16.2.12 HIGHER ASSEMBLY USAGE TRAVERSAL

The unique identification of a **product item version** used within the context of a higher level parent than the immediate parent within an assembly hierarchy. That is, there is more than one "generation" separating ANCESTOR and DESCENDENT.

The most common use of this entity is to capture assembly requirements at some high level of assembly, where the identification of instances of some (much) lower level assembly or item is required. An example would be where part A in assembly B must mate within a specified tolerance with part C in assembly D, when assembly B and assembly D are used in assembly E. Two instances of higher assembly usage traversal would exist with assembly E as the context: one to uniquely identify the lower level part A, and the other for lower level part C.

This higher parent assembly is viewed as the top node of the assembly tree within the given context; any relationships or properties associated with the occurrence are not dependent on any higher level of assembly than the given context.

```
*)
ENTITY higher_assembly_usage_traversal
  SUBTYPE OF (assembly_component_usage_traversal);
  steps : LIST [2 : #] OF next_assembly_usage_occurrence;
WHERE
  ancestor = steps[1].ancestor;
  descendant = steps[SIZEOF(steps)].descendant;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

steps: An ordered list of the assemblies, starting at the ancestor and ending with the descendent, comprising the chain linking ancestor and descendent.

```
*)
RULE higher_assembly_usage_traversal_rule FOR
                                    (higher_assembly_usage_traversal);
  REPEAT i := 2 TO SIZEOF(steps);
    IF (ancestor.version = steps[i].ancestor.version) THEN
      VIOLATION;
    END_IF;
    IF (steps[i].ancestor.version <> steps[i-1].descendant.version)
    THEN VIOLATION;
    END_IF;
  END_REPEAT;
  REPEAT i := 1 TO SIZEOF(steps) - 1;
    REPEAT j := i + 1 TO SIZEOF(steps);
      IF (steps[i].ancestor.version = steps[j].ancestor.version)
      THEN VIOLATION;
      END_IF;
      IF (steps[i].descendant.version = steps[j].descendant.version)
      THEN VIOLATION;
      END_IF;
    END_REPEAT;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. The first and last entries in steps are ancestor and descendent.
2. The ancestor in step[i] must be the descendent in step[i-1].
3. No ancestor or descendent may occur more than once in the traversal.

4.16.2.13 PRODUCT MODEL

An identification of a collection of specific product features that are associated with marketing requirements.

Models of a product are essentially marketing ideas, but are used to carry through the physical item's identification.

Each manifestation (physical item) of a **product item version** is associated with one and only one **product model** (via the **configuration item**), although another manifestation (physical item) for the same **product item version** may be associated with a different **product model**.

```
*)
ENTITY product_model;
  name        : STRING;
  model_id    : STRING;
  components : OPTIONAL SET [1 : #] OF configuration_item;
UNIQUE
  model_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: The name of the **product model**.

model_id: The unique identification label for the **product model**, such as a sales model number.

components: The **configuration items** comprising the **product model**.

PROPOSITIONS:

1. **Model id** must be UNIQUE.

4.16.2.14 CONFIGURATION ITEM

A **configuration item** is the identification of a portion of a **product model** for the purpose of managing its configuration.

All configuration management (as planned, as built, etc.) is tracked against these **configuration items**.

A **configuration item** can be an entire **product model** or some portion thereof. It may be sold as part of other **configuration items** or by itself.

```
*)
ENTITY configuration_item;
  name              : STRING;
  configuration_id : STRING;
  item_units        : SET [0 : #] OF planned_physical_unit;
UNIQUE
  configuration_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: The name of the **configuration item**.

configuration_id: The unique identification label for the item (a configuration item number).

item_units: The planned physical units forming the configuration item.

```
*)
RULE product_model_and_configuration FOR (product_model,
                                          configuration_item);
  IF (instantiation(configuration_item, product_model)) <> 1
  THEN VIOLATION;
  (* A CONFIGURATION_ITEM is associated with a
     single PRODUCT_MODEL *)
  END_IF;
END_RULE;
(*
```

PROPOSITIONS:

1. Configuration id must be UNIQUE.
2. A configuration item is associated with a single product model.

4.16.2.15 PLANNED PHYSICAL UNIT

A physical unit which has not actually been produced, but has been identified for planning the production of one or more built physical units.

A planned physical unit identifies a single unit of product (such as a single serial or lot number) for which a given configuration is intended. This intended configuration is typically, but not always, specified by a "Configuration Control Board" or equivalent organization after the design phase of a product is complete.

Although all planned physical units are intended to be produced, some may never be.

Product item versions normally allow for many optional or substitute components, all of which fulfill the design's objective. OCCURRENCE EFFECTIVITY designates a subset of these optional or substitute components which are to be used when producing a physical manifestation of the description. As such, this entity allows for capturing a more precise intended configuration (assembly/component usage or make-from usage) for a physical unit than that allowed within the description of the product item version's definition.

The ID of a planned physical unit is the serial or lot number. The ID and the referenced design (product item version) must form a unique pair. Multiple planned units can have the same ID only if they refer to different designs. An example: Part A can have serial numbers 001 and 002, but not two serial numbers 001. Part B may also have serial numbers 001 and 002, but these must be different planned physical units than those for Part A.

```
*)
ENTITY planned_physical_unit
  SUPERTYPE OF (discrete_planned_unit XOR
                lot_planned_unit);
  physical_unit_id        : STRING;
  design                  : product_item_version;
  configuration_manager   : person_and_organization;
  occurrence_effectivity  : SET [1 : #] OF
                                product_item_usage_traversal;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

physical_unit_id: The identification label for the **planned physical unit** (e.g the serial or lot number).

design: The **product item version** planned to be produced.

configuration_manager: The name and organization of the person responsible for managing the configuration of the planned unit.

occurrence_effectivity: Information that a distinct **product item usage occurrence** of a **product item version** is supposed to appear in specific **planned physical units.**

```
*)
RULE planned_unit_and_configuation_item FOR (planned_physical_unit,
                                             configuration_item);
  IF (instantiation(planned_physical_unit, configuration_item)) <> 1
  THEN VIOLATION;
  (* A PLANNED PHYSICAL UNIT is associated with no more than
     one CONFIGURATION ITEM *)
  END_IF;
END_RULE;
(*

*)
RULE planned_unit_and_piv FOR (planned_physical_unit,
                               product_item_version);
  LOCAL
    k : INTEGER := instantiation(MODEL,planned_physical_unit);
  END_LOCAL;
  REPEAT i := 1 TO k - 1;
    REPEAT j := i + 1 TO k;
      IF (planned_physical_unit[i].design =
          planned_physical_unit[j].design) AND
         (planned_physical_unit[i].physical_unit_id =
          planned_physical_unit[j].physical_unit_id) THEN
        VIOLATION;
      END_IF;
    END_REPEAT;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

1. A **planned physical unit** is associated with one **configuration item.**

2. The values of the PHYSICAL UNIT ID and DESIGN attributes must form a unique pair.

4.16.2.16 DISCRETE PLANNED UNIT

A physical unit that is planned to be produced individually. This entity is normally used for "serialized" items, where each instance of an item is given a unique serial number.

```
*)
ENTITY discrete_planned_unit
  SUBTYPE OF (planned_physical_unit);
  open_ended_range : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

open_ended_range: Indication of whether or not this is the start of an open ended range.

PROPOSITIONS:

4.16.2.17 LOT PLANNED UNIT

Physical units planned to be produced as a batch. Lots are used where physical units are planned to be produced in "batches", and where characteristics which vary between the lots (such as a production method used or facility where created) are of importance, but not between individual members of a lot.

```
*)
ENTITY lot_planned_unit
  SUBTYPE OF (planned_physical_unit);
  lot_size : INTEGER;
WHERE
  lot_size > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

lot_size: The planned lot quantity.

PROPOSITIONS:

1. The lot size must be greater than zero.

4.16.2.18 PSCM Classification Structure

The following indented listing provides the Classification Structure for the PSCM Integrated Model.

```
ASSEMBLY COMPONENT USAGE TRAVERSAL SUBSTITUTE
CONFIGURATION ITEM
MAKE FROM USAGE OPTION GROUP
PLANNED PHYSICAL UNIT
  DISCRETE PLANNED UNIT
  LOT PLANNED UNIT
PRODUCT ITEM
PRODUCT ITEM USAGE TRAVERSAL
  ASSEMBLY COMPONENT USAGE TRAVERSAL
```

```
    HIGHER ASSEMBLY USAGE TRAVERSAL
    NEXT ASSEMBLY USAGE OCCURRENCE
  MAKE FROM USAGE OPTION
PRODUCT ITEM VERSION
PRODUCT ITEM VERSION FUNCTIONAL DEFINITION
PRODUCT MODEL
RANKED ASSEMBLY COMPONENT USAGE TRAVERSAL
RANKING RATIONALE

*)
  END_SCHEMA; -- end of PSCM schema
END_SCHEMA; -- end MECHANICAL PRODUCTS schema
(*
```

4.17 AEC Applications

```
*)
SCHEMA ipim_aec_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_aec_core_schema,
         ipim_ship_structure_schema);
(*
```

4.17.1 AEC Core Model

4.17.1.1 Editorial Introduction

```
*)
SCHEMA ipim_aec_core_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_geometry_schema,
         ipim_shape_interface_schema,
         ipim_resources_schema,
         ipim_material_schema,
         ipim_drafting_alt1_schema,
         ipim_fem_schema);

(*
```

4.17.1.2 AEC TYPE definitions

Resource types for this schema.

4.17.1.2.1 SHAPE REPR

```
*)
TYPE shape_repr = SELECT
     (geometric_model,
      drawing,
      finite_element_model);
END_TYPE;
(*
```

4.17.1.2.2 MATERIAL REPR

```
*)
TYPE material_repr = SELECT
     (material_property);
END_TYPE;
(*
```

4.17.1.2.3 STATUS

```
*)
TYPE status = ENUMERATION OF
     (rejected,
      alternative,
      selected);
END_TYPE;
(*
```

4.17.1.2.4 ALLOWED PARAMETER DOMAIN LIST

```
*)
TYPE allowed_parameter_domain_list = LIST [1:#] OF
     allowed_parameter_domain;
END_TYPE;
(*
```

4.17.1.2.5 ACTUAL PARAMETER LIST

```
*)
TYPE actual_parameter_list = LIST [1:#] OF
     actual_parameter;
END_TYPE;
(*
\end{verbatim

\sssssclause{CURRENCY ENUM}

%%\mnote{Referenced but not defined.}
\begin{verbatim}
*)
TYPE currency_enum = ENUMERATION OF
     (ecu,
      dollar_us);
END_TYPE;
(*
```

4.17.1.3 Function Definitions

4.17.1.3.1 ACTUAL MEET FORMAL

```
*)
FUNCTION actual_meet_formal(f_par: allowed_parameter_domain_list;
                            a_par: actual_parameter_list): LOGICAL;
LOCAL
  i, j             : INTEGER;
  result, matched : LOGICAL;
END_LOCAL;
  (* Check if a list of actual parameters matches the
```

```
    * formal specification. Note that parameters are not positional.
    *
    * Perform a loop over both lists and check.
    *)
   result := FALSE;
   IF (SIZEOF(f_par) = SIZEOF(a_par)) THEN
     i := 1; result := TRUE;
     REPEAT WHILE (i <= SIZEOF(f_par) AND result);
       matched := FALSE;
       REPEAT WHILE (j <= SIZEOF(a_par) AND NOT matched);
         matched := (a_par[j].formal_par = f_par[i]);
       END_REPEAT;
       result := matched;
     END_REPEAT;
   END_IF;
   RETURN(result);
END_FUNCTION;
(*
```

4.17.1.3.2 IN DOMAIN

```
*)
FUNCTION in_domain(f_par: allowed_parameter_domain;
                   val: GENERIC): LOGICAL;
LOCAL
  valid    : LOGICAL;
END_LOCAL;
  (* Check if the value of an actual parameter lies in
   * the specified domain of the formal parameter.
   *
   * First check discrete values then check value-ranges.
   *)
  valid := FALSE;
  CASE TYPEOF(val) OF
    INTEGER:
      BEGIN
        REPEAT WHILE (i < SIZEOF(f_par.discretes) AND NOT valid);
          valid := (f_par.discretes[i] = val);
        END_REPEAT;

        (* Check ranges if value isn't valid for discrete values.
         *)
        REPEAT WHILE (i < SIZEOF(f_par.domains) AND NOT valid);
          valid := (f_par.int_domain[i].low_bound >= val AND
                    val <= f_par.int_domain[i].high_bound);
        END_REPEAT;
      END;
    REAL:
      BEGIN
```

```
        REPEAT WHILE (i < SIZEOF(f_par.discretes) AND NOT valid);
          valid := (f_par.discretes[i] = val);
        END_REPEAT;
        (* Check ranges if value isn't valid for discrete values. *)
        REPEAT WHILE (i < SIZEOF(f_par.domains) AND NOT valid);
          valid := (f_par.int_domain[i].low_bound >= val AND
                    val <= f_par.int_domain[i].high_bound);
        END_REPEAT;
      END;
    STRING:
      BEGIN
        REPEAT WHILE (i < SIZEOF(f_par.discretes) AND NOT valid);
          valid := (f_par.discretes[i] = val);
        END_REPEAT;
      END;
  END_CASE;
  RETURN(valid);
END_FUNCTION;
(*
```

4.17.1.4 Product Definition Unit and Aspect

4.17.1.5 PRODUCT DEFINITION UNIT

The product definition unit has only two subtypes to stage in this version of the Express model, the functional unit and technical solution. Other stages of the products lifecycle are not yet addressed. A product definition unit has characteristics; this relation with characteristic is however not defined here, but, after stage-discrimination, with functional unit (has required characteristics) and technical solution (has expected characteristics).

```
*)
ENTITY product_definition_unit
  SUPERTYPE OF (functional_unit AND
                technical_solution);
END_ENTITY;
(*
```

4.17.1.6 CHARACTERISTIC

Only the "as required" and "as designed" stages are currently covered. If extended with "as built" and subsequent stages, this entity becomes also the supertype of measured characteristic.

```
*)
ENTITY characteristic
  SUPERTYPE OF (required_characteristic AND
                expected_characteristic);
  is_given_for : aspect;
  relates_to   : OPTIONAL LIST [1:?] OF environmental_agent;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

is_given_for:

relates_to:

4.17.1.7 ENVIRONMENTAL AGENT

In this version only one Product-type specific collection of agents is identified: for Buildings we have the list of Agents defined in IS0 6241.

```
*)
ENTITY environmental_agent
  SUPERTYPE OF (agent_6241);
END_ENTITY;
(*
```

4.17.1.8 AGENT 6241

```
*)
ENTITY agent_6241
  SUPERTYPE OF (mechanical_agent AND
                electronical_agent AND
                thermal_agent AND
                chemical_agent AND
                biological_agent)
  SUBTYPE OF (environmental_agent);
END_ENTITY;
(*
```

4.17.1.9 MECHANICAL AGENT

```
*)
ENTITY mechanical_agent
  SUBTYPE OF (agent_6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.10 ELECTRONICAL AGENT

```
*)
ENTITY electronical_agent
  SUBTYPE OF (agent_6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.11 THERMAL AGENT

```
*)
ENTITY thermal_agent
  SUBTYPE OF (agent_6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.12 CHEMICAL AGENT

```
*)
ENTITY chemical_agent
  SUBTYPE OF (agent_6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.13 BIOLOGICAL AGENT

```
*)
ENTITY biological_agent
  SUBTYPE OF (agent_6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.14 ASPECT

In this version only one Product-type specific collection of aspects is identified: for Buildings we have the list of Aspects defined in ISO 6241 and, one a more detailed level, ISO 6242.

```
*)
ENTITY aspect
        SUPERTYPE OF (ISO6241);
END_ENTITY;
(*
```

4.17.1.15 ISO6241

As an example to show how more detailed definitions of requirements can be described we have chosen here for a translation of ISO/DP6241 into EXPRESS. A table of these requirements is published in the 'CIB masterlist', CIB report 18, 1983. Users of the AEC reference model are requested to suggest additional options for requirements which are applicable to their situation. The AEC reference model is intended to allow the use of a variety of company, national and/or international standards. Each of these standards should be supplied as an EXPRESS schema by the developer of that standard. They are not considered being part of the this standard, and are therefore not included in the AEC reference model. However below is shown how additional standards can be tied to the general AEC reference model. Consider the following break down of the ISO6241 entity.

```
*)
ENTITY ISO6241
  SUPERTYPE OF (stability_6241 OR
                fire_safety_6241 OR
                safety_6241 OR
                thightness_6241 OR
                hygro_thermal_6242 OR
                air_purity_6242 OR
                acoustical_6242 OR
                visual_6242 OR
                tactile_6241 OR
                anthropodynamic_6241 OR
                hygiene_6241 OR
                specific_use_6241 OR
                duralibilty_6241 OR
                economic_6241 OR
                energy_6242)
  SUBTYPE OF (aspect);
END_ENTITY;
(*
```

4.17.1.16 ECONOMIC 6241

For example economic requirements can be broken down further into capital, running and maintenance costs. For each of these entities a unit of measurement can be chosen, e.g.: US$ or ECU (european currency unit). The value-domain of the actual requirement-value can be specified using the 'functional par' attribute which is inherited from the entity 'requirement aspect'. So further break down results in the following entities.

```
*)
ENTITY economic_6241
  SUPERTYPE OF (capital_cost AND
                running_cost AND
                maintenance_cost)
  SUBTYPE OF (ISO6241);
END_ENTITY;
(*
```

4.17.1.17 CAPITAL COST

```
*)
ENTITY capital_cost
  SUBTYPE OF (economic_6241);
  currency      : currency_enum;
  currency_date : date_time;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

currency:

currency_date:

4.17.1.18 RUNNING COST

```
*)
ENTITY running_cost
  SUBTYPE OF (economic_6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.19 MAINTENANCE COST

```
*)
ENTITY maintenance_cost
  SUBTYPE OF (economic_6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.20 ENERGY 6242

A similar breakdown is given for energy-use by buildings, and based on ISO 6242. Other aspects mentioned in this standard are not modelled in Express, since these entities form only an example.

```
*)
ENTITY energy_6242
  SUPERTYPE OF (energy_use_6242 OR
                heat_transfer_6242)
  SUBTYPE OF (iso6241);
END_ENTITY;
(*
```

4.17.1.21 ENERGY USE 6242

```
*)
ENTITY energy_use_6242
  SUBTYPE OF (energy_6242);
  (* C = global energy use coefficient, expressed in watts per cubic
  metre per degree Celsius, where cubic metre refers to the volume
  enclosed by the building and watts refers to constant conditions
  to maintain a temperature difference of 1 degree Celsius between
  the interior and exterior of the building, when the latter is
  colder. *)
WHERE
  (* C must be a REAL *)
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:
PROPOSITIONS:

4.17.1.22 HEAT TRANSFER 6242

```
*)
ENTITY heat_transfer_6242
  SUBTYPE OF (energy_6242);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.23 STABILITY 6241

```
*)
ENTITY stability_6241
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.24 FIRE SAFETY 6241

```
*)
ENTITY fire_safety_6241
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.25 SAFETY 6241

```
*)
ENTITY safety_6241
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.26 THIGHTNESS 6241

```
*)
ENTITY thightness_6241
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.27 HYGRO THERMAL 6242

```
*)
ENTITY hygro_thermal_6242
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.28 AIR PURITY 6242

```
*)
ENTITY air_purity_6242
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.29 ACOUSTICAL 6242

```
*)
ENTITY acoustical_6242
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.30 VISUAL 6242

```
*)
ENTITY visual_6242
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.31 TACTILE 6241

```
*)
ENTITY tactile_6241
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.32 ANTHROPODYNAMIC 6241

```
*)
ENTITY anthropodynamic_6241
```

```
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.33 HYGIENE 6241

```
*)
ENTITY hygiene_6241
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.34 SPECIFIC USE 6241

```
*)
ENTITY specific_use_6241
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.35 DURALIBILTY 6241

```
*)
ENTITY duralibilty_6241
  SUBTYPE OF (iso6241);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.36 FUNCTIONAL UNIT

Functional unit can be discriminated to level into generic functional unit, specific functional unit and functional unit occurrence. A functional unit is defined by a set of required characteristics. Its meaning can be defined by a reference to a functional unit classification.

```
*)
ENTITY functional_unit
  SUPERTYPE OF (generic_functional_unit AND
                specific_functional_unit AND
                functional_unit_occurrence)
  SUBTYPE OF (product_definition_unit);
  function_class : OPTIONAL functional_unit_classification;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

function_class: The

4.17.1.37 FUNCTIONAL UNIT CLASSIFICATION

Meaning can be given to a **functional unit** by reference to and use of a classification technique. Only reference to CI/Sfb is now included. Users are free to use the classification technique which they prefer, and may add other functional classification techniques where appropriate. Sfb1 means that only table 1 is used for functional units.

```
*)
ENTITY functional_unit_classification
  SUPERTYPE OF (Sfb1);
END_ENTITY;
(*
```

4.17.1.38 SFB1

The Sfb code is in this version included as a sequence of one to three integers, with a value between 0 and 100. This means that the code is regarded as data, and is not (yet) interpreted as an entity with a specific meaning.

```
*)
ENTITY Sfb1
  SUBTYPE OF (functional_unit_classification);
  code : LIST[1:3] OF Sfb_code;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

code: The

4.17.1.39 SFB CODE

```
*)
ENTITY Sfb_code;
  code : INTEGER;
WHERE
  {1 <= code <= 99};
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

code: The

PROPOSITIONS:

1. Code must be greater than zero and less than 100.

4.17.1.40 GENERIC FUNCTIONAL UNIT

Functional unit described in general terms, such as used in regulations, standards, etc. They can be regarded as parametrically defined Units. If all parameters are known, it becomes a specific functional unit. Generic functional units may form a hierarchy, if several levels of generality are defined. This hierarchy is modelled by means of the entity generic functional unit structure.

```
*)
ENTITY generic_functional_unit
  SUBTYPE OF (functional_unit);
  structure : generic_functional_unit_structure;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

structure: The

4.17.1.41 GENERIC FUNCTIONAL UNIT STRUCTURE

```
*)
ENTITY generic_functional_unit_structure;
  is_child_of  : OPTIONAL generic_functional_unit;
  is_parent_of : OPTIONAL LIST [1:#] OF generic_functional_unit;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

is_child_of:

is_parent_of:

4.17.1.42 SPECIFIC FUNCTIONAL UNIT

A specific functional unit is defined by reference to functional requirements. It inherits the characteristics of a generic functional unit.

```
*)
ENTITY specific_functional_unit
  SUBTYPE OF (functional_unit);
  is_derived_from      : generic_functional_unit;
  is_characterised_by : OPTIONAL SET [1:#] OF
                                  required_characteristic;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

is_derived_from:

is_characterised_by:

4.17.1.43 REQUIRED CHARACTERISTIC

The **required characteristic** entity makes use of the **allowed parameter domain** entity to specify parameters and their allowed values. The remarks given for the use of **allowed parameter domain** with **generic technical solution**, are also valid here: the parameters themselves are not semantically defined by the General AEC Reference Model, but additional sets and/or classifications for specific application areas are to be defined by future versions of the model. Through the current flexible specification it is possible to make use of this entity by regarding the name and meaning of the parameters as data.

```
*)
ENTITY required_characteristic
  SUBTYPE OF (characteristic);
  should_be_within : allowed_parameter_domain;
  given_for        : aspect;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

should_be_within:

given_for:

4.17.1.44 FUNCTIONAL UNIT OCCURRENCE

The **functional unit occurrence** is an instance of a **specific functional unit.**

```
*)
ENTITY functional_unit_occurrence
  SUBTYPE OF (functional_unit);
  specific_fu : specific_functional_unit;
  is_part_of  : technical_solution_occurrence;
  located_by  : OPTIONAL place;
  connection  : aec_node;
DERIVE
  owner       : person_and_organization := is_part_of.owner;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

specific_fu:

is_part_of:

located_by:

connection:

owner:

4.17.1.45 TECHNICAL SOLUTION

```
*)
ENTITY technical_solution
  SUPERTYPE OF (generic_technical_solution AND
                specific_technical_solution AND
                technical_solution_occurrence)
  SUBTYPE OF (product_definition_unit);
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.46 PORT OCCURRENCE

A technical solution has ports, through which it can be connected with other technical solutions. The port occurrence refers to a specific port and through that to all the relevant information contained by the specific port, such as the orientation vectors and the offset value. In addition, a port occurrence refers to an End of the functional unit occurrence which uses the technical solution. A port occurrence may copy the decomposition into Free Ends described on the Generic and Specific levels, so that every unique interface between components of the technical solution with components of other technical solutions can be described.

```
*)
ENTITY port_occurrence;
  belongs_to       : technical_solution_occurrence;
  is_described_by  : specific_port;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

belongs_to:

is_described_by:

4.17.1.47 SPECIFIC PORT

Port defined on a specific level. The number of ports, their position and orientation are known. Each port has two orientation-vectors and an attribute Off-set value, which defines the location of the origin of the port. This origin is positioned on the primary orientation-vector of the port.

```
*)
ENTITY specific_port;
  belongs_to       : specific_technical_solution;
  is_derived_from  : generic_port;
  offset           : REAL;
  p_vector         : vector;
  s_vector         : vector;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

belongs_to:

is_derived_from:

offset:

p_vector:

s_vector:

4.17.1.48 GENERIC TECHNICAL SOLUTION

A **generic technical solution** describes a parameterised object, independent of its representation. It refers to one or many parametric (or procedural) descriptions, each for a specific representation type. Since version 1 of this Standard does not cover parametric/procedural product definitions, this part of the model cannot be based on this Standard itself. It is however possible to use national standards and/or modelling languages of CAD-systems until this aspect can be handled by this Standard too.

The parameters are defined by the generic entity **allowed parameter domain**. The current approach sees the name (and meaning) of parameters as data. It makes the AEC reference model in this respect flexible for adaption to various application areas. The need to capture also the meaning of parameters in the conceptual model is however recognised. This will be solved by the development and use of parameter sets or classifications for specific application areas in future versions of the AEC reference model.

```
*)
ENTITY generic_technical_solution
 SUBTYPE OF (technical_solution);
  owner          : person_and_organization;
  described_by   : procedural_descr;
  formal_pars    : OPTIONAL LIST [1:#] OF allowed_parameter_domain;
  structure      : generic_technical_solution_structure;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

owner:

described_by:

formal_pars:

structure:

4.17.1.49 PROCEDURAL DESCR

```
*)
ENTITY procedural_descr;
  atts : undefined;
END_ENTITY;
(*
```

4.17.1.50 GENERIC TECHNICAL SOLUTION STRUCTURE

```
*)
ENTITY generic_technical_solution_structure;
  is_child_of  : OPTIONAL generic_technical_solution;
  is_parent_of : OPTIONAL LIST [1:#] OF generic_technical_solution;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

is_child_of:

is_parent_of:

4.17.1.51 SPECIFIC TECHNICAL SOLUTION

A specific technical solution is an implicit definition of an object (feature, joint, part or assembly). It is defined by a set of parameter values, which have to match the formal parameters defined by its corresponding generic technical solution. A specific technical solution can be defined explicitly within one or more models (being 2D drawings, solids, wireframes, FEM models, etc.)

```
*)
ENTITY specific_technical_solution
 SUBTYPE OF (technical_solution);
  owner               : person_and_organization;
  is_derived_from     : generic_technical_solution;
  is_represented_by : LIST [1:#] OF
                           explicit_technical_solution_definition;
  actual_pars         : OPTIONAL LIST [1:#] OF actual_parameter;
  port                : OPTIONAL LIST [1:#] OF specific_port;
WHERE
  actual_meet_formal(generic_ts.formal_pars, actual_pars);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

owner:

is_derived_from:

is_represented_by:

actual_pars:

port:

```
*)
RULE specific_technical_solution_constraint FOR
     (specific_technical_solution);
  LOCAL
```

```
    i : INTEGER;
  END_LOCAL;
  REPEAT i := 1 TO SIZEOF(actual_pars);
   IF  (actual_pars[i].domain_check <> TRUE) THEN
     VIOLATION;
    END_IF;
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

4.17.1.52 TECHNICAL SOLUTION OCCURRENCE

```
*)
ENTITY technical_solution_occurrence
 SUBTYPE OF (technical_solution);
  status                : technical_solution_decision;
  is_described_by       : specific_technical_solution;
  is_represented_by     : LIST [1:#] OF
                    explicit_technical_solution_definition_instance;
  is_connected_through : OPTIONAL LIST [1:#] OF port_occurrence;
  fulfils              : functional_unit_occurrence;
DERIVE
  owner       : person_and_organization := is_described_by.owner;
  using_owner : person_and_organization := status.taken_by;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

status:

is_described_by:

is_represented_by:

is_connected_through:

fulfils:

owner:

using_owner:

4.17.1.53 EXPECTED CHARACTERISTIC

```
*)
ENTITY expected_characteristic
  SUBTYPE OF (characteristic);
  given_for : aspect;
  result    : actual_parameter;
```

```
DERIVE
  conforms  : LOGICAL := result.domain_check;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

given_for:

result:

conforms:

4.17.1.54 EXPLICIT TECHNICAL SOLUTION DEFINITION

This entity represents the specific technical solution in an application-, view- and/or aspect-dependent product model. This can for instance be a representation by a line in a wireframe model, a symbol in a drawing, a volume in a solid model. etc.

```
*)
ENTITY explicit_technical_solution_definition;
  is_part_of : product_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

is_part_of: The

4.17.1.55 EXPLICIT TECHNICAL SOLUTION DEFINITION INSTANCE

```
*)
ENTITY explicit_technical_solution_definition_instance;
  is_part_of       : product_model;
  is_evaluated_by : analysis;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

is_part_of:

is_evaluated_by:

4.17.1.56 PRODUCT ASPECT MODEL

A product aspect model contains shape and material definitions of the product, and is referred to by technical solutions. Product aspect models are considered here as application-, view and/or aspect- dependent; technical solutions are independent of application and aspect, and link therefore the various product representations together. A product model can derived from another; for instance, a B-rep model can be derived from a CSG-rep model, and a 2D drawing can be derived from a B-rep model.

```
*)
ENTITY product_aspect_model;
  shape            : shape_repr;
  material         : material_repr;
  is_derived_from  : OPTIONAL product_aspect_model;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

shape:

material:

is_derived_from:

4.17.1.57 TECHNICAL SOLUTION DECISION

A Decision is taken by a person and organization (this is one of the miscellaneous entities) on a certain date. It defines the status of a technical solution occurrence. A decision can be based on zero, one or many Arguments.

```
*)
ENTITY technical_solution_decision;
  alternatives : OPTIONAL LIST [1:#] OF
                          technical_solution_occurrence;
  selected     : OPTIONAL technical_solution_occurrence;
  rejected     : OPTIONAL LIST [1:#] OF
                          technical_solution_occurrence;
  user         : functional_unit_occurrence;
  date         : date;
  arguments    : OPTIONAL SET [1:#] OF technical_solution_argument;
  approvals    : OPTIONAL SET [1:#] OF aec_approval;
DERIVE
  taken_by : person_and_organization := user.owner;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

alternatives:

selected:

rejected:

user:

date:

arguments:

approvals:

taken_by:

4.17.1.58 TECHNICAL SOLUTION ARGUMENT

An argument is based on the comparison of an **analysis** result and a corresponding **required characteristic**. The comparison is executed by the **in domain** function.

```
*)
ENTITY technical_solution_argument;
  based_on    : required_characteristic;
  analysis    : expected_characteristic;
  comment     : OPTIONAL STRING;
DERIVE
  evaluation : LOGICAL := analysis.conforms;
WHERE
  based_on.given_for = analysis.given_for;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

based_on:

analysis:

comment:

evaluation:

PROPOSITIONS:

4.17.1.59 ANALYSIS

This entity represents an analysis (by calculation, simulation or any other method) done by an expert to judge the quality of a **technical solution**. An analysis to one or more results, named **expected characteristics**.

```
*)
ENTITY analysis;
  done_by     : expert;
  day_time    : date_time;
  determines : expected_characteristic;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

done_by:

day_time:

determines:

4.17.1.60 EXPERT-

```
*)
ENTITY expert;
  is : person_and_organization;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

is: The

4.17.1.61 AEC APPROVAL

```
*)
ENTITY aec_approval;
  given_by : expert;
  for_ts   : technical_solution_occurrence;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

given_by:

for_ts:

4.17.1.62 ALLOWED PARAMETER DOMAIN

The allowed parameter domain defines the allowed value of a parameter. The domain may consist of a range of discrete values or value domains with an upper and lower boundary.

```
*)
ENTITY allowed_parameter_domain
  SUPERTYPE OF (int_formal AND
                real_formal AND
                logic_formal AND
                string_formal);
  description : STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

description: The

4.17.1.63 INT FORMAL

An actual parameter may be valid for certain values or within one or more ranges of values. the following entities define the parameter domain.

```
*)
ENTITY int_formal
  SUBTYPE OF (allowed_parameter_domain);
  discretes : OPTIONAL SET [1:#] OF INTEGER;
  domains   : OPTIONAL SET [1:#] OF int_domain;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

discretes:

domains:

4.17.1.64 INT DOMAIN

```
*)
ENTITY int_domain;
  low_bound  : INTEGER;
  high_bound : INTEGER;
WHERE
  low_bound < high_bound;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

low_bound:

high_bound:

PROPOSITIONS:

1. The low bound must be less than the high bound.

4.17.1.65 REAL FORMAL

```
*)
ENTITY real_formal
  SUBTYPE OF (allowed_parameter_domain);
  discretes : OPTIONAL SET [1:#] OF REAL;
  domains   : OPTIONAL SET [1:#] OF real_domain;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

discretes:

domains:

4.17.1.66 REAL DOMAIN

```
*)
ENTITY real_domain;
  low_bound  : REAL;
  high_bound : REAL;
WHERE
  low_bound < high_bound;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

low_bound:

high_bound:

PROPOSITIONS:

1. The low bound must be less than the high bound.

4.17.1.67 LOGIC FORMAL

```
*)
ENTITY logic_formal
  SUBTYPE OF (allowed_parameter_domain);
END_ENTITY;
(*
```

4.17.1.68 STRING FORMAL

```
*)
ENTITY string_formal
  SUBTYPE OF (allowed_parameter_domain);
  discretes : OPTIONAL SET [1:#] OF STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

discretes: The

4.17.1.69 ACTUAL PARAMETER

The actual parameters can be represented by the following entity. Only base-type values are allowed so far.

```
*)
ENTITY actual_parameter
  SUPERTYPE OF (int_val AND
                real_val AND
```

```
                logic_val AND
                string_val);
  formal_par : allowed_parameter_domain;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

formal_par: The

4.17.1.70 INT VAL

Entities representing actual-values. A generic type isn't allowed in EXPRESS.

```
*)
ENTITY int_val
  SUBTYPE OF (actual_parameter);
  value         : INTEGER;
DERIVE
        (* Test validity of value *)
  domain_check : LOGICAL := in_domain(formal_par, value);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

value:

domain_check:

4.17.1.71 REAL VAL

```
*)
ENTITY real_val
  SUBTYPE OF (actual_parameter);
  value         : REAL;
DERIVE
        (* Test validity of value *)
  domain_check : LOGICAL := in_domain(formal_par, value);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

value:

domain_check:

4.17.1.72 LOGIC VAL

```
*)
ENTITY logic_val
  SUBTYPE OF (actual_parameter);
  value          : LOGICAL;
DERIVE
  domain_check : LOGICAL := TRUE;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

value:

domain_check:

4.17.1.73 STRING VAL

```
*)
ENTITY string_val
  SUBTYPE OF (actual_parameter);
  value          : STRING;
DERIVE
        (* Test validity of value *)
  domain_check : LOGICAL := in_domain(formal_par, value);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

value:

domain_check:

4.17.1.74 Functional Network and Topology TYPE Definitions

4.17.1.74.1 ADJACENCY TYPE

```
*)
TYPE adjacency_type = ENUMERATION OF
      (domain,
       boundary);
END_TYPE;
(*
```

4.17.1.75 DIMENSIONAL DIRECTION

```
*)
TYPE dimensional_direction = ENUMERATION OF
      (ext,
```

```
      int);
END_TYPE;
(*
```

4.17.1.75.1 CONNECTIVITY STATUS

```
*)
TYPE connectivity_status = ENUMERATION OF
      (free,
       mated);
END_TYPE;
(*
```

4.17.1.75.2 DIMENSIONAL ORDER

```
*)
TYPE dimensional_order = INTEGER;
END_TYPE;
(*
```

4.17.1.75.3 CONNECTOR

```
*)
TYPE connector = SELECT
      (interface,
       port_occurrence);
END_TYPE;
(*
```

4.17.1.75.4 SIDE OR REGION

```
*)
TYPE side_or_region = SELECT
      (side,
       aec_region);
END_TYPE;
(*
```

4.17.1.75.5 CONNECTION ENUM

```
*)
TYPE connection_enum = ENUMERATION OF
      (functional,
       topological);
END_TYPE;
(*
```

4.17.1.75.6 NODE END LIST

```
*)
TYPE node_end_list = LIST [1:#] OF node_end;
END_TYPE;
(*
```

4.17.1.76 Functional Network and Topology FUNCTION Definitions

4.17.1.76.1 NUMBER OF DOMAIN EXT

```
*)
FUNCTION number_of_domain_ext(node_ends : node_end_list): INTEGER;
  LOCAL
    i, domain_ext_n : INTEGER;
  END_LOCAL;
  domain_ext_n := 0;
  REPEAT i := 1 TO SIZEOF(node_ends);
    IF (node_ends[i].adj_type = domain AND
        node_ends[i].direction = ext) THEN
      domain_ext_n := domain_ext_n + 1;
    END_IF;
  END_REPEAT;
  RETURN(domain_ext_n);
END_FUNCTION;
(*
```

4.17.1.76.2 GET CONNECTIVITY STATUS

```
*)
FUNCTION get_connectivity_status(is_connected_via : connector):
                                  connectivity_status;
  IF (TYPEOF(is_connected_via) = interface) THEN
    RETURN(mated);
  ELSE
    IF (TYPEOF(is_connected_via) = port_occurrence) THEN
      RETURN(free);
    END_IF;
  END_IF;
END_FUNCTION;
(*
```

4.17.1.76.3 CONNECTS TYPE

```
*)
FUNCTION connects_type(connects:side_or_region): adjacency_type;
  IF (TYPEOF(connects) = side) THEN
    RETURN(boundary);
  ELSE
    RETURN(domain);
```

```
  END_IF;
END_FUNCTION;
(*
```

4.17.1.77 Functional Network

The functional network defines the functional relationships between the **functional units** which are part of one **technical solution**. This network is based on three entities: **aec node**, **node end** and **interface**. Meta-topology enriches the general network in the direction of topology/ geometry. Therefore each network entity is further specified using the category subdivision. Two basic meta-topological relationships are distinguished: the Boundary Interface and the Domain Interface. The topology level is a specification of the generalised topology level, in which the dimensional order is determined: Vertex (0), Edge(1), Face(2) and Shell(3).

4.17.1.78 AEC NODE

A **functional unit** is represented by an **aec node** in the network. **nodes** may have a limited number of **node ends**. Topological Nodes can be distinguished in Open Topological Nodes and Closed Topological Nodes. A Closed Topological Node refers directly to an **aec domain**, while an Open Topological Node refers indirectly via an External Domain End or even a Domain Interface.

```
*)
ENTITY aec_node;
  topological      : LOGICAL;
  closed           : OPTIONAL LOGICAL;
  has              : OPTIONAL LIST [1:?] OF node_end;
  working_area     : OPTIONAL LIST [1:1] OF aec_domain;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

topological:

closed:

has:

working_area:

```
*)
RULE aec_node_rule FOR (aec_node);
  LOCAL
    i : INTEGER;
  END_LOCAL;
  IF (NOT topological) THEN
    REPEAT i := 1 TO SIZEOF(has);
      IF (has[i].topological <> FALSE) THEN
        VIOLATION;
      END_IF;
    END_REPEAT;
```

```
    END_IF;
    REPEAT i := 1 TO SIZEOF(has);
      IF (has[i].topological AND
          has[i].adj_type = domain AND
          has[i].direction = ext) THEN
        IF (closed <> FALSE) OR (topological <> TRUE) THEN
          VIOLATION;
        END_IF;
      END_IF;
    END_REPEAT;
    IF (NOT closed) THEN
      IF (number_of_domain_ext(has) <> 1) THEN
        VIOLATION;
      END_IF;
    END_IF;
    IF (SIZEOF(working_area) = 1) THEN
      IF (topological <> TRUE OR closed <> TRUE) THEN
        VIOLATION;
      END_IF;
    END_IF;
  END_RULE;
  (*
```

PROPOSITIONS:

4.17.1.79 NODE END

Each node end can be connected to another node end by means of an interface. The constituting Ends of an Interface are considered as Mated Ends. The counterpart of the mated state is the (local) free state. In that case the End refers to a port (occurrence) of its aggregated technical solution occurrence. A Boundary End declares one side of a possible Boundary Interface between two Nodes. A further refinement defines the direction of the dimensional hop: an Internal Boundary End refers to the boundaries of the working area of the Node it belongs to, the dimensional order will decrease by one in this direction. an External Boundary End offers the working area of the Node it belongs to, to act as a boundary itself, the dimensional order will increase by one in this direction. A Domain End declares one side of a possible Domain Interface between two Nodes. A further refinement defines the direction of the dimensional hop: an Internal Domain End refers to the inner regions of the working area of the Node it belongs to, the dimensional order will stay the same or decrease in this direction. an External Domain End offers the working area of the Node it belongs to, to act as an inner region itself, the dimensional order will stay the same or increase in this direction.

```
  *)
  ENTITY node_end;
    topological       : LOGICAL;
    adj_type          : OPTIONAL adjacency_type;
    direction         : OPTIONAL dimensional_direction;
    is_connected_via  : connector;
    represented_by    : OPTIONAL side_or_region;
  DERIVE
    status            : connectivity_status :=
```

```
                    get_connectivity_status(is_connected_via);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

topological:

adj_type:

direction:

is_connected_via:

represented_by:

status:

```
*)
RULE node_end_constraints FOR (node_end);
  IF (status = mated) THEN
    IF (NOT topological) THEN
      IF (is_connected_via.topological <> FALSE) THEN
        VIOLATION;
      END_IF;
    ELSE
      BEGIN
        IF (is_connected_via.topological <> TRUE) THEN
          VIOLATION;
        END_IF;
        IF (adj_type = domain) THEN
          IF (is_connected_via <> domain) THEN
            VIOLATION;
          END_IF;
        ELSE
          IF (adj_type = boundary) THEN
            IF (is_connected_via <> boundary) THEN
              VIOLATION;
            END_IF;
          END_IF;
        END_IF;
      END;
    END_IF;
  ELSE
    IF (topological) THEN
      BEGIN
        IF (TYPEOF(represented_by) = aec_region) THEN
          IF (adj_type <> domain) OR
             (direction <> represented_by.direction) THEN
            VIOLATION;
          END_IF;
```

```
        ELSE
          IF (TYPEOF(represented_by) = side) THEN
            IF (adj_type <> boundary) OR
               (direction <> represented_by.direction) THEN
              VIOLATION;
            END_IF;
          END_IF;
        END_IF;
        IF (represented_by.connection <> functional) THEN
          VIOLATION;
        END_IF;
      END;
    END_IF;
  END_IF;
END_RULE;
(*
```

PROPOSITIONS:

4.17.1.80 INTERFACE

An interface establishes a relationship between two node ends of two different aec nodes within the same functional network. A Boundary Interface establishes a relationship between two Nodes, in which the working area of one Node acts as the boundary of the working area of the other Node. A Domain Interface establishes a relationship between two Nodes, in which the working area of one Node resides completely within the working area of the other Node.

```
*)
ENTITY interface;
  topological : LOGICAL;
  connects    : OPTIONAL LIST [2:2] OF side_or_region;
DERIVE
  adj_type : OPTIONAL adjacency_type := connects_type(connects[1]);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

topological:

connects:

adj_type:

```
*)
RULE interface_constraint FOR (interface);
  IF (topological) THEN
    BEGIN
      IF (TYPEOF(connects[1]) <> TYPEOF(connects[2])) THEN
        VIOLATION;
```

```
      END_IF;
      IF (connects[1].direction = connects[2].direction) THEN
        VIOLATION;
      END_IF;
      IF (TYPEOF(connects[1]) = side) THEN
        IF (ABS(connects[1].bounds.dimension -
                connects[2].bounds.dimension) <> 1) THEN
          VIOLATION;
        END_IF;
      END_IF;
    END;
  END_IF;
END_RULE;
(*

*)
RULE interface_and_two_node_ends FOR (interface, node_end);
  LOCAL
    conn_n   : INTEGER;
    dir      : ARRAY [1:2] OF dimensional_direction;
    adj_type : ARRAY [1:2] OF adjacency_type;
  END_LOCAL;
  REPEAT FOR EACH interface IN MODEL;
    conn_n := 0;
    REPEAT FOR EACH node_end IN MODEL;
      IF (node_end.is_connected_via = interface) THEN
        BEGIN
          conn_n := conn_n + 1;
          IF (node_end.topological) THEN
            BEGIN
              dir[conn_n] := node_end.direction;
              adj_type[conn_n] := node_end.adj_type;
            END;
          END_IF;
        END;
      END_IF;
    END_REPEAT;
    IF (conn_n <> 2) THEN
      VIOLATION;
    ELSE
      IF (topological AND
          (dir[1] = dir[2] OR
           adj_type[1] <> adj_type[2])) THEN
        VIOLATION;
      END_IF;
      IF (topological AND
          adj_type[1] <> interface.adj_type) THEN
        VIOLATION;
      END_IF;
    END_IF;
```

```
  END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

4.17.1.81 Topology

The generalised topology level implements a specific shape definition based on the general network level and meta-topology level. The generalised topology level consists of five entities: **aec domain**, **side**, **boundary**, **aec region** and **void**. They mirror on the generalised topology level the entities on the general network level and meta-topology level, as in Table 13.

Table 13: Relationship between AEC Network and Topology Levels.

network/meta-topology level	generalised topology level
AEC Node	AEC Domain
Boundary End	Side
Boundary Interface	Boundary
Domain End	AEC Region
Domain Interface	Void

To avoid redundancy the functional relationships rank above the topological relationships, i.e. if a relationship is both functional and topological it will be established as an Interface. Boundaries and Voids are reserved for pure topological relationships.

4.17.1.82 AEC DOMAIN

A **domain** defines an area of a certain **dimensional order**. To complete the definition it may refer, indirectly, to Domains of lesser or equal dimensional order. E.g. a second order Domain (Face) refers indirectly (by means of **sides** and **boundaries** or Boundary Interfaces) to first order Domains (Edges) which represent its outer boundaries. To specify inner areas a Domain may refer indirectly (by means of **aec regions** and **voids** or Domain Interfaces) to Domains of lower or equal order. E.g. a second order Domain (Face) may specify internal areas of again 2nd order (inner loop), 1st order (Edge shaped) and 0th order (Vertex shaped).

```
*)
ENTITY aec_domain;
  dimension : dimensional_order;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

dimension: The

4.17.1.83 SIDE

A **side** declares one side of a functional or topological relationship of type "Domain A is bounded by/bounds Domain B". Functional relationships are handled by Boundary Ends, which may be free

or mated by means of a Boundary Interface. Pure topological relationships are established by a **boundary**.

```
*)
ENTITY side;
  direction  : dimensional_direction;
  connection : connection_enum;
  bounds     : aec_domain;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

direction:

connection:

bounds:

4.17.1.84 BOUNDARY

A **boundary** establishes a topological relationship between two **aec domains**, in which one Domain is bounded by/bounds another Domain. The Domains in question should differ one in dimensional order.

```
*)
ENTITY boundary;
  int_side : side;
  ext_side : side;
WHERE
  (int_side.connection = ext_side.connection = topological);
  int_side.direction = int;
  ext_side.direction = ext;
  int_side.bounds.dimension = (ext_side.bounds.dimension + 1);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

int_side:

ext_side:

PROPOSITIONS:

4.17.1.85 AEC REGION

An **aec region** declares one side of a functional or topological relationship of type "Domain A encloses/is enclosed by Domain B". Functional relationships are handled by Domain Ends, which may be free, or mated by means of a Domain Interface. Pure topological relationships are established by a **void**.

```
*)
ENTITY aec_region;
  direction  : dimensional_direction;
  connection : connection_enum;
  encloses   : aec_domain;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

direction:

connection:

encloses:

```
*)
RULE one_ext_region_per_domain FOR (aec_region, aec_domain);
  LOCAL
     ext_n : INTEGER;
 END_LOCAL;
 REPEAT FOR EACH aec_domain IN MODEL;
   ext_n := 0;
   REPEAT FOR EACH aec_region IN MODEL;
     IF (aec_region.direction = ext AND
         aec_region.encloses = aec_domain) THEN
       ext_n := ext_n + 1;
     END_IF;
   END_REPEAT;
   IF (ext_n > 1) THEN
     VIOLATION;
   END_IF;
 END_REPEAT;
END_RULE;
(*
```

PROPOSITIONS:

4.17.1.86 VOID

A void establishes a topological relationship between two aec domains, in which one Domain resides completely within another Domain. The dimensional order of the enclosed Domain should be less or equal to the dimensional order of the enclosing Domain.

```
*)
ENTITY void;
  int_region : aec_region;
  ext_region : aec_region;
WHERE
  {int_region.connection = ext_region.connection = topological};
```

```
  int_region.direction = int;
  ext_region.direction = ext;
  int_region.encloses.dimension >= ext_region.encloses.dimension;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

int_region:

ext_region:

PROPOSITIONS:

4.17.1.87 PLACE

```
*)
ENTITY place;
  loc      : point;
  s_vector : vector;
  p_vector : vector;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

loc:

s_vector:

p_vector:

4.17.1.88 AEC Core Schema Classification Structure

The following indented listing provides the classification structure for the entities within the AEC Core Model.

```
ACTUAL PARAMETER
  INT VAL
  LOGIC VAL
  REAL VAL
  STRING VAL
AEC APPROVAL
AEC DOMAIN
AEC NODE
AEC REGION
ALLOWED PARAMETER DOMAIN
  INT FORMAL
  LOGIC FORMAL
  REAL FORMAL
  STRING FORMAL
```

ANALYSIS
ASPECT
  ISO6241
    ACOUSTICAL 6242
    AIR PURITY 6242
    ANTHROPODYNAMIC 6241
    DURALIBILTY 6241
    ECONOMIC 6241
      CAPITAL COST
      MAINTENANCE COST
      RUNNING COST
    ENERGY 6242
      ENERGY USE 6242
      HEAT TRANSFER 6242
    FIRE SAFETY 6241
    HYGIENE 6241
    HYGRO THERMAL 6242
    SAFETY 6241
    SPECIFIC USE 6241
    STABILITY 6241
    TACTILE 6241
    TIGHTNESS 6241
    VISUAL 6242
BOUNDARY
CHARACTERISTIC
  EXPECTED CHARACTERISTIC
  REQUIRED CHARACTERISTIC
ENVIRONMENTAL AGENT
  AGENT 6241
    BIOLOGICAL AGENT
    CHEMICAL AGENT
    ELECTRONICAL AGENT
    MECHANICAL AGENT
    THERMAL AGENT
EXPERT
EXPLICIT TECHNICAL SOLUTION DEFINITION
FUNCTIONAL UNIT CLASSIFICATION
  SFB1
GENERIC FUNCTIONAL UNIT STRUCTURE
GENERIC PORT
GENERIC TECHNICAL SOLUTION
GENERIC TECHNICAL SOLUTION STRUCTURE
INT DOMAIN
INTERFACE
NODE END
PLACE
PORT OCCURRENCE
PROCEDURAL DESCR
PRODUCT DEFINITION UNIT

```
  FUNCTIONAL UNIT
    FUNCTIONAL UNIT OCCURRENCE
      NON TOPOLOGICAL FU OCCURRENCE
      TOPOLOGICAL FU OCCURRENCE
    GENERIC FUNCTIONAL UNIT
    SPECIFIC FUNCTIONAL UNIT
  TECHNICAL SOLUTION
PRODUCT MODEL
REAL DOMAIN
SFB CODE
SIDE
SPECIFIC PORT
SPECIFIC TECHNICAL SOLUTION
TECHNICAL SOLUTION ARGUMENT
TECHNICAL SOLUTION DECISION
TECHNICAL SOLUTION OCCURRENCE
VOID

*)
END_SCHEMA;  --- end IPIM AEC Core Schema
(*
```

4.18 Ship Models

4.18.1 Ships Structural Model

4.18.1.1 Introduction

The purpose of this document is to provide, in an information model, an adequate framework for the formulation of a data exchange standard for ship's structure. In particular, the goal has been to identify sufficient information, such that data which represents the majority of a ship's structural system can be communicated digitally via this Standard between two CAD/CAM modeling systems without manual intervention or interpretation. This structural information model is but one of many such models which are required for definition and integration prior to the formulation of this exchange standard.

While the information contained in this document is not complete, it received sufficient review by both the Navy/Industry Digital Data Exchange Standards Committee (NIDDESC) and the Architecture, Engineering, and Construction (AEC) Committee to warrant submission to the Product Data Exchange Specification (PDES) organization at this time. It is expected that subsequent revisions to this document will be made to incorporate refinements, and information not now included. For these reasons, it is not the intent that the information contained herein, in its present form, be used as a shipbuilding industry standard. As with other information models, it should be noted that the information contained herein is only one of many possible ways of representing a ship's structural system.

```
*)
SCHEMA ipim_ship_structure_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_resources_schema,
         ipim_geometry_schema,
         ipim_material_schema);

(*
```

In the context of this purview, the product of an AEC industry activity is sites, factories, plants, ships (floating plants), and/or buildings. These are in turn composed of various systems.

Engineering systems contain equipment interconnected by distribution systems. The reference model for distribution systems is described in ISO TC184/SC4/WG1 document number 3.2.2.2.

A structural system provides supporting structure, integrity, shelter, and habitability. The scope of this document is a steel structural system, and in particular a ship's structural system. It should be noted that although this model was developed specifically for a ship's structural system, many of the concepts and entities presented in this model can readily be applied to other types of steel structures.

4.18.1.2 Model Integration

This model is an application model. As such it incorporates concepts and in some cases specific entities from other models in the development organization. These models include Geometry, Topology, Form Features, Solids, Materials, and Product Structure Configuration Management (PSCM). Although much work remains to be done, integration of this model with the above mentioned models has begun within the Integration Sub-committee.

Integration points have been identified in this model. These points are identified in the entity descriptions. Subsequent revisions to the model will carry the integration effort further.

4.18.1.3 Scope

The scope of this document includes the level of definition of a structural product model resulting from the completion of detailed design and lofting. Specifically included are all items listed below. Nesting data for plates and shapes is excluded because typically it is developed in an organizationally specific manner or format. The intent is to include product definition data (e.g. the product model) since this data is logically transferable between different organizations responsible for building and/or maintaining the ship.
This document includes a definition of geometry, topology and property data for the following items:

- Lines
- Stiffened surfaces (shell, bulkheads, decks, web frames, etc.)
- Cutouts, lightening holes and penetrations
- Weld data and bevels
- Stiffener data (scantlings/traces/orientation/end cuts)
- Material definition (thickness, type, material, etc.)
- Brackets, collar plates
- Stanchions
- Units/Assemblies
- Foundations
- Rudder

This version does not define the following:

- Hangers for distribution systems
- Non-structural tanks
- Struts and bossings
- Castings and forgings
- False decks and gratings

These five items will be reviewed for inclusion in subsequent versions of this document. Of the above mentioned items, castings and forgings stand out as an integration area with the Solids Modeling group.
It is not the intent of this document to cover configuration management of structural items in any great detail. While some attempt has been made with the inclusion of a Date/Time concept, this is intended to be a place holder for future work. This has been identified as an area of integration with the Product Structure Configuration Management (PSCM) model.

4.18.1.4 Hull Unit Assembly TYPE Definitions

4.18.1.4.1 HULL NAME

A **hull name** is the character identification of a **hull**, e.g. USS Thomas S. Gates.

```
*)
TYPE hull_name = STRING;
END_TYPE;
(*
```

4.18.1.4.2 HULL NUMBER

A **hull number** is the numeric identifier of a **hull**. The **hull number** typically takes the form of a Navy hull number or a manufacturers hull number, e.g. H420.

```
*)
TYPE hull_number = STRING;
END_TYPE;
(*
```

4.18.1.4.3 PROJECT PHASE

The **project phase** is a design stage, ie. functional design, detail design, etc.

```
*)
TYPE project_phase = STRING;
END_TYPE;
(*
```

4.18.1.4.4 ASSEMBLY ID NAME

An **assembly id name** is used to identify a type of unit assembly. Examples include Unit, Sub-assembly, and Section.

```
*)
TYPE assembly_id_name = STRING;
END_TYPE;
(*
```

4.18.1.4.5 ASSEMBLY ID NUMBER

An **assembly id number** is the numeric identifier of a unit assembly, e.g. 3200-0045.

```
*)
TYPE assembly_id_number = STRING;
END_TYPE;
(*
```

4.18.1.4.6 PART ID

A **part id** is the unique identifier of a part. It may take the form of a manufacturers stock number.
Integration point: PSCM Model, Product Item ID entity.

```
*)
TYPE part_id = STRING;
END_TYPE;
(*
```

4.18.1.4.7 SYSTEM ID

A **system id** is the unique identifier of a system.

```
*)
TYPE system_id = STRING;
END_TYPE;
(*
```

4.18.1.5 HULL

A **hull** is a collection of **systems** which comprise a ship (product model).
Integration point: Product Structure Model; Product Model entity.

```
*)
ENTITY hull;
  identified_by_hull_name    : SET [1:?] OF hull_name;
  identified_by_hull_number : hull_number;
  made_up_of_system          : SET [1:?] OF system;
UNIQUE
  identified_by_hull_number;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_hull_name: Character identification of a hull, e.g., USS Thomas S. Gates.

identified_by_hull_number: Numeric identifier of a hull, e.g., 4420

made_up_of_system:

PROPOSITIONS:

1. **Identified by hull number** must be UNIQUE.

4.18.1.6 SYSTEM

A **system** is a functionally related group of elements (potentially recursive). Some examples of functional groupings are structure, HVAC or electrical systems.

```
*)
ENTITY system
  SUPERTYPE OF (structural_system);
  with_system_id : system_id;
UNIQUE
  with_system_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_system_id: Unique identifier of a system.

PROPOSITIONS:

1. With system id must be UNIQUE.

4.18.1.7 STRUCTURAL SYSTEM

A structural system is a collection of structural parts used, in general, to compartment and support all other systems.

```
*)
ENTITY structural_system
  SUBTYPE OF (system);
  made_up_of_unit_assembly : SET [1:?] OF unit_assembly;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

made_up_of_unit_assembly: Component unit assemblies.

4.18.1.8 UNIT ASSEMBLY

A unit assembly gathers together parts and/or sub-assemblies. The unit assembly can represent a logical grouping (HFO tank, Space 42, etc) or a physical grouping associated with actual construction phases of the hull.

```
*)
ENTITY unit_assembly
  SUPERTYPE OF (sub_assembly);
  with_project_phase         : project_phase;
  identified_by_assembly_id : assembly_id;
UNIQUE
  identified_by_assembly_id;
  with_project_phase, identified_by_assembly_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_project_phase:-

identified_by_assembly_id:

PROPOSITIONS:

1. Identified by assembly id must be UNIQUE.
2. The combination of with project phase and identified by assembly id must be UNIQUE.

4.18.1.9 ASSEMBLY ID

An assembly id is the unique identifier of a unit assembly. The assembly id consists of an assembly ID name and an assembly ID number.

```
*)
ENTITY assembly_id;
  having_assembly_id_number : assembly_id_number;
  having_assembly_id_name   : assembly_id_name;
UNIQUE
  having_assembly_id_name, having_assembly_id_number;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_assembly_id_number:

having_assembly_id_name:

PROPOSITIONS:

1. The combination of having assembly id name and having assembly id number must be UNIQUE.

4.18.1.10 SUB-ASSEMBLY

A sub-assembly is a collection of parts and/or other sub-assemblies. Sub-assemblies can gather parts and/or sub-assemblies into logical groupings (decks, bulkheads, etc) or into physical groupings representing actual construction phases of the hull.

```
*)
ENTITY sub_assembly
  SUBTYPE OF (unit_assembly);
  made_of_sub_assembly : OPTIONAL SET [1:#] OF sub_assembly;
  part_of_sub_assembly : OPTIONAL sub_assembly;
  made_of_part         : OPTIONAL SET [1:#] OF part;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

made_of_sub_assembly:

part_of_sub_assembly:

made_of_part:

4.18.1.11 PART

A part is a unique structural element or component consumed during the production process.
Integration point: Product Structure Model, Product Item entity.

```
*)
ENTITY part
  SUPERTYPE OF (library_part XOR
                plate_part XOR
                shape_part);
  identified_by_part_id       : part_id;
  last_modified_on_date_time  : ship_date_time;
  created_on_date_time        : ship_date_time;
  made_of_material            : ship_material;
UNIQUE
  identified_by_part_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_part_id:

last_modified_on_date_time:

created_on_date_time:

made_of_material:

PROPOSITIONS:

1. **Identified by part id** must be UNIQUE.

4.18.1.12 SHIP DATE TIME

A **ship date time** is expressed in the form *yymmdd.hhmmss.*
Integration point: Miscellaneous Resources Model: Unit, Time-Unit entities.

```
*)
ENTITY ship_date_time;
  with_date_time_value            : date_time;
UNIQUE
  with_date_time_value;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_date_time_value: -

PROPOSITIONS:

1. **With date time value** must be UNIQUE.

4.18.1.13 SHIP MATERIAL

A **ship material** is the substance making up a **part**. This entity includes a description of the material and its properties.

Integration point: Materials Model: Material entity.

```
*)
ENTITY ship_material;
  with_material_specification : material_property;
UNIQUE
  with_material_specification;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_material_specification: -

PROPOSITIONS:

1. **With material specification** must be UNIQUE.

4.18.1.14 Ship Geometry

The product model for a ship's structural system must contain both the topological relationships, the geometry, material properties and joining information (welds and bevels). Topology is expressed in terms of surfaces which bound other surfaces or shapes, surfaces which are bounded by other surfaces, and shapes which lie on surfaces and are bounded by surfaces and/or other shapes.

4.18.1.15 Ship Geometry TYPE Definitions

4.18.1.15.1 COMPARTMENT ID

A **compartment id** is a unique identifier for a **compartment**. For example, '1-346-0-L'.

```
*)
TYPE compartment_id = STRING;
END_TYPE;
(*
```

4.18.1.15.2 COMPARTMENT NAME

A **compartment name** is an identifier for a **compartment**, for example 'CREW LIVING SPACE NO. 4'.

```
*)
TYPE compartment_name = STRING;
END_TYPE;
(*
```

4.18.1.15.3 SURFACE ID NAME

The **surface id name** is a part of the **surface id** and may be non-unique. Deck and Bulkhead are both examples of a **surface id name**.

```
*)
TYPE surface_id_name = STRING;
END_TYPE;
(*
```

4.18.1.15.4 SURFACE ID NUMBER

A **surface id number** is the unique part of the **surface id** which defines the surface, e.g. 320-0045.

```
*)
TYPE surface_id_number = STRING;
END_TYPE;
(*
```

4.18.1.15.5 CURVE ID

A **curve id** is the unique identifier for a **molded curve**.

```
*)
TYPE curve_id = STRING;
END_TYPE;
(*
```

4.18.1.16 SHIP BOUNDED SURFACE

A **ship bounded surface** is a parameterized space, representing an orientable locus of points, bounded by a set of **molded curves** which forms a closed contour. A **ship bounded surface** may be planar (deck) or sculptured (shell).

Integration point: Geometry Model: Bounded Surface entity.

```
*)
ENTITY ship_bounded_surface
  SUPERTYPE OF (ship_curved_surface XOR
                ship_elementary_surface);
  defining_node                : OPTIONAL SET [1:#] OF ship_node;
  defining_molded_curve        : OPTIONAL SET [1:#] OF molded_curve;
  bounded_by_surface_edge      : SET [1:#] OF surface_edge;
  defining_structural_opening : OPTIONAL SET [1:#] OF
                                         structural_opening;
  identified_by_surface_id     : surface_id;
UNIQUE
  identified_by_surface_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

defining_node:

defining_molded_curve:

bounded_by_surface_edge:

defining_structural_opening:

identified_by_surface_id:

PROPOSITIONS:

1. Identified by surface id must be UNIQUE.

4.18.1.17 SURFACE ID

A surface id is a unique identifier for a surface. It consists of a Surface ID Name and a Surface ID Number, the combination of which uniquely identify the surface (e.g DECK-320-45).

```
*)
ENTITY surface_id;
  having_surface_id_number       : surface_id_number;
  having_surface_id_name         : surface_id_name;
UNIQUE
  having_surface_id_number;
  having_surface_id_name, having_surface_id_number;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_surface_id_number:

having_surface_id_name:

PROPOSITIONS:

1. Having surface id number must be UNIQUE.
2. The combination of having surface id name and having surface id number must be UNIQUE.

4.18.1.18 SHIP CURVED SURFACE

A ship curved surface is a non-planar surface representing such areas the shell of a ship.

```
*)
ENTITY ship_curved_surface
  SUPERTYPE OF (ship_bspline_surface XOR
                ship_bezier_surface)
  SUBTYPE OF (ship_bounded_surface);
END_ENTITY;
(*
```

4.18.1.19 SHIP ELEMENTARY SURFACE

A **ship elementary surface** is a simple surface such as a planar, conical or cylindrical surface.
Integration point: Geometry model, Elementary Surface entity.

```
*)
ENTITY ship_elementary_surface
  SUPERTYPE OF (ship_planar_surface)
  SUBTYPE OF (ship_bounded_surface);
END_ENTITY;
(*
```

4.18.1.20 SHIP B-SPLINE SURFACE

A **ship bspline surface** identifies a particular mathematical representation for a curved surface.
Integration point: Geometry model, Bspline surface entity.

```
*)
ENTITY ship_bspline_surface
  SUBTYPE OF (ship_curved_surface);
  geometry_definition : bspline_surface;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

geometry_definition: The geometric definition of the surface.

4.18.1.21 SHIP BEZIER SURFACE

A **ship bezier surface** identifies a particular mathematical representation for a curved surface.
Integration point: Geometry model, Bezier surface entity.

```
*)
ENTITY ship_bezier_surface
  SUBTYPE OF (ship_curved_surface);
  geometry_definition : bezier_surface;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

geometry_definition: The geometric definition of the surface.

4.18.1.22 SHIP PLANAR SURFACE

A **ship planar surface** is a surface with no curvature anywhere within its boundaries.
Integration point: Geometry model, Plane entity.

```
*)
ENTITY ship_planar_surface
  SUBTYPE OF (ship_elementary_surface);
  oriented_by_unit_vector    : ship_unit_vector;
  located_by_position_point : ship_position_point;
UNIQUE
  located_by_position_point, oriented_by_unit_vector;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

oriented_by_unit_vector:

located_by_position_point:

PROPOSITIONS:

1. The combination of **located by position point** and **oriented by unit vector** must be UNIQUE.

4.18.1.23 SURFACE EDGE

A **surface edge** is a sequence of **molded curves** bounding a **ship surface.**

```
*)
ENTITY surface_edge;
  identified_by_edge_sequence_number : INTEGER;
  defined_by_molded_curve            : molded_curve;
  bounding_bounded_surface           : ship_bounded_surface;
UNIQUE
  identified_by_edge_sequence_number, defined_by_molded_curve,
   bounding_bounded_surface;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_edge_sequence_number: Provides an ordered sequence.

defined_by_molded_curve:

bounding_bounded_surface:

PROPOSITIONS:

1. **Identified by edge sequence number** must be UNIQUE.

4.18.1.24 MOLDED CURVE

A **molded curve** is a curve on a surface, a curve bounding a surface or a curve formed by the intersection of two surfaces.

Integration point: Geometry Model: Curve, Curve-On Surface, Intersection-Curve entities.

```
*)
ENTITY moldad_curve;
  identified_by_curve_id      : curve_id;
  defined_by_curve_geometry   : ship_curve_geometry;
  locating_node               : OPTIONAL SET [1:#] OF ship_node;
UNIQUE
  identified_by_curve_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_curve_id:

defined_by_curve_geometry:

locating_node:

PROPOSITIONS:

1. Identified by curve id must be UNIQUE.

4.18.1.25 SHIP POSITION POINT

A ship position point is a point defined in three dimensions. It provides the x, y and z coordinates positioning a planar surface.

Integration point: Geometry model, point and cartesian three coordinate entities.

```
*)
ENTITY ship_position_point;
  having_x_position        : REAL;
  having_y_position        : REAL;
  having_z_position        : REAL;
UNIQUE
  having_x_position, having_y_position, having_z_position;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_x_position: The X coordinate of the point location.

having_y_position: The Y coordinate of the point location.

having_z_position: The Z coordinate of the point location.

PROPOSITIONS:

1. The combination of having x position, having y position and having z position must be UNIQUE.

4.18.1.26 SHIP UNIT VECTOR

A **ship unit vector** is a vector with a magnitude of one, indicating direction in space.
Integration point: Geometry model, vector and direction entities.

```
*)
ENTITY ship_unit_vector;
  having_x_value                : REAL;
  having_y_value                : REAL;
  having_z_value                : REAL;
UNIQUE
  having_x_value, having_y_value, having_z_value;
WHERE
  (having_x_value**2 +
   having_y_value**2 +
   having_z_value**2) = 1.0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_x_value: X value of unit vector.

having_y_value: Y value of unit vector.

having_z_value: Z value of unit vector.

PROPOSITIONS:

1. The combination of **having x value**, **having y value** and **having z value** must be UNIQUE.
2. The direction is a unit vector.

4.18.1.27 SHIP TRANSFORMATION MATRIX

A **ship transformation matrix** is a 4 x 4 matrix specifying the translation and orientation (rotation) of a library part.
Integration point: Geometry Model: Transformation entity.

```
*)
ENTITY ship_transformation_matrix;
  defined_by_transformation_matrix_value : ARRAY [1:4] OF
                                           ARRAY [1:4] OF
                                           REAL;
UNIQUE
  defined_by_transformation_matrix_value;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

defined_by_transformation_matrix_value: -

PROPOSITIONS:

1. **Defined by transformation matrix value** must be UNIQUE.

4.18.1.28 SHIP CURVE GEOMETRY

The **ship curve geometry** provides the mathematical representation for a curve in space.
Integration point: Geometry model, curve entity.

```
*)
ENTITY ship_curve_geometry;
  geometry_definition                    : curve;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

geometry_definition: The definition of the curve geometry.

4.18.1.29 COMPARTMENT

A **compartment** is an enclosed space within a ship. An example of a compartment would be "Auxiliary Machinery Room 2".

```
*)
ENTITY compartment;
  identified_by_compartment_id    : compartment_id;
  identified_by_compartment_name  : OPTIONAL compartment_name;
  bounded_by_bounded_surface      : SET [1:?] OF
                                        ship_bounded_surface;
UNIQUE
  compartment_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_compartment_id:

identified_by_compartment_name:

bounded_by_bounded_surface:

PROPOSITIONS:

1. **Identified by compartment id** must be UNIQUE.

4.18.1.30 Plate Parts

A plate part is a part cut from flat material stock. The part may be used as is or subsequently bent to form a flange (flange part) or rolled. Plate parts are defined on surfaces and may may be offset from that surface some distance. Plate parts can be joined with other plate/shape parts at plate part edges or along traces on the plate part face.

4.18.1.31 Plate Part TYPE Definitions

4.18.1.31.1 NODE ID

A **node id** is a unique identifier of a **ship node**.

```
*)
TYPE node_id = STRING;
END_TYPE;
(*
```

4.18.1.31.2 PATH SEGMENT ID

A **path segment id** is the unique identifier of a **path segment**.

```
*)
TYPE path_segment_id = STRING;
END_TYPE;
(*
```

4.18.1.31.3 EDGE PREPARATION DESCRIPTION

Edge preparation description provides a description of the physical geometry or condition at a plate or shape edge.

```
*)
TYPE edge_preparation_description = STRING;
END_TYPE;
(*
```

4.18.1.32 PLATE PART

A **plate part** is a part cut from flat material stock. The part may be used as is or subsequently bent to form a flange (flange part) or rolled.

Integration point: Product Structure Model: Product-Item entity.

```
*)
ENTITY plate_part
  SUBTYPE OF (part);
  with_plate_surface_offset : REAL;
  with_plate_part_thickness : REAL;
  marked_with_nc_mark       : OPTIONAL SET [1:#] OF nc_mark;
  having_plate_part_edge    : SET [1:#] OF plate_part_edge;
  having_part_flange        : OPTIONAL SET [1:#] OF part_flange;
  cut_by_structural_opening : OPTIONAL SET [1:#] OF
                                  structural_opening;
  defined_by_path_segment   : OPTIONAL SET [1:#] OF path_segment;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_plate_surface_offset: Offset value of **plate part** from bounded surface.

with_plate_part_thickness: Thickness of plate material.

marked_with_nc_mark:

having_plate_part_edge:

having_part_flange:

cut_by_structural_opening:

defined_by_path_segment:

4.18.1.33 PLATE PART EDGE

```
*)
ENTITY plate_part_edge;
  identified_by_edge_sequence_number : INTEGER;
  defined_by_path_segment            : path_segment;
  having_edge_preparation            : OPTIONAL edge_preparation;
  for_plate_part                     : plate_part;
UNIQUE
  for_plate_part, identified_by_edge_sequence_number,
  defined_by_path_segment;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_edge_sequence_number: Provides an ordered sequence.

defined_by_path_segment:

having_edge_preparation:

for_plate_part:

PROPOSITIONS:

1. The combination of **for plate part**, **identified by edge sequence number** and **defined by path segment** must be UNIQUE.

4.18.1.34 SHIP NODE

A **ship node** is the logical equivalent of a geometric point. It is a unique (topological) point R3 with dimensionality 0 and extent 0.

Integration point: Topology Model; Vertex entity, and Geometry Model; Point entity.

```
*)
ENTITY ship_node;
  identified_by_node_id        : node_id;
UNIQUE
  identified_by_node_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_node_id: -

PROPOSITIONS:

1. **Identified by node id** must be UNIQUE.

4.18.1.35 PATH SEGMENT

A **path segment** is a bounded portion of a **molded curve** defined by two **ship nodes** with the positive direction of the path from the first node to the second node.
Integration point: Geometry Model: Curve, Bounded-Curve, Trimmed-Curve entities.

```
*)
ENTITY path_segment;
  identified_by_path_segment_id : path_segment_id;
  ending_on_node                : ship_node;
  starting_on_node              : ship_node;
  defined_on_molded_curve       : molded_curve;
UNIQUE
  identified_by_path_segment_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_path_segment_id: Unique identifier.

ending_on_node: Ending point of **path segment.**

starting_on_node: Starting point of **path segment.**

defined_on_molded_curve: Reference curve.

PROPOSITIONS:

1. **Identified by path segment id** must be UNIQUE.

4.18.1.36 EDGE PREPARATION

Edge preparation is the physical description of a plate part edge or shape part edge. It is typically a bevel or chamfer as required by the welding process.
Integration point: Form Feature Model.

```
*)
ENTITY edge_preparation;
  described_by_edge_preparation_description :
                                  edge_preparation_description;
UNIQUE
  described_by_edge_preparation_description;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

described_by_edge_preparation_description: -

PROPOSITIONS:

1. Described by edge preparation description must be UNIQUE.

4.18.1.37 PART FLANGE

A part flange is a plate part that has a roll or knuckle along a path segment to form a flange.

```
*)
ENTITY part_flange;
  with_flange_angle         : REAL;
  with_flange_radius        : REAL;
  defined_on_molded_curve   : molded_curve;
  with_endcut               : OPTIONAL endcut;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_flange_angle: Angle of the bend of a flange.

with_flange_radius: Bend radius.

defined_on_molded_curve:

with_endcut:

4.18.1.38 Ship Shape Parts

Shape parts are rolled, extruded, or built up structural shapes. Shape parts may have standard or non-standard cross sections, they may be twisted, they may follow a curved or straight path and they may be joined along their length and/or ends to other shapes and/or plates. In addition, shape parts have endcuts which define the geometry at the ends of the shapes or which may create a transition piece between two different size shape parts.

4.18.1.39 Ship Shape Part TYPE Definitions

4.18.1.39.1 SHAPE REFERENCE POINT

A **shape reference point** identifies the position of a **shape part** relative to a **path segment**; that is, to the left, right or centered on the **path segment**.

```
*)
TYPE shape_reference_point = ENUMERATION OF
      (left,
       center,
       right);
END_TYPE;
(*
```

4.18.1.39.2 SHAPE PART TYPE

A **shape part type** identifies the type of a **shape part**. The types of **shape parts** used in shipbuilding are I-beam, T-bar, Angle bar, Flat bar and Channel.

```
*)
TYPE shape_part_type = ENUMERATION OF
      (I_beam,
       T_bar,
       angle_bar,
       flat_bar,
       channel);
END_TYPE;
(*
```

4.18.1.39.3 STANDARD SHAPE ID

A **standard shape id** is the unique identifier of a **shape part** with standard cross section. The identifier uses the unified numbering system in accordance with ASTM and SAE.

```
*)
TYPE standard_shape_id = STRING;
END_TYPE;
(*
```

4.18.1.39.4 CROSS SECTION CODE

The **cross section code** identifies the variables defining the cross section of a shape part. For example: tf = average flange thickness, r = fillet radius, t = clear web height between fillets, etc.

```
*)
TYPE cross_section_code = STRING;
END_TYPE;
(*
```

4.18.1.39.5 NC MARK ID

An nc mark id is the unique identifier of an nc mark.

```
*)
TYPE nc_mark_id = STRING;
END_TYPE;
(*
```

4.18.1.39.6 NC TEXT PARAMETER CODE

The nc text parameter code provides the variables associated with nc text, ie. c = character height.

```
*)
TYPE nc_text_parameter_code = STRING;
END_TYPE;
(*
```

4.18.1.39.7 NC TEXT STRING

The nc text string provides the actual character data to be marked by the N/C marking machine on the part.

```
*)
TYPE nc_text_string = STRING;
END_TYPE;
(*
```

4.18.1.39.8 ENDCUT PARAMETER CODE

An endcut parameter code identifies a variable which defines an attribute of an endcut, ie. p1 = web snipe length.

```
*)
TYPE endcut_parameter_code = STRING;
END_TYPE;
(*
```

4.18.1.39.9 PARAMETRIC ENDCUT ID

A parametric endcut id is the unique identifier of a parametric endcut. It could be a reference to a standard endcut in a library or a standard macro, e.g 'T86S'.

```
*)
TYPE parametric_endcut_id = STRING;
END_TYPE;
(*
```

4.18.1.40 SHAPE PART

A shape part is a rolled, extruded, or built up structural shape.
Integration point: Product Structure Model; Product-Item entity.

```
*)
ENTITY shape_part
  SUBTYPE OF (part);
  located_by_shape_reference_point : shape_reference_point;
  offset_by_shape_surface_offset   : REAL;
  identified_with_shape_part_type  : shape_part_type;
  having_shape_part_edge           : SET [1:#] OF shape_part_edge;
  ending_with_endcut               : endcut;
  starting_with_endcut             : endcut;
  oriented_by_shape_orientation    : SET [1:#] OF shape_orientation;
  ending_with_shape_clearance      : shape_clearance;
  starting_with_shape_clearance    : shape_clearance;
  identified_with_cross_section    : cross_section;
  cut_by_structural_opening        : OPTIONAL SET [1:#] OF
                                             structural_opening;
  penetrating_cutout_hole          : OPTIONAL SET [1:#] OF
                                             cutout_hole;
  marked_by_nc_mark                : OPTIONAL SET [1:#] OF nc_mark;
  defined_on_path_segment          : SET [1:#] OF path_segment;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

located_by_shape_reference_point:

offset_by_shape_surface_offset:

identified_with_shape_part_type:

having_shape_part_edge:

ending_with_endcut:

starting_with_endcut:

oriented_by_shape_orientation:

ending_with_shape_clearance:

starting_with_shape_clearance:

identified_with_cross_section:

cut_by_structural_opening:

penetrating_cutout_hole:

marked_by_nc_mark:

defined_on_path_segment:

PROPOSITIONS:

4.18.1.41 SHAPE CLEARANCE

A **shape clearance** is the distance between the end nodes defining a **path segment**, and the actual extreme starting and ending points of the **shape part** defined on that **path segment**. In effect, the actual length of a **shape part** may be <= the length of the **path segment** it is defined on.

```
*)
ENTITY shape_clearance;
  with_clearance_length   : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_clearance_length: Numeric value of clearance

4.18.1.42 SHAPE PART EDGE

A **shape part edge** is one of an ordered set of **path segments** defining the edge of a **shape part**.

```
*)
ENTITY shape_part_edge;
  identified_by_edge_sequence_number : INTEGER;
  for_shape_part                     : shape_part;
  having_edge_preparation            : OPTIONAL edge_preparation;
  defined_by_path_segment            : path_segment;
UNIQUE
  identified_by_shape_part;
  for_shape_part, identified_by_edge_sequence_number,
                  defined_by_path_segment;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_edge_sequence_number: Provides an ordered sequence.

for_shape_part:

having_edge_preparation:

defined_by_path_segment:

PROPOSITIONS:

1. For **shape part** must be UNIQUE.
2. The combination of **for shape part**, **identified by edge sequence number** and **defined by path segment** must be UNIQUE.

4.18.1.43 SHAPE ORIENTATION

A **shape orientation** defines the orientation of a **shape part** at a **ship node**. Multiple **shape orientations** can be used to represent a twisted shape part.

```
*)
ENTITY shape_orientation;
  defined_by_unit_vector : ship_unit_vector;
  defined_at_node        : ship_node;
UNIQUE
  defined_by_unit_vector, defined_at_node;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

defined_by_unit_vector: Indicates orientation.

defined_at_node: Location of unit vector.

PROPOSITIONS:

1. The combination of **defined by unit vector** and **defined at node** must be UNIQUE.

4.18.1.44 CROSS SECTION

A shape part **cross section** provides a description of the cross section geometry (a cut perpendicular to the linear axis) of the **shape part**. **Cross section** descriptions may be parametric or non-parametric.

```
*)
ENTITY cross_section
  SUPERTYPE OF (non_standard_cross_section XOR
                standard_cross_section);
END_ENTITY;
(*
```

4.18.1.45 STANDARD CROSS SECTION

A **standard cross section** is a parametric description whose parameters are provided through the unified numbering system established in accordance with ASTM and SAE.

```
*)
ENTITY standard_cross_section
  SUBTYPE OF (cross_section);
  with_standard_shape_id                : standard_shape_id;
  described_by_cross_section_parameter : SET [1:#] OF
                                         cross_section_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_standard_shape_id:

described_by_cross_section_parameter:

4.18.1.46 NON-STANDARD CROSS SECTION

A **non-standard cross section** is a non-parametric description. Specific parameters associated with web and flanges, for a specific shape part type, are provided by this description.

1. Only I-beams and channels have lower flanges.
2. A flat bar is made up only of a web, it has no flanges.

```
*)
ENTITY non_standard_cross_section
  SUBTYPE OF (cross_section);
  with_lower_flange_thickness : OPTIONAL REAL;
  with_lower_flange_width     : OPTIONAL REAL;
  with_upper_flange_thickness : OPTIONAL REAL;
  with_upper_flange_width     : OPTIONAL REAL;
  with_web_thickness          : REAL;
  with_web_height             : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_lower_flange_thickness: The thickness of the lower flange (I-beam or channel).

with_lower_flange_width: The width of the lower flange (I-beam or channel).

with_upper_flange_thickness: The thickness of the upper flange (I-beam or channel).

with_upper_flange_width: The width of the upper flange (I-beam or channel).

with_web_thickness: The thickness of the web.

with_web_height: The height of the web.

4.18.1.47 CROSS SECTION PARAMETER

A **cross section parameter** describes an attribute of a shape part **cross section**. These include web and flange thicknesses, fillet radius, etc.

```
*)
ENTITY cross_section_parameter;
  having_cross_section_value : REAL;
  having_cross_section_code  : cross_section_code;
UNIQUE
  having_cross_section_code;
  having_cross_section_code, having_cross_section_value;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_cross_section_value: The value of the **cross section code**.

having_cross_section_code:

PROPOSITIONS:

1. Having cross section code must be UNIQUE.
2. The combination of having cross section code and having cross section value must be UNIQUE.

4.18.1.48 ENDCUT

The endcut provides the flange/web cut details at the end of a shape part.
Integration point: Form Features model.

```
*)
ENTITY endcut
  SUPERTYPE OF (transition_cut AND
                (non_parametric_endcut XOR
                parametric_endcut));
END_ENTITY;
(*
```

4.18.1.49 ENDCUT PARAMETER

An endcut parameter describes an attribute of a parametric endcut. For example, snipe and radius are two attributes of an endcut.

```
*)
ENTITY endcut_parameter;
  having_endcut_parameter_value : REAL;
  having_endcut_parameter_code  : endcut_parameter_code;
UNIQUE
  having_endcut_parameter_code;
  having_endcut_parameter_value, having_endcut_parameter_code;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_endcut_parameter_value: The value of the endcut parameter code.

having_endcut_parameter_code:

PROPOSITIONS:

1. Having endcut parameter code must be UNIQUE.
2. The combination of having endcut parameter value and having endcut parameter code must be UNIQUE.

4.18.1.50 PARAMETRIC ENDCUT

A parametric endcut is an endcut with the cut geometry specified by reference to a standard endcut and its associated endcut parameters.

```
*)
ENTITY parametric_endcut
  SUBTYPE OF (endcut);
  identified_by_parametric_endcut_id : parametric_endcut_id;
  defined_by_endcut_parameter        : SET [1:#] OF
                                           endcut_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_parametric_endcut_id:

defined_by_endcut_parameter: Set of endcut parameters.

4.18.1.51 NON-PARAMETRIC ENDCUT

A non parametric endcut is a type of endcut with the cutting geometry specified by free curves.

```
*)
ENTITY non_parametric_endcut
  SUBTYPE OF (endcut);
  with_web_curve_geometry          : ship_curve_geometry;
  with_lower_flange_curve_geometry : OPTIONAL ship_curve_geometry;
  with_upper_flange_curve_geometry : OPTIONAL ship_curve_geometry;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_web_curve_geometry: Cut path for web endcut.

with_lower_flange_curve_geometry: Cut path for lower flange endcut.

with_upper_flange_curve_geometry: Cut path for upper flange endcut.

4.18.1.52 TRANSITION CUT

```
*)
ENTITY transition_cut
  SUBTYPE OF (endcut);
END_ENTITY;
(*
```

4.18.1.53 NC MARK

An nc mark is a piece of text or contour marked on a plate part or shape part by a cam device such as a burning machine or robot. Such marking is typically performed by a punch marker, zinc marker or an ink-jet marker.

```
*)
ENTITY nc_mark
  SUPERTYPE OF (nc_text XOR
                trace_mark);
  identified_by_nc_mark_id : nc_mark_id;
  marking_plate_part       : OPTIONAL SET [1:#] OF plate_part;
  marking_shape_part       : OPTIONAL SET [1:#] OF shape_part;
WHERE
  has_at_least_one_of (marking_plate_part,
                       marking_shape_part);
  has_at_most_one_of (marking_plate_part,
                      marking_shape_part);
UNIQUE
  identified_by_nc_mark_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_nc_mark_id:

marking_plate_part:

marking_shape_part:

PROPOSITIONS:

1. Identified by nc mark id must be UNIQUE.

4.18.1.54 NC TEXT

An nc text is a type of nc mark consisting of a character string.

```
*)
ENTITY nc_text
  SUBTYPE OF (nc_mark);
  defined_with_nc_text_string : nc_text_string;
  having_by_nc_text_parameter : SET [1:#] OF nc_text_parameter;
  oriented_by_unit_vector     : ship_unit_vector;
  located_at_node             : ship_node;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

defined_with_nc_text_string: Text string.

identified_by_nc_text_parameter: Text attributes; size, font etc.

oriented_by_unit_vector: Direction (orientation) of text from origin node.

located_at_node: Origin of text string on part.

4.18.1.55 NC TEXT PARAMETER

An **nc text parameter** describes an attribute of an **nc text mark**, ie. size, font, etc.
Integration Point: Presentation.

```
*)
ENTITY nc_text_parameter;
  having_nc_text_parameter_code  : nc_text_parameter_code;
  having_nc_text_parameter_value : REAL;
UNIQUE
  having_nc_text_parameter_code;
  having_nc_text_parameter_code, having_nc_text_parameter_value;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_nc_text_parameter_code: Defining a text attribute, e.g text height.

having_nc_text_parameter_value: The value for the text parameter code.

PROPOSITIONS:

1. Having nc text parameter code must be UNIQUE.
2. The combination of having nc text parameter code and having nc text parameter value must be UNIQUE.

4.18.1.56 TRACE MARK

A **trace mark** is a numerically controlled (N/C) marking contour.

```
*)
ENTITY trace_mark
  SUBTYPE OF (nc_mark);
  with_mark_interval       : REAL;
  with_mark_length         : REAL;
  defined_on_path_segment  : path_segment;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_mark_interval: The spacing between incremental marks.

with_mark_length: The length of incremental marks.

defined_on_path_segment:

4.18.1.57 Library Parts

Parts which are used throughout the ship with geometrically similar shape and size may be modeled once in a part library and used again and again wherever needed. Chocks, brackets and gussets are examples of possible library parts. These may also be referred to as relocatable parts.

There are two types of library parts: parametric and non-parametric. Non-parametric library parts are defined with a specific geometry and always look the same wherever used. Parametric library parts are defined through the use of standard geometric shapes with a corresponding set of parameters. Values for these parameters are input at the time of usage, therefore, two different occurrences of the same parametric library part may appear physically different.

Library parts are specified by their library identification, by their occurrence (as a part), by their location (at a node), by their translation and orientation from the node via a transformation entity, and parameters if necessary. The actual geometry of the library part is not a part of this model. It is intended that the geometry be generated by the host CAD system from the above mentioned attributes. Library parts, being a subtype of part, all carry part attributes such as material, date-time modified/created, and association with a unit-assembly or sub-assembly.

Various issues concerning library part geometry need further attention. These issues involve the details of part geometry representation within a library, cross referencing between libraries of different systems, and methods for the exchange or merging of different libraries. Such issues are currently being addressed by other participants and within the Marine industry. It is intended that enhancements will be added to this model in the future.

Integration point: Presentation.

4.18.1.58 Library Part TYPE Definitions

4.18.1.58.1 LIBRARY ID

```
*)
TYPE library_id = STRING;
END_TYPE;
(*
```

4.18.1.58.2 LIBRARY PART ID

```
*)
TYPE library_part_id = STRING;
END_TYPE;
(*
```

4.18.1.58.3 LIBRARY VERSION

The library version identifies the version of the library being used, ie. Rev. A, Rev. B, etc.

```
*)
TYPE library_version = STRING;
END_TYPE;
(*
```

4.18.1.58.4 PART PARAMETER CODE

The parametric library part parameter code identifies the variables which define a parametric library part, e.g. t = thickness.

```
*)
TYPE part_parameter_code = STRING;
END_TYPE;
(*
```

4.18.1.59 SHIP PART LIBRARY

A ship part library is a collection of standard parts, also referred to as a standard parts catalog, which provides a pre-defined part for use in the ship product model.

```
*)
ENTITY ship_part_library;
  identified_by_library_version : library_version;
  identified_by_library_id      : library_id;
UNIQUE
  identified_by_library_id;
  identified_by_library_version, identified_by_library_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_library_version:

identified_by_library_id:

PROPOSITIONS:

1. Identified by library id must be UNIQUE.
2. The combination of identified by library version and identified by library id must be UNIQUE.

4.18.1.60 LIBRARY PART

A library part is a pre-defined part contained in a ship part library. There are two types of library parts, parametric and non-parametric. Chocks, brackets and gussets are examples of library parts.

```
*)
ENTITY library_part
  SUPERTYPE OF (non_parametric_library_part XOR
                parametric_library_part)
  SUBTYPE OF (part);
  identified_by_library_part_id      : library_part_id;
  oriented_by_transformation_matrix : ship_transformation_matrix;
  located_by_node                    : ship_node;
  defined_in_library                 : SET [1:#] OF
```

```
                                        ship_part_library;
UNIQUE
  identified_by_library_part_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_library_part_id:

oriented_by_transformation_matrix:

located_by_node:

defined_in_library:

PROPOSITIONS:

1. Identified by library part id must be UNIQUE.

4.18.1.61 NON-PARAMETRIC LIBRARY PART

A non-parametric library part is a type of library part whose geometry (shape, size) is the same for every occurrence.

```
*)
ENTITY non_parametric_library_part
  SUBTYPE OF (library_part);
END_ENTITY;
(*
```

4.18.1.62 PARAMETRIC LIBRARY PART

A parametric library part is a type of library part whose geometry is necessarily defined by both the library part id and parameter values associated with a particular library part occurrence. two parametric library parts with the same library part id may or may not be physically identical.

```
*)
ENTITY parametric_library_part
  SUBTYPE OF (library_part);
  defined_by_part_parameter : SET [1:#] OF part_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

defined_by_part_parameter: Set of part parameters.

4.18.1.63 PART PARAMETER

A parametric library **part parameter** describes the attributes associated with a parametric library part. These include shape and size.

```
*)
ENTITY part_parameter;
  having_part_parameter_value : REAL;
  having_part_parameter_code  : part_parameter_code;
UNIQUE
  having_part_parameter_code;
  having_part_parameter_code, having_part_parameter_value;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_part_parameter_value:

having_part_parameter_code:

PROPOSITIONS:

1. **Having part parameter code** must be UNIQUE.
2. The combination of **having part parameter code** and **having part parameter value** must be UNIQUE.

4.18.1.64 Ship Structural Joints

A joint is a connection between plate parts and/or shape parts. Usually the connection is made by welding but may be bolted or riveted. Joints in this model carry all the necessary relationships and information for joining two or more parts.

4.18.1.65 Structural Joint TYPE Definitions

4.18.1.65.1 BOLT PARAMETER CODE

A **bolt parameter code** identifies a variable which is used to define an attribute of a bolt, ie. d = diameter, t = threads/inch, etc.

```
*)
TYPE bolt_parameter_code = STRING;
END_TYPE;
(*
```

4.18.1.65.2 BOLT PROCESS

The **bolt process** describes the process for a bolted joint. This includes references to standards or detailed drawings providing bolting pattern, tightening torque, etc.

```
*)
TYPE bolt_process = STRING;
END_TYPE;
(*
```

4.18.1.65.3 INSPECTION PROCEDURE

An **inspection procedure** provides the description for the inspection of a joint. The **inspection procedure** may include references to standard inspection procedures, ie. magnaflux, visual, x-ray, etc.

```
*)
TYPE inspection_process = STRING;
END_TYPE;
(*
```

4.18.1.65.4 JOINT ID

A **joint id** is the unique identification of a **joint.**

```
*)
TYPE joint_id = STRING;
END_TYPE;
(*
```

4.18.1.65.5 JOINING PROCEDURE

The **joining procedure** describes the joining of parts at a **joint. Joining procedures** may include references to standard joining procedures.

```
*)
TYPE joining_procedure = STRING;
END_TYPE;
(*
```

4.18.1.65.6 RIVET PARAMETER CODE

A **rivet parameter code** identifies the variables defining the attributes of a rivet. for example, d = diameter, l = length, etc.

```
*)
TYPE rivet_parameter_code = STRING;
END_TYPE;
(*
```

4.18.1.65.7 RIVET PROCESS

The **rivet process** describes the process for a riveted joint. This includes references to standards or detailed drawings providing riveting pattern, etc.

```
*)
TYPE rivet_process = STRING;
END_TYPE;
(*
```

4.18.1.65.8 STANDARD DETAILS REFERENCE

A **standard details reference** provides references to standard details, drawings and procedures identifying information for a **joint**.
Integration point: External Reference.

```
*)
TYPE standard_details_reference = STRING;
END_TYPE;
(*
```

4.18.1.65.9 WELD TYPE

A **weld type** describes the type of welded joint, e.g double fillet, single fillet, butt etc.

```
*)
TYPE weld_type = STRING;
END_TYPE;
(*
```

4.18.1.65.10 WELD PROCESS

The **weld process** describes the process for a welded joint. This includes references to standards and detailed drawings.

```
*)
TYPE weld_process = STRING;
END_TYPE;
(*
```

4.18.1.66 JOINT

A **joint** is used to define the physical connection between **shape parts** and **plate parts**. **Joints** occur at **path segments** and **ship nodes**. The type of **joint** is identified as either **path joint** or **nodal joint**.

```
*)
ENTITY joint
  SUPERTYPE OF ((nodal_joint XOR
                 path_joint) AND
                 (bolt_joint XOR
                 rivet_joint XOR
                 weld_joint));
  with_inspection_process      : OPTIONAL inspection_process;
  identified_by_joint_id       : joint_id;
```

```
  joined_by_joining_procedure : joining_procedure;
  referring_to_standard_details_reference :
                                standard_details_reference;
  penetrating_cutout_hole     : OPTIONAL SET [1:#] OF
                                cutout_hole;
UNIQUE
  identified_by_joint_id;
  referring_to_standard_details_reference;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_inspection_process:

identified_by_joint_id:

joined_by_joining_procedure:

referring_to_standard_details_reference:

penetrating_cutout_hole:

PROPOSITIONS:

1. Identified by joint id must be UNIQUE.
2. Referring to standard details reference must be UNIQUE.

4.18.1.67 NODAL JOINT

A nodal joint is a type of joint. A nodal joint is a connection between at least two parts occurring at a ship node.

```
*)
ENTITY nodal_joint
  SUBTYPE OF (joint);
  joining_plate_part   : OPTIONAL SET [1:#] OF plate_part;
  joining_library_part : OPTIONAL SET [1:#] OF library_part;
  joining_shape_part   : OPTIONAL SET [1:#] OF shape_part;
  located_at_node      : ship_node;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

joining_plate_part:

joining_library_part:

joining_shape_part:

located_at_node:

4.18.1.68 PATH JOINT

A path joint is a type of joint. A path joint is a connection between at least two part (shape or plate) edges.

```
*)
ENTITY path_joint
  SUBTYPE OF (joint);
  joining_shape_part_edge : OPTIONAL SET [1:#] OF shape_part_edge;
  joining_path_segment    : OPTIONAL SET [1:#] OF path_segment;
  joining_plate_part_edge : OPTIONAL SET [1:#] OF plate_part_edge;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

joining_shape_part_edge:

joining_path_segment:

joining_plate_part_edge:

4.18.1.69 BOLT JOINT

A bolt joint is a type of joint using threaded fastenings to secure two or more parts together.

```
*)
ENTITY bolt_joint
  SUBTYPE OF (joint);
  described_by_bolt_process : bolt_process;
  defined_by_bolt_parameter : SET [1:#] OF bolt_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

described_by_bolt_process:

defined_by_bolt_parameter:

4.18.1.70 BOLT PARAMETER

A bolt parameter describes an attribute of a bolt used in a bolt joint, ie. bolt diameter, bolt length, threads/inch etc.

```
*)
ENTITY bolt_parameter;
  having_bolt_parameter_value : REAL;
  having_bolt_parameter_code  : bolt_parameter_code;
UNIQUE
  having_bolt_parameter_code;
  having_bolt_parameter_code, having_bolt_parameter_value;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_bolt_parameter_value: The specific value for a bolt parameter code.

having_bolt_parameter_code:

PROPOSITIONS:

1. Having bolt parameter code must be UNIQUE.
2. The combination of having bolt parameter code and having bolt parameter value must be UNIQUE.

4.18.1.71 RIVET JOINT

A rivet joint is a type of joint using rivets to securely fasten two or more parts together.

```
*)
ENTITY rivet_joint
  SUBTYPE OF (joint);
  with_rivet_process    : rivet_process;
  with_rivet_parameter : SET [1:#] OF rivet_parameter;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_rivet_process:

with_rivet_parameter:

4.18.1.72 RIVET PARAMETER

A rivet parameter contains an attribute of a rivet used in a rivet joint, ie. rivet diameter, rivet length, etc.

```
*)
ENTITY rivet_parameter;
  having_rivet_parameter_value : REAL;
  having_rivet_parameter_code  : rivet_parameter_code;
UNIQUE
  having_rivet_parameter_code;
  having_rivet_parameter_code, having_rivet_parameter_value;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_rivet_parameter_value: The specific value for the rivet parameter code.

having_rivet_parameter_code:

PROPOSITIONS:

1. Having rivet parameter code must be UNIQUE.
2. The combination of having rivet parameter code and having rivet parameter value must be UNIQUE.

4.18.1.73 WELD JOINT

```
*)
ENTITY weld_joint
  SUBTYPE OF (joint);
  with_weld_process : weld_process;
  of_weld_size      : REAL;
  with_weld_type    : weld_type;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

with_weld_process:

of_weld_size: The size of the weld joint.

with_weld_type:

4.18.1.74 Structural Openings

Structural openings consist of a variety of different types, namely; lightening or access holes, distribution system penetrations, and cutouts through which stiffeners pass. All structural openings have been classified into these three basic types for the following reasons. Access/lightening holes are either voids or have doors/closures attached to them; distribution system penetrations are openings through which either vent duct, electrical cable, or pipe penetrate; and cutouts are holes through which other structure penetrates.

A structural opening can be defined as parametric opening or a non- parametric opening. Non-parametric openings have their location and geometry defined by a path segment. Parametric openings are identified by a parametric opening type, a reference to a standard library opening or a macro. Their locations are specified by nodes and they are oriented by vectors.

4.18.1.75 Structural Opening TYPE Definitions

4.18.1.75.1 HOLE ID

A hole id is the unique identifier for a structural opening.

```
*)
TYPE hole_id = STRING;
END_TYPE;
(*
```

4.18.1.75.2 OPENING PARAMETER CODE

The opening parameter code identifies the variables which define a parametric opening, ie. r = radius.

```
*)
TYPE opening_parameter_code = STRING;
END_TYPE;
(*
```

4.18.1.75.3 PARAMETRIC OPENING ID

A parametric opening id is the unique identifier of a parametric (standard) structural opening. It is the name of a standard library opening or the name of the macro defining the opening.

```
*)
TYPE parametric_opening_id = STRING;
END_TYPE;
(*
```

4.18.1.75.4 DISTRIBUTION SYSTEM PART ID

A distribution system part id is the unique identifier of a distribution system part.

```
*)
TYPE distribution_system_part_id = STRING;
END_TYPE;
(*
```

4.18.1.75.5 PENETRATION PART ID

A penetration part id is the unique identifier of a penetration part.

```
*)
TYPE penetration_part_id = STRING;
END_TYPE;
(*
```

4.18.1.76 STRUCTURAL OPENING

```
*)
ENTITY structural_opening
  SUPERTYPE OF ((system_penetration_hole XOR
                 access_lightening_hole OR
                 cutout_hole) AND
                (non_parametric_opening XOR
                 parametric_opening));
  identified_by_hole_id       : hole_id;
  defined_on_bounded_surface  : OPTIONAL ship_bounded_surface;
  having_edge_preparation     : OPTIONAL edge_preparation;
```

```
UNIQUE
  identified_by_hole_id;
WHERE
  has_at_least_one_of (cutting_plate_part,
                       cutting_shape_part);
  has_at_most_one_of (cutting_plate_part,
                      cutting_shape_part);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_hole_id:

defined_on_bounded_surface:

prepared_by_edge_preparation:

PROPOSITIONS:

1. Identified by hole id must be UNIQUE.

4.18.1.77 PARAMETRIC OPENING

A parametric opening is a type of standard structural opening whose geometry is defined by reference to a parametric opening id along with parameters and an orientation for the particular opening occurrence.

```
*)
ENTITY parametric_opening
  SUBTYPE OF (structural_opening);
  identified_by_parametric_opening_id : parametric_opening_id;
  located_at_node                     : ship_node;
  defined_by_opening_parameter        : opening_parameter;
  oriented_by_unit_vector             : ship_unit_vector;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_parametric_opening_id: Standard opening id or macro name.

located_at_node: Opening location.

defined_by_opening_parameter: Opening parameters.

oriented_by_unit_vector: Opening orientation.

4.18.1.78 OPENING PARAMETER

An **opening parameter** is one of a set of one or more attributes defining the size and shape of a parametric (structural) opening. An opening parameter, for example, would define the radius of a circular opening.

```
*)
ENTITY opening_parameter;
  having_opening_parameter_value : REAL;
  having_opening_parameter_code  : opening_parameter_code;
UNIQUE
  having_opening_parameter_code;
  having_opening_parameter_code, having_opening_parameter_value;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

having_opening_parameter_value: Value (e.g 15.0).

having_opening_parameter_code: Code (e.g Radius).

PROPOSITIONS:

1. **Having opening parameter code** must be UNIQUE.
2. The combination of **having opening parameter code** and **having opening parameter value** must be UNIQUE.

4.18.1.79 NON-PARAMETRIC OPENING

A **non-parametric opening** is a type of **structural opening** whose opening is defined by a **path segment** representing the hole contour.

```
*)
ENTITY non_parametric_opening
  SUBTYPE OF (structural_opening);
  defined_by_path_segment : path_segment;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

defined_by_path_segment: A **path segment.**

4.18.1.80 CUTOUT HOLE

A **cutout hole** is a type of **structural opening.** It is generally a hole in a **plate part** or a **shape part** to allow the penetration by a **shape part.** Other uses are **rathole, air escape** and **oil stop.** These holes are not penetrated by shape parts.

```
*)
ENTITY cutout_hole
  SUPERTYPE OF (air_escape XOR
                oil_stop XOR
                rathole)
  SUBTYPE OF (structural_opening);
END_ENTITY;
(*
```

4.18.1.81 AIR ESCAPE

An **air escape** is a type of **cutout hole**. Its function is to prevent air pockets from forming when fluids are being transferred into a tank.

```
*)
ENTITY air_escape
  SUBTYPE OF (cutout_hole);
END_ENTITY;
(*
```

4.18.1.82 OIL STOP

An **oil stop** is a type of **cutout hole** used in combination with a **weld joint** to form a fluid barrier between welded parts.

```
*)
ENTITY oil_stop
  SUBTYPE OF (cutout_hole);
END_ENTITY;
(*
```

4.18.1.83 RATHOLE

A **rathole** is a type of **cutout hole**.

```
*)
ENTITY rathole
  SUBTYPE OF (cutout_hole);
END_ENTITY;
(*
```

4.18.1.84 ACCESS/LIGHTENING HOLE

An **access lightening hole** is a type of **structural opening** used primarily for accessing areas of a **hull** or for lightening structure. **Access lightening holes** may be penetrated by **distribution system parts** (a non-tight penetration).

```
*)
ENTITY access_lightening_hole
```

```
  SUBTYPE OF (structural_opening);
  penetrated_by_distribution_system_part : OPTIONAL SET [1:#] OF
                                           distribution_system_part;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

penetrated_by_distribution_system_part: Set of distribution system parts which penetrate the hole.

4.18.1.85 SYSTEM PENETRATION HOLE

A system penetration hole is a type of structural opening that is penetrated by a distribution system part or a penetration part. This type of opening may be tight or non-tight.

```
*)
ENTITY system_penetration_hole
  SUBTYPE OF (structural_opening);
  penetrated_by_distribution_system_part : SET [1:#] OF
                                           distribution_system_part;
  penetrated_by_penetration_part         : penetration_part;
UNIQUE
  penetrated_by_penetration_part;
WHERE
  has_at_least_one_of (penetrated_by_penetration_part,
                       penetrated_by_distribution_system_part);
  has_at_most_one_of (penetrated_by_distribution_system_part,
                      penetrated_by_penetration_part);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

penetrated_by_distribution_system_part:

penetrated_by_penetration_part:

PROPOSITIONS:

1. Penetrated by penetration part must be UNIQUE.

4.18.1.86 PENETRATION PART

A penetration part is a sleeve or insert enclosing a distribution system part in locations where the distribution system part penetrates a structural part.

```
*)
ENTITY penetration_part;
  identified_by_penetration_part_id      : penetration_part_id;
  penetrating_system_penetration_hole    : system_penetration_hole;
  penetrated_by_distribution_system_part : distribution_system_part;
```

```
UNIQUE
  identified_by_penetration_part_id;
  penetrating_system_penetration_hole;
  penetrated_by_distribution_system_part;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_penetration_part_id: The unique identification of a penetration part.

penetrating_system_penetration_hole:

penetrated_by_distribution_system_part:

PROPOSITIONS:

1. identified by penetration part idmust be UNIQUE.
2. penetrating system penetration hole must be UNIQUE.
3. penetrated by distribution system part must be UNIQUE.

4.18.1.87 DISTRIBUTION SYSTEM PART

A distribution system part is a part that distributes fluid, electricity, air, etc. A distribution system part may be enclosed by a penetration part (a sleeve or insert) when it penetrates a structural part.
Integration point: NIDDESC Outfit Model; Distributive System Part entity.

```
*)
ENTITY distribution_system_part;
  identified_by_distribution_system_part_id :
                                        distribution_system_part_id;
UNIQUE
  identified_by_distribution_system_part_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identified_by_distribution_system_part_id: -

PROPOSITIONS:

1. Identified by distribution system part id must be UNIQUE.

4.18.1.88 Ship Structure FUNCTION Definitions

4.18.1.88.1 HAS AT MOST ONE OF

```
*)
FUNCTION
  has_at_most_one_of ( arg1, arg2 : GENERIC) : LOGICAL;
 (* to be written *)
END_FUNCTION;
(*
```

4.18.1.88.2 HAS AT LEAST ONE OF

```
*)
FUNCTION
  has_at_least_one_of ( arg1, arg2 : GENERIC) : LOGICAL;
 (* to be written *)
END_FUNCTION;
(*
```

4.18.1.89 Classification Structure for Ship Structural Model

The following indented listing provides the Classification Structure for the entities in the Ships Structural model.

```
ASSEMBLY ID
BOLT PARAMETER
COMPARTMENT
CROSS SECTION
  NON STANDARD CROSS SECTION
  STANDARD CROSS SECTION
CROSS SECTION PARAMETER
DISTRIBUTION SYSTEM PART
EDGE PREPARATION
ENDCUT
  NON PARAMETRIC ENDCUT
  PARAMETRIC ENDCUT
  TRANSITION CUT
ENDCUT PARAMETER
HULL
JOINT
  BOLT JOINT
  NODAL JOINT
  PATH JOINT
  RIVET JOINT
  WELD JOINT
MOLDED CURVE
NC MARK
  NC TEXT
  TRACE MARK
NC TEXT PARAMETER
OPENING PARAMETER
PART
  LIBRARY PART
    NON PARAMETRIC LIBRARY PART
    PARAMETRIC LIBRARY PART
  PLATE PART
  SHAPE PART
PART FLANGE
PART PARAMETER
PATH SEGMENT
```

```
PENETRATION PART
PLATE PART EDGE
RIVET PARAMETER
SHAPE CLEARANCE
SHAPE ORIENTATION
SHAPE PART EDGE
SHIP BOUNDED SURFACE
  SHIP CURVED SURFACE
    SHIP BEZIER SURFACE
    SHIP BSPLINE SURFACE
  SHIP ELEMENTARY SURFACE
    SHIP PLANAR SURFACE
SHIP CURVE GEOMETRY
SHIP DATE TIME
SHIP PART LIBRARY
SHIP MATERIAL
SHIP NODE
SHIP POSITION POINT
SHIP TRANSFORMATION MATRIX
SHIP UNIT VECTOR
STRUCTURAL OPENING
  ACCESS LIGHTENING HOLE
  CUTOUT HOLE
    AIR ESCAPE
    OIL STOP
    RATHOLE
  NON PARAMETRIC OPENING
  PARAMETRIC OPENING
  SYSTEM PENETRATION HOLE
SURFACE EDGE
SURFACE ID
SYSTEM
  STRUCTURAL SYSTEM
UNIT ASSEMBLY
  SUB ASSEMBLY

*)
  END_SCHEMA; -- end of ipim_ship_structure_schema

END_SCHEMA; -- end of AEC schema
(*
```

4.19 Electrical Applications

Collected here are the Electrical Applications.

```
*)
SCHEMA ipim_electrical_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_electrical_schematic_schema,
         ipim_electrical_functional_schema,
         ipim_lep_schema);
(*
```

4.19.1 Electrical Functional Model

This model represents the portion of the Cal Poly Electrical Functional model (A/EFTL/A) voted to Draft status by the Electrical Applications Committee on 7/14/88. It has been included in this document in order to provide increased model visibility.

4.19.1.1 Integration

The model is integrated with the entities **product item functional definition** and **shape element** from the **ipim shape interface schema**; otherwise the model is free standing. The model has not yet become concerned with the data which tracks changes in design. When the concepts of "version" of designs are considered, the model is expected to undergo some change.

4.19.1.2 Scope

The Cal Poly Task Team adopted the notion that there is a logical separation of the functional description of product from the physical description. These two descriptions then have linkages or "mappings" to each other. The EXPRESS model represented here is the functional part of electrical product. The Task Team started with an initial scope that limited its efforts to the data which represents the data defining the design of electrical and electronic products at the point of handoff for fabrication and assembly.

4.19.1.3 Model Overview

The team has produced a data model which is believed to establish a basic structure of data within which the functional description of electrical product can be developed. There are three fundamental parts of this model: functional hierarchy, characteristics and behavior, and logical connectivity. The EXPRESS version of the model contained within this document describes functional hierarchy and logical connectivity only. The functional hierarchy represents the functional "bill of material." The logical connectivity describes the way functional units are connected in the logical sense (Link, Port and Electrical Node relationships). The logical connectivity part of this model is believed to be complete and stable. The EXPRESS model does not cover the behavior part of the Cal Poly model.

4.19.1.4 Functional Hierarchy

This is the core structure of the functional model. It represents the idea that a functional description of a product can be, and generally is, composed of large units of function which divide into smaller units of function until the designer arrives at his most discrete units. The design process probably is an iterative one of dividing large units into smaller and combining small units into a desired larger function.

Every identifiable unit of function appears as an instance of a **defined functional unit**. A **defined functional unit** may be a component of a higher functional assembly and it may have components of its own.

The functional hierarchy is described by the cluster of entities in Figure 8.

Each instance of a **defined functional sub unit occurrence** represents the data that a **defined functional unit** is a component of another **defined functional unit**. The above two entities with their relationships (observe membership cardinality in **defined functional sub unit occurrence**) provide the model for the data which describes all of the levels of assembly.

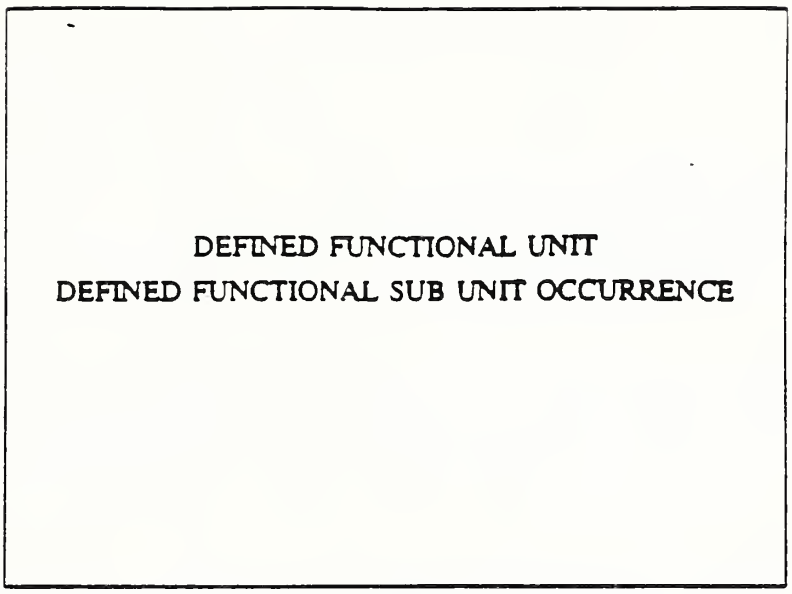


Figure 8: ENTITY CLUSTER #1

4.19.1.5 Connectivity

Logical access to functionality and electrical linkage are the important concepts in connectivity. Logical access occurs through functional ports. These functional ports are instanced as electrical nodes which logically link together to form nets. Of primary concern are two kinds of access: external access and boundary access. External access is concerned with connecting the electrical node of some functional unit with a net contained within the next level of functional assembly. Boundary access is concerned with making nets internal to a functional unit accessible to a higher level assembly through the ports of the functional unit. The connectivity model provides entities and relations to accommodate both kinds of access. Connectivity is described by the cluster of entities in Figure 9.

A **defined functional unit** which to the definer is a "black box" with no knowledge of its details has no known connectivity and therefore contains no **defined electrical logical links**. A **defined functional unit** which has known details and connectivity will contain one or more **defined electrical logical links**. A **defined electrical logical link** must contain at least one **electrical node** and may contain more. This is established by one or more related instances of **electrical node**.

A **defined functional unit** contains one or more **defined functional port**. A **defined functional port** is a logical place of access to the function of the **defined functional unit**. This is consistent with the definition in IEEE Standard 100. Whenever a logical point of access is defined by the designer as a place of functional connect, or by a test engineer as a point for functional test, that act creates a **defined functional port**.

An **electrical node** is a specific instance of a **defined functional port** within a next higher functional assembly.

Defined functional port and **electrical node** cannot be members of more than one **defined electrical logical link**. Since the **defined electrical logical link** represents logical points that are electrically common, joining two links through a common node would have the effect of joining two links into a single link.

Ports of **defined function units** which have not been used in higher functional assemblies are not members of links. **Electrical nodes** which are unused in the next assemblies are also not members

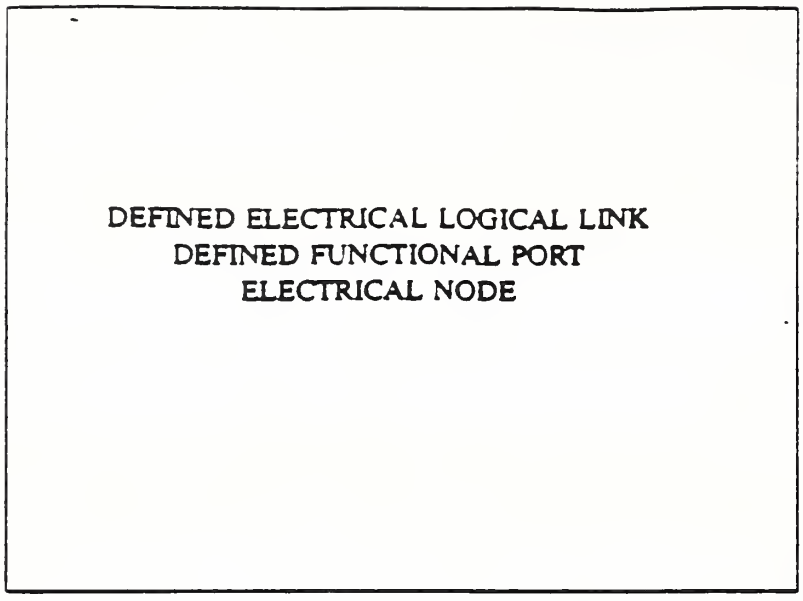


Figure 9: ENTITY CLUSTER #2

of defined electrical logical link.

At this point, the model provides for the fundamental data of logical connectivity. When the physical solution data model has been developed, there will be some mapping from the defined electrical logical link to the entities which establish the physical description where that logical connectivity is implemented. For example: The Layered Electrical Product model (LEP) maps physical elements on a board layer (e.g., traces on a PWB) to defined electrical logical link.

```
*)
SCHEMA ipim_electrical_functional_schema;

ASSUME (ipim_shape_interface_schema);
(*
```

4.19.1.6 DEFINED FUNCTIONAL UNIT

Data about the functional, or performance, aspects of a portion of a system. A defined functional unit may be a sub unit of another defined functional unit and it may have sub units of its own.

```
*)
ENTITY defined_functional_unit;
    external_ports     = LIST [1:#] OF defined_functional_port;
    boundary_nets      = SET [0:#] OF
                          defined_electrical_logical_link;
    sub_units          = SET [0:#] OF
                          defined_functional_sub_unit_occurrence;
    physical_solution : SET [0:#] OF
                          product_item_functional_definition;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

ports: The set of functional accesses to the **defined functional unit.**

nets: The set of all interconnection nets internal to the **defined functional unit.**

sub_units: The set of all sub units from which the **defined functional unit** is functionally composed.

physical_solution: The physical materializations of the **defined functional unit.**

4.19.1.7 DEFINED FUNCTIONAL SUB UNIT OCCURRENCE

Data that a particular **defined functional unit** occurs as a component of another **defined functional unit.** There is an instance of this entity for each occurrence of a **defined functional unit** within a higher level **defined functional unit.** This provides a functional view of the various levels of component/assembly product relationship (the functional hierarchy).

```
*)
ENTITY defined_functional_sub_unit_occurrence;
    nodes              : LIST [1:#] OF electrical_node;
    parent_definition  : defined_functional_unit;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

nodes: The set of nodes that are instanced when a particular **defined functional unit** occurs as a component.

parent_definition: Defined functional unit whose definition is being instanced.

```
*)
RULE definition_instance FOR (defined_functional_unit,
  defined_functional_sub_unit_occurrence);
    defines_details_of_this_occurrence :
     defined_functional_unit;
    has_as_instances : SET [1:#] OF
      defined_functional_sub_unit_occurrence;
WHERE
    FOR_ALL (is_a_component_of_this_unit,
      (is_a_component_of_this_unit.parent_definition <>
      defines_details_of_this_occurrence));
END_RULE;
(*
```

PROPOSITIONS:

1. An instance of **defined sub unit occurrence** cannot exist as a component without first being an instance of **defined functional unit.** This means that it must be defined before it can be instanced as a component. It has exactly one definition.

2. A **defined functional unit** cannot be used in its own definition.

4.19.1.8 DEFINED-ELECTRICAL LOGICAL LINK

Data about the logical electrical connectivity within a Functional Unit. It establishes a set of two or more **electrical nodes** or one **defined functional port** and one or more **electrical nodes** as being electrically in common.

```
*)
ENTITY defined_electrical_logical_link;
    net  : SET [1:#] OF electrical_node;
    port : OPTIONAL defined_functional_port;
UNIQUE
  port;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

net: The set of **electrical nodes** which are electrically in common.

port: A **defined functional port** of the next higher assembly that is connected to the **defined electrical logical link** that makes the **defined electrical logical link** accessible externally to the next higher assembly.

```
*)
RULE ports_of_functional_units FOR (defined_functional_unit,
  defined_electrical_logical_link);
    contains_in_its_definition_the_logical_link :
      defined_functional_unit
    is_a_part_of_definition_of_functional_unit :
      SET [0 : #] OF defined_electrical_logical_link;
WHERE
  IN(is_a_part_of_definition_of_functional_unit.node,
    contains_in_its_definition_the_logical_link.external_ports);
END_RULE;
(*
```

PROPOSITIONS:

1. The **port**, if it occurs, must be UNIQUE.
2. An instance of **defined electrical logical link** cannot exist apart from the instance of **defined functional unit** of which it is a component.
3. The **port** must be an instance of **defined functional port** of the **defined functional unit** of which this **defined electrical logical link** is a component.

4.19.1.9 DEFINED FUNCTIONAL PORT

Data describing the logical accessibility to one of the functions of a **defined functional unit**. (These ports are what is represented by lines or other graphic symbols identifying inputs and outputs for electrical devices.) The **defined functional port** is characteristic of a **defined functional unit** independent of its usage. The **defined functional port** can be thought of as a window through which the boundary networks of the functional unit are seen.

```
*)
ENTITY defined_functional_port;
    physical_solution : SET [0:#] OF shape_element;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

physical_solution: A set of **shape elements** that physically materialize the **defined functional port.**

4.19.1.10 ELECTRICAL NODE

Data about a logical (functional) accessibility to the functionality of a specific occurrence of a **defined functional unit.** An **electrical node** is a **defined functional port** that becomes instanced in a higher assembly.

```
*)
ENTITY electrical_node;
    port       : defined_functional_port;
    occurrence : defined_functional_sub_unit_occurrence;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

port: The **defined functional port** that is being instanced.

occurrence: The **defined functional sub unit occurrence** associated with **port.**

4.19.1.11 PRODUCT ITEM FUNCTIONAL DEFINITION

```
*)
ENTITY product_item_functional_definition;
    dummy_attribute : undefined;
END_ENTITY;
(*
```

4.19.1.12 Electrical Functional Classification Structure

The following indented listing provides the classification structure for the entities in the Electrical Functional model.

```
DEFINED ELECTRICAL LOGICAL LINK
DEFINED FUNCTIONAL PORT
DEFINED FUNCTIONAL UNIT
DEFINED FUNCTIONAL SUB UNIT OCCURRENCE
ELECTRICAL NODE
PRODUCT ITEM FUNCTIONAL DEFINITION

*)
END_SCHEMA;  -- end ELECTRICAL FUNCTIONAL schema
(*
```

4.19.2 Electrical Schematics

4.19.2.1 Electrical Schematic Application — User View

4.19.2.1.1 Definition

The schematic is a symbolic representation of component parts and their electrical connections. The schematic may be for any hierarchical level of product definition, and becomes part of the packaging requirements at that level. Schematics may also contain details of related mechanical nature (e.g., heat sinks, connectors, etc.) and transmission of optical, magnetic or microwave energy.

4.19.2.1.2 Inputs (Sources of Design Constraints)

- Block diagram, system or subsystem with target block identified
- Interface requirements (e.g., signals and power)
- Mechanical package requirements (e.g., chip, hybrid micro-electronic assembly or printed board size and mounting)
- Design (and product) constraints and characteristics (e.g., specifications, system equations, test requirements)
- Schedules and/or budget
- Approved (or preferred) parts lists
- Symbol set and drafting standards

4.19.2.1.3 Items Schematic Relates to During Design

- Design block diagram
- Boolean operators
- Detail equations/transformations
- Static or dynamic models and simulations

4.19.2.1.4 Schematic Constituents

SYMBOL IDENTIFICATION: part family, part number, part values, part tolerance and reference designator (identifies an instance of the part, and may be later changed to match physical package assignment).

SYMBOL CONNECTION POINT IDENTIFICATION: function code (e.g., Q, D1, D2, VCC, GND) and pin number (the latter may be different after physical design is completed).

UTILITY SYMBOLS: non-part symbols of (usually) non-electrical requirements or parts (e.g., optical energy source, heat sink).

4.19.2.1.5 Schematic Outputs

PICTORIAL: used to review circuit design and as part of final product documentation.

NET LIST: a topology derived by examining the logical relations (links) created by the connection lines (joins). Resultant list must be formatted, and contain information, for each use (e.g., physical packaging process, circuit analysis methodology, test development system). Minimal net list information includes part identification (which must match identification of parts in the library or each using system), symbol connection point identification, and unique link identification. May be augmented with electrical characteristics (waveform, delay, etc.).

BILL-OF-MATERIALS: a list of parts cited in schematic. Used for stress analysis, packaging the physical circuit and in documentation parts lists.

4.19.2.2 Model Notes for Reviewers

Inasmuch as the model intended use is a logical view of the data, independent of existing systems or methods, for development of a neutral product data exchange structure, the reviewer is asked to keep in mind several important modeling constraints.

The entities and attributes are to reflect the data inherent in a schematic, as opposed to the information conveyed by a schematic. An example is that the User View proceeding the logical IDEF-1 model describes a netlist. The netlist is here recognized as information assembled by a query against schematic data. In turn, the model must be capable of supporting such a query. Elements of information found in the netlist are found in the **Network**, **Symbol Connection Place**, and **Symbol Instance** entity classes.

The model must exist independently of implementations. The entity classes in the logical model must be equally valid for a pencil-drawn schematic or a CAE system with nodal data structures. The **Text Template** entity found in the IGES PWB model has no validity under this rule and has not been included in this model. The data concept of a "place where a connection line can terminate" would be valid, and has been included as a **Line Connection Place** entity.

The model must provide for a union with other related models. This has resulted in entity classes which also belong in other models, but are included to show the key migration and the relation classes. The entities **Product Assembly Definition** and **Component Part** are examples of such entities from other models.

The entity classes involved in circuit analysis have been developed more than necessary for a schematic because of the high degree of coupling with schematic entities. While packaging requires information from the schematic data, analysis is an interactive data-exchange with the schematic. No requirement for conveying analysis data in a product assembly structure should be assumed by this.

Several entity classes have been created as part of the normalization processes. These entities are not included in the Overview FEO. An example is **Electrical Symbol Instance**, which provides a home for the **Reference Designator** (key) attribute. This attribute can be null for some **Symbol Instances** (e.g., utility symbols), creating a separate entity class.

The opinion is also offered that a schematic (all of this model) is not part of a "product definition": Schematics exist as a design aid and as reference documentation.

4.19.3 Electrical Schematic Model

The model provided here is an EXPRESS rendition of the IDEF model in the document *IGES Schematic Logical Schema*, dated 9 October 1985.

```
*)
SCHEMA ipim_electrical_schematic_schema;

  EXPORT EVERYTHING;
  ASSUME (ipim_geometry_schema,
          ipim_drafting_schema);
(*
```

4.19.3.1 Electrical Schematic TYPE Definitions

4.19.3.1.1 COMPONENT SELECT

```
*)
TYPE component_select = SELECT
     (component_part,
      product_assembly_definition);
END_TYPE;
(*
```

4.19.3.1.2 INPUT OUTPUT ENUMERATION

```
*)
TYPE input_output_enumeration = ENUMERATION OF
     (input,
      output);
END_TYPE;
(*
```

4.19.3.2 PRODUCT ASSEMBLY DEFINITION

That collection of information necessary to define a complex (hierarchically defined) product for purposes of documentation, analysis, manufacturing, testing and procuring.

```
*)
ENTITY product_assembly_definition;
  document_number      : STRING;
  schematic_reference  : schematic;
  component_list       : LIST [1:#] OF component_select;
UNIQUE
  document_number;
  schematic_reference;
WHERE
  NOT (product_assembly_definition IN component_list);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

document_number: The unique reference to this product assembly definition.

schematic_reference: Reference to a schematic representing the product.

components: The components making up this product.

PROPOSITIONS:

1. **Document number** must be UNIQUE.
2. **Schematic reference** must be UNIQUE.
3. A **product assembly definition** cannot reference itself.
4. A **schematic** may be referenced by only one **product assembly definition.**

4.19.3.3 SCHEMATIC

A symbolic representation of a function.

```
*)
ENTITY schematic;
  schematic_number  : STRING;
  document_number   : STRING;
  network_list      : LIST [1:#] OF network;
  instance_list     : LIST [1:#] OF symbol_instance;
  component_list    : LIST [1:#] OF schematic;
UNIQUE
  schematic_number;
WHERE
  NOT (schematic IN component_list);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

schematic_number: The unique identifier for this **schematic.**

document_number: A secondary identification for the **schematic.**

network_list: The **networks** appearing on this **schematic.**

instance_list: The component symbols appearing on this **schematic.**

component_list: The components appearing on this **schematic.**

PROPOSITIONS:

1. **Schematic number** must be UNIQUE.
2. A **schematic** may not reference itself.

4.19.3.4 SYMBOL LIBRARY

```
*)
ENTITY symbol_library;
  name           : library_name_structure;
  component_list : LIST [1:#] OF symbol;
UNIQUE
  name;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: Library name; consists of name and version.

component_list: List of contents of library (**symbols**).

PROPOSITIONS:

1. Name must be UNIQUE.

4.19.3.5 LIBRARY NAME STRUCTURE

```
*)
ENTITY library_name_structure;
  name    : STRING;
  version : INTEGER;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: String (name) portion of unique library identifier.

version: Version number of library.

4.19.3.6 SYMBOL

A **symbol** is a 2-dimensional figure commonly accepted as a representation for a part's functionality.

```
*)
ENTITY symbol;
  name       : STRING;
  line_list  : LIST [1:#] OF symbol_line;
  arc_list   : LIST [1:#] OF symbol_arc;
UNIQUE
  name;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: Symbol identifier.

line_list: List of line entities in symbol.

arc_list: List of arc entities in symbol.

PROPOSITIONS:

1. Name must be UNIQUE.

4.19.3.7 SYMBOL INSTANCE

```
*)
ENTITY symbol_instance;
  symbol_location               : cartesian_two_coordinate;
  sheet_number                  : drawing_sheet;
  symbol_type                   : symbol;
  electrical_data               : OPTIONAL
                                   electrical_symbol_instance;
  connection_list               : LIST [1:#] OF
                                   symbol_connection_place;
  corresponding_component_part  : OPTIONAL component_part;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

symbol_location: Geometrical location of symbol on schematic sheet.

sheet_number: Sheet on which symbol instance appears.

symbol_type: Reference to symbol prototype (definition).

electrical_data: Electrical data for symbol.

connection_list: List of electrical connect points for symbol.

corresponding_component_part: Component part corresponding to symbol.

4.19.3.8 ELECTRICAL SYMBOL INSTANCE

The occurrence of a symbol on a schematic.

```
*)
ENTITY electrical_symbol_instance;
  identification_value : NUMBER;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

identification_value: Identifier for this instance.

4.19.3.9 COMPONENT PART

A part which is a constituent part of the defined level.

```
*)
ENTITY component_part;
END_ENTITY;
(*

*)
RULE SYMBOL_COMPONENT;
  REPRESENTS_COMPONENT_PART           : OPTIONAL SYMBOL_INSTANCE;
  IS_REPRESENTED_BY_SYMBOL_INSTANCE   : OPTIONAL COMPONENT_PART;
END_RULE;
(*
```

PROPOSITIONS:

1. A **component part** may be represented by a **symbol instance.**

4.19.3.10 SYMBOL LINE

The occurrence of a line in a symbol.

```
*)
ENTITY symbol_line;
  direction : two_space_direction;
  location  : cartesian_two_coordinate;
  line_data : schematic_line;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

direction: Slope of LINE

location: Location of end point of line.

line_data: Pointer to (electrical schematic) LINE entity.

4.19.3.11 SCHEMATIC LINE

A narrow, elongated mark drawn or projected.

```
*)
ENTITY schematic_line;
  length : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

length: Length of line.

4.19.3.12 SYMBOL ARC

The occurrence of an arc in a symbol.

```
*)
ENTITY symbol_arc;
  location    : cartesian_two_coordinate;
  line_width  : REAL;
  arc_data    : schematic_circular_arc;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Location of arc.

line_width: Width of line representing arc.

arc_data: Pointer to schematic circular arc.

4.19.3.13 SCHEMATIC CIRCULAR ARC

A narrow, evenly curved mark drawn or projected.

```
*)
ENTITY schematic_circular_arc;
  radius        : REAL;
  start_angle   : REAL;
  stop_angle    : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

radius: Radius of ARC.

start_angle: Start angle for ARC.

stop_angle: End angle for ARC.

4.19.3.14 CONNECTION GEOMETRY

Connection geometry: Geometry which represents the logic of an electrical joining. It is a set of one or more 2-D lines or arcs between symbol connection places or intersection places.

```
*)
ENTITY connection_geometry
  SUPERTYPE OF (electrical_string XOR
                bus_string);
  start_place      : cartesian_two_coordinate;
  end_place        : cartesian_two_coordinate;
  line_weight      : REAL;
```

```
  connection_list : LIST [1:#] OF connection_place;
  line_list       : LIST [1:#] OF schematic_line;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

start_place: Start point of connection.

end_place: End point of connection.

line_weight: Weight of lines representing connection.

connection_list: List of connection points associated with connection.

line_list: List of lines forming connections.

```
*)
RULE NETWORK_CONNECTION_GEOMETRY;
  IS_REPRESENTED_BY_CONNECTION_GEOMETRY : NETWORK;
  REPRESENTS_NETWORK                    : CONNECTION_GEOMETRY;
WHERE
  SIZEOF (IS_REPRESENTED_BY_CONNECTION_GEOMETRY) = 1;
  SIZEOF (REPRESENTS_NETWORK) >= 1;
END_RULE;

RULE LINE_CONNECTION_GEOMETRY;
  IS_IN_CONNECTION_GEOMETRY : LIST [1:#] OF SCHEMATIC_LINE;
  IS_REPRESENTED_BY_LINES : CONNECTION_GEOMETRY;
END_RULE;
(*
```

PROPOSITIONS:

1. Connection geometry can consist of one or more lines.

4.19.3.15 CONNECTION PLACE

A 2-D location which represents the place where one or more connection strings may begin or end.

```
*)
ENTITY connection_place
  SUPERTYPE OF (intersection_place XOR
                symbol_connection_place);
  location : cartesian_two_coordinate;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: Location of connect point.

```
*)
RULE CONNECTION_GEOMETRY_CONNECTION_PLACE;
  IS_JOINED_BY_GEOMETRY   : CONNECTION_PLACE;
  JOINS_CONNECTION_PLACES : CONNECTION_GEOMETRY;
WHERE
  SIZEOF (IS_JOINED_BY_GEOMETRY) >= 1;
  SIZEOF (JOINS_CONNECTION_PLACES) >= 1;
END_RULE;
(*
```

PROPOSITIONS:

1. **Connection geometry** may contain one or more **connection places.**
2. A **connection place** is associated with one or more **connection geometrys.**

4.19.3.16 SYMBOL CONNECTION PLACE

The location on a **symbol** where a connection may occur.

```
*)
ENTITY symbol_connection_place
  SUBTYPE OF (connection_place);
  function_code    : STRING;
  pin_number       : INTEGER;
  input_or_output  : input_output_enumeration;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

function_code: Code for this connection point (pin).

pin_number: Number of this pin.

input_or_output: Flag showing input/output nature of pin.

4.19.3.17 INTERSECTION PLACE

A place where **connection lines** are considered to be connected.

```
*)
ENTITY intersection_place
  SUBTYPE OF (connection_place);
  bullet : two_space_shape_size;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

bullet: A small geometric symbol which may be located at an **intersection point** to indicate that two or more **connection lines** belong to the same **network.**

4.19.3.18 TWO SPACE SHAPE SIZE

This is a dummy entry to be used as a place-holder for the **bullet** attribute of **intersection place.**

```
*)
ENTITY two_space_shape_size;
END_ENTITY;
(*
```

4.19.3.19 NETWORK

A collection of places which are defined to be at the same potential (aka, Node).

```
*)
ENTITY network;
  net_name                  : STRING;
  representation_geometry   : LIST [1:#] OF connection_geometry;
  characteristics_list      : LIST [0:#] OF circuit_characteristics;
UNIQUE
  net_name;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

net_name: Signal name (single), bus name or trunk ID (bus).

representation_geometry: Geometry constituting the **network.**

characteristics_list: Electrical characteristics of **network.**

```
*)
RULE SYMBOL_NETWORK;
  SIGNAL_IS_CARRIED_ON : LIST [1:#] OF NETWORK;
  CARRIES_SIGNAL       : NETWORK;
END_RULE;
(*
```

PROPOSITIONS:

1. Net name must be UNIQUE.
2. One signal may be carried on one or more networks.

4.19.3.20 CIRCUIT CHARACTERISTICS

The analysis-defined activity of a network.

```
*)
ENTITY circuit_characteristics;
  sequence_number : INTEGER;
```

```
  characteristic  : algorithm;
UNIQUE
  sequence_number;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

sequence_number: Identifier of specific circuit characteristics.

characteristic: Details of characteristics.

PROPOSITIONS:

1. Sequence number must be UNIQUE.

4.19.3.21 ALGORITHM

This entity has been added to represent the attributes of the CIRCUIT CHARACTERISTICS IDEF1X entity.

```
*)
ENTITY algorithm;
END_ENTITY;
(*
```

4.19.3.22 CIRCUIT ANALYSIS

The modeling of the electrical functions on a network based on the characteristics of parts connected to that network and the electrical functions occurring on the networks connected to those parts.

```
*)
ENTITY circuit_analysis;
  name                  : STRING;
  version               : INTEGER;
  model                 : LIST [1:#] OF model_data;
  characteristics_list  : LIST [0:#] OF circuit_characteristics;
UNIQUE
  name;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: Identifier for this circuit analysis.

version: Version for this circuit analysis.

model: Model data used for this circuit analysis.

characteristics_list: Results of this circuit analysis.

```
*)
RULE ANALYSIS_NETWORK_CIRCUIT_CHARACTERISTICS;
  SPECIFIC_NETWORK                                    : NETWORK;
  SPECIFIES_CHARACTERISTICS_FOR_NETWORK_AND_ANALYSIS :
    CIRCUIT_CHARACTERISTICS
  DEFINES_CHARACTERISTICS_IN_TERMS_OF_NETWORK         :
    CIRCUIT_ANALYSIS;
END_RULE;
(*
```

PROPOSITIONS:

1. Name must be UNIQUE.

4.19.3.23 MODEL LIBRARY

The reference index for part performance parameters as required for a using circuit analysis process.

```
*)
ENTITY model_library;
  name       : STRING;
  model_list : LIST [1:#] OF model_data;
UNIQUE
  name;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

name: Library name.

model_list: Contents of library.

PROPOSITIONS:

1. Name must be UNIQUE.

4.19.3.24 MODEL DATA

The parameters for a given part required by an analysis package to simulate the function of that part.

```
*)
ENTITY model_data;
  model_identification : STRING;
  stimulus_value       : REAL;
  units                : undefined;
  transfer_function    : algorithm;
  part                 : component_part;
UNIQUE
  model_identification;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

model_identification: Identifier of model data.

stimulus_value: Value of electrical stimulus for this model data.

units:

transfer_function:

part: Part to which this model refers.

```
*)
RULE ANALYSIS_MODEL;
  PRODUCT_DATA_FOR_CHARACTERISTICS_IN_SPECIFIC_ANALYSIS          :
    MODEL_DATA;
  GIVES_RESULTS_IN_SPECIFIC_NETWORK_FOR_SPECIFIC_ANALYSIS_MODEL :
    MODEL;
END_RULE;
(*

*)
RULE MODEL_DATA_COMPONENT_PART;
  MODELS_COMPONENT_PART : OPTIONAL MODEL_DATA;
  HAS_DEFINING_MODEL    : LIST [0:#] OF COMPONENT_PART;
END_RULE;
(*
```

PROPOSITIONS:

1. Model identification must be UNIQUE.
2. A component part may be modeled by model data.

4.19.3.25 BUS STRING

A piecewise, continuous form of a 2-D line (aka, "string") which represents a collection of connection lines.

```
*)
ENTITY bus_string
  SUBTYPE OF (connection_geometry);
  bus_name    : STRING;
  width       : INTEGER;
  string_list : LIST [2:#] OF electrical_string;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

bus_name: Name of bus.

width: Number of electrical strings in bus.

string_list: Electrical strings constituting bus.

4.19.3.26 ELECTRICAL STRING

```
*)
ENTITY electrical_string
  SUBTYPE OF (connection_geometry);
  s_inflection : LIST [0:#] OF cartesian_two_coordinate;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

s_inflections: The places where the unit vector or segmented line (see STRING) takes on a new value.

4.19.4 Electrical Schematics Classification Structure

The following indented listing provides the classification structure for the Electrical Schematic entities.

```
ALGORITHM
CIRCUIT ANALYSIS
CIRCUIT CHARACTERISTICS
COMPONENT PART
CONNECTION GEOMETRY
  BUS STRING
  ELECTRICAL STRING
CONNECTION PLACE
  INTERSECTION PLACE
  SYMBOL CONNECTION PLACE
ELECTRICAL SYMBOL INSTANCE
LIBRARY NAME STRUCTURE
MODEL DATA
MODEL LIBRARY
NETWORK
PRODUCT ASSEMBLY DEFINITION
SCHEMATIC
SCHEMATIC CIRCULAR ARC
SCHEMATIC LINE
SYMBOL
SYMBOL ARC
SYMBOL INSTANCE
SYMBOL LIBRARY
SYMBOL LINE
TWO SPACE SHAPE SIZE

*)
END_SCHEMA; -- end ELECTRICAL SCHEMATIC schema
(*
```

4.19.5 Layered Electrical Product

```
*)
SCHEMA ipim_lep_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_geometry_schema,
         ipim_topology_schema,
         ipim_design_shape_schema,
         ipim_material_schema);
(*
```

4.19.5.1 Scope

This model is intended to define the data about the as-designed electrical product which is expressible in terms of topological layers. Included are voids, boundaries and information contained on, or joining layers. Products which belong to this scope include integrated circuits, hybrid microelectronic assemblies, printed wiring (circuits and boards), flex harnesses and microwave strip line. In addition, entities are included which provide for the manufacture of printed wiring boards on a panel together with test coupons. The model as it relates to this scope is discussed in the model overview.

4.19.5.2 Purpose

The model was constructed by successive meetings between May and December 1987 of the Cal Poly team to achieve a common logical view of the data needed to drive factory automation (CIM) for the products mentioned above. In turn, the model is intended to become a contribution to the integrated logical layer.

4.19.5.3 Model Overview

The CPTTX model is built on three levels of abstraction. Taking this approach gives the model a stable (and standard) foundation at the highest level of abstraction, generality at the middle level, and flexibility of expression at the lowest level of abstraction. This approach allows us to build the model assuring that each increment is of high quality. If the team cannot come to a consensus on any increment of the model, that increment is low in quality and the rest of the model will probably suffer until consensus is achieved. As a team we learned that we could not tackle the entire model at all levels at once and achieve a quality model. This approach gives structure to the overall model building process. First one asks "do we understand the fundamental nature of the object we are modeling?" Once that fundamental structure is understood (independent of technology) in abstract terms, we hang more user-friendly terminology on the abstractions, or leaves on the tree, to make the model assessable to persons not interested in the abstract foundations of the model. Once the model has been enriched with user friendly terminology, it is adapted to a specific type of technology, for example, large-scale integration (LSI), hybrid micro-circuit assembly (HMA) or printed wiring board (PWB). In this phase of our effort, the model was adapted to PWB technology; however, hopefully, the model is general enough to be adapted to LSIs and HMAs as well.

Level 1: Fundamental Level (most abstract)

Level 2: Layered Electrical Product

Level 3: Technology-Specific (least abstract)

4.19.5.3.1 Level 1 — Topology

A Layered electrical product (LEP) is abstractly represented as an assembly of topological faces. Each layer of a LEP is modeled as a distinct fact which bounds the material of that layer. This layer face contains sub faces which model the existence of only one material but the material can be broken into small pieces. The subfaces are in the same topological plane as the parent face, or, expressed in geometric terms, all material elements in a layer are co-planar. The surfaces themselves contain subfaces which are regions in which layer to layer joining occurs. Joining is a very general term for physical connectivity between layers. These connection regions are joined two at a time. Two regions joined together are referred to as *join* in this model. Joins are further abstracted as topological edges. The vertices of these edges are abstractions for the joined regions. The edge abstracts out all details of the process of joining while allowing us to state that a joining has occurred. In general the use of topology allows us to model the connectivity and boundedness of the LEP assembly without having to refer to geometry, specific technology, or manufacturing processes. The use of topology allowed the CPTTX group to come to an agreement on the basic assembly nature of an LEP without being involved in technology specific discussions.

4.19.5.3.2 Level 2 — Layered Electrical Product

This level of the model describes the LEP in non topological terms but relates each term to entities in Level 1. Basically this level of the model states that a layered electrical product has one or more (sequential) layers — each layer containing one or more layer elements of the same material. Layer elements are further divided into basic shapes such as point shapes (pads), graph shapes (traces), and area shapes (ground planes). A layer element is created by pasting these basic shapes together. These basic shapes in turn relate directly to topological entities — a point shape corresponds to a simple face, a graph shape corresponds to a loop or a path, and an area shape corresponds to an arbitrarily complex face. By doing this, we allow the layered electronic product expert to talk in terms of shapes instead of edges, vertices, faces, etc. In a sense these shapes are a type of feature which can be used as a handle far for more abstract notions. This is what features should do. This is the user friendly handle referred to earlier. Connections between layers are done with joins. Inter-layer joins physically connect together element regions that are on different layers. Intra-layer joins paste together the basic shapes on a layer. A component lead join physically ties a component lead to some region on a layer element (in our model the assumption is that the connection occurs on the element region closest to the body of the component). For convenience we also define logical regions on a layer for general purpose usage such as defining the region in which text is to be located. A logical region could define any region of interest, for example, a keep out region. It does not have to be bound material. In this layer of the model, layer elements are associated with electrical signals. However, the defined electrical logical link is an entity on another model. The component is not a principal part of this model but it is referenced. Material is also referenced.

4.19.5.3.3 Level 3 — Printed Wiring Board

This level of the model relates terminology used by a particular technology (e.g. for manufacturing planning) to the other two levels of the model. In our model that aspect is the particular way that joins are manufactured — as passages with deposition. Passages relate to voids (loops) in layer elements. Deposition (the plating inside the passage) is related to a set of joins that it implements. Vias are kinds of plated passages that join layer elements but do not accommodate component leads. Component

leads themselves may provide the mechanism for joining. The important thing about this part of the model is that it describes objects that are closely related to the process of manufacturing (physically realizing) the concepts described in the other parts of the model. Level 3 can be "unplugged" and another model put in its place. This level of the model allows for flexibility for different viewpoints from different enterprises and different product types (HMA, LSI). It is not the intention that our level 3 is the ultimate answer; it is a typical perception of a PWB. Other level 3's could easily be wired in by relating the level 3 terms with their counterparts in level 1 or 2. In a sense, there is a kind of poor man's mapping or correspondence process which is occurring in our model. Also described in level 3 is the manufacturing object "panel." (A panel is defined as "a rectangular or square base material of pre-determined size intended for, or containing one or more, layered electrical products, and, when required, one or more test coupons." A panel illustrates the nesting relationship between an individual LEP and a panel with its panel layers, test coupons and panel detail. This represents a process planning view of a LEP. As an example, if someone else were to develop a level 3 application, they might want to refer to a silicon wafer instead of a panel, and a new set of child entities and level 2 relationships.

4.19.5.4 LEP TYPE Definitions

4.19.5.4.1 LAYER NOMENCLATURE ENUMERATION

```
*)
TYPE layer_nomenclature_enumeration = ENUMERATION OF
     (ground_plane,
      power_plane,
      silkscreen,
      solder_mask,
      diffusion);
END_TYPE;
(*
```

4.19.5.4.2 LEP PRODUCT TYPE ENUMERATION

```
*)
TYPE lep_product_type_enumeration = ENUMERATION OF
     (rigid,
      flex_rigid,
      hybrid,
      integrated_circuit,
      flex_harness);
END_TYPE;
(*
```

4.19.5.4.3 PASSAGE USES ENUMERATION

```
*)
TYPE passage_uses_enumeration = ENUMERATION OF
     (fixturing,
      registration,
      mounting,
```

```
      clearance);
END_TYPE;
(*
```

4.19.6 Level 1 Model

The Level 1 model consists of topological entities (**vertex, edge, path, loop, face** and **subface**) and a restricted set of geometry entities (**point, line, circle** and (**plane**) **surface**). These entities do not appear here as their definitions are already available elsewhere in the Integrated Product Information Model (IPIM).

4.19.7 Level 2 Model

This model is couched in terms more appropriate to the practicing design engineer than to a topologist.

4.19.7.1 LAYERED ELECTRICAL PRODUCT

A **layered electrical product** is a physically defined product that has two or more **layers** stacked in sequential order and whose primary functionality is electrical.

```
*)
ENTITY layered_electrical_product
  SUBTYPE OF (design_model);
  material_bound_abstraction    : face;
  product_type                  : lep_product_type_enumeration;
  outline                       : undefined;
  clearance_limits              : undefined;
  design_rules                  : undefined;
  layer_sequence                : LIST [2 : #] OF layer;
  joins                         : SET [0 : #] OF assembly_join;
WHERE
  planar(material_bound_abstraction);
  embedded(layer_sequence, material_bound_abstraction);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

material_bound_abstraction: The overall boundaries of the product. Material may only exist in this region.

layer_sequence: The ordered sequence of the layers forming the LEP.

joins: The set of joins between the layers of the product and between the product and external items.

PROPOSITIONS:

1. The geometry associated with the **face** must be planar.
2. The individual layers must all be contained within the overall bound.

4.19.7.2 LAYER -

A stratum of one uniformly constituted material the thickness of which may vary on one layer. A layer contains elements constituted of the same material. Layer elements may completely consume the region of the layer. A layer may contain voids.

```
*)
ENTITY layer;
  layer_boundary_abstraction : select_face_or_subface;
  layer_nomenclature         : layer_nomenclature_enumeration;
  layer_outline              : undefined;
  layer_thickness            : size;
  material_specification     : lep_material;
  layer_elements             : SET [1 : ?] OF layer_element;
  layer_subregions           : SET [0 : ?] OF logical_subregion;
  joins                      : SET [0 : ?] OF intralayer_join;
WHERE
  embedded(layer_elements, layer_boundary_abstraction);
  embedded(layer_subregions, layer_boundary_abstraction);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

layer_boundary_abstraction: The domain of the layer. Material is limited to this domain.

layer_nomenclature: The purpose of the layer.

layer_thickness: The thickness of the material comprising the layer.

material_specification: The specification of the material comprising the layer.

layer_elements: The set of regions embedded in the layer.

layer_subregions: The set of logical subregions embedded in the layer.

joins: The set of joins within the layer.

PROPOSITIONS:

1. All layer elements must be contained geometrically within the layer.
2. All logical subregions must be contained geometrically within the layer.
3. Layer elements within a layer are disjoint (Need WHERE clause).

4.19.7.3 LAYER ELEMENT

A continuous topological region of dimensionality 2 that is contained within one and only one layer. A layer element may have fully contained subregions. An electrical signal may be associated with a layer element.

```
*)
ENTITY layer_element;
  is_described_by : SET [1 : #] OF lep_shape;
  subregions      : SET [0 : #] OF layer_element_subregion;
  signal          : OPTIONAL defined_electrical_logical_link;
WHERE
  embedded(subregions, layer_element);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

is_described_by: The set of shapes defining the **layer element** region

subregions: The set of subregions within the **layer element.**

signal: An electrical signal that is associated with the **layer element.**

PROPOSITIONS:

1. The **subregions** must be geometrically contained within the **layer element.**

4.19.7.4 LAYER ELEMENT SUBREGION

A subregion which is completely contained within a **layer element.** The **layer element subregion** may share zero or more boundaries of the **layer element** or of other **layer element subregions.**

```
*)
ENTITY layer_element_subregion
  SUPERTYPE OF (element_join_subregion);
  abstracted_as : OPTIONAL select_face_or_subface;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

abstracted_as: The definition of the domain of the region.

4.19.7.5 ELEMENT JOIN SUBREGION

A **layer element subregion** that exists for the purpose of connecting **layer elements** or connecting a layer element and a component. An **element join subregion** can be connected to at most two interlayer joins.

```
*)
ENTITY element_join_subregion
  SUBTYPE OF (layer_element_subregion);
  location    : vertex;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location: An abstraction of the location of the join region

4.19.7.6 LEP JOIN

A connection between two disjoint items.
The items that are connected are abstracted as vertices.

```
*)
ENTITY lep_join
  SUPERTYPE OF (assembly_join XOR
                intralayer_join);
  location_abstraction : edge;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

location_abstraction: The location of the join.

4.19.7.7 ASSEMBLY JOIN

```
*)
ENTITY assembly_join
  SUPERTYPE OF (layer_to_layer_join XOR
                component_lead_join)
  SUBTYPE OF (lep_join);
END_ENTITY;
(*
```

4.19.7.8 LAYER TO LAYER JOIN

A connection between exactly two element join subregions that exist on two adjacent layers.

```
*)
ENTITY layer_to_layer_join
  SUBTYPE OF (assembly_join);
END_ENTITY;
(*
```

4.19.7.9 COMPONENT LEAD JOIN

A connection between exactly one element join subregion and a component lead.

```
*)
ENTITY component_lead_join
  SUBTYPE OF (assembly_join);
END_ENTITY;
(*
```

4.19.7.10 INTRALAYER JOIN

A connection between exactly two element join subregions that exist on the same layer element.

```
*)
ENTITY intralayer_join
  SUBTYPE OF (lep_join);
END_ENTITY;
(*
```

4.19.7.11 LOGICAL SUBREGION

An artificial area which is superimposed anywhere on a layer and does not have to contain any material. A logical subregion may be partly or fully contained within another logical subregion. Special types of logical subregion include: information subregion and restricted subregion.

```
*)
ENTITY logical_subregion
  SUPERTYPE OF (restricted_subregion XOR
                information_subregion);
  boundary         : loop;
  is_described_by : SET [1 : ?] OF lep_shape;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

boundary: A loop defining the boundary of the region.

is_described_by: The shape of the region.

4.19.7.12 RESTRICTED SUBREGION

A logical subregion which either fully or partially defines a keep-out boundary.

```
*)
ENTITY restricted_subregion
  SUBTYPE OF (logical_subregion);
END_ENTITY;
(*
```

4.19.7.13 INFORMATION SUBREGION

A logical subregion that contains information that is represented graphically (i.e Text, Icons bar codes etc.)

```
*)
ENTITY information_subregion
  SUBTYPE OF (logical_subregion);
```

```
  iconic_information   : SET [0 : #] OF icon_placement;
  textual_information : SET [0 : #] OF text_placement;
WHERE
  NOT (iconic_information = NULL AND
   textual_information = NULL);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

iconic_information: Icons to be placed in the region.

textual_information: Text to be placed in the region.

PROPOSITIONS:

1. At least one of the attributes must not be NULL.

4.19.7.14 ICON PLACEMENT

A located instance of an icon.

```
*)
ENTITY icon_placement;
  symbol    : icon;
  location : axis_placement;
END_ENTITY;
(*
```

4.19.7.15 ICON

A graphic (i.e company logo, symbols etc.)

```
*)
ENTITY icon;
  type_of_icon : undefined;
END_ENTITY;
(*
```

4.19.7.16 TEXT PLACEMENT

An instance of text in a specific location.

```
*)
ENTITY text_placement;
  symbol    : text;
  location : axis_placement;
END_ENTITY;
(*
```

4.19.7.17 TEXT

Textual information represented graphically.

```
*)
ENTITY text;
  character_string : STRING;
  font             : UNDEFINED;
  height           : REAL;
  aspect_ratio     : REAL;
END_ENTITY;
(*
```

4.19.7.18 LEP SHAPE

A description that can be applied to either layer elements or logical subregions thereby providing a boundary definition.
If aspect is TRUE then the interior of the shape contains (layer) material and if ASPECT is FALSE then the interior does not contain (layer) material.

```
*)
ENTITY lep_shape
  SUPERTYPE OF (point_shape XOR
                graph_shape XOR
                area_shape);
  aspect : LOGICAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

aspect: Indication of material or not.

4.19.7.19 POINT SHAPE

A simple shape that is located by a single point.

```
*)
ENTITY point_shape
  SUPERTYPE OF (explicit_point_shape XOR
                implicit_point_shape)
  SUBTYPE OF (lep_shape);
  locating_vertex : vertex;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

locating_vertex : The (abstraction of the) location of the shape.

4.19.7.20 IMPLICIT POINT SHAPE

A simple **shape** that is located by a single **point**, for example, a pad which is implicitly defined by, for example, a position on a photo plotter aperture wheel.

```
*)
ENTITY implicit_point_shape
  SUBTYPE OF (point_shape);
  definition          : UNDEFINED;
  explicit_boundary : OPTIONAL face;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

definition: The reference to the definition of the shape.

explicit_boundary: An explicit definition of the shape boundary.

4.19.7.21 EXPLICIT POINT SHAPE

An explicit description defined by a **face**, applied to a **point shape**. It possess its own coordinate system that is independent from the product. A typical example would be a library part resident in a CAD system.

```
*)
ENTITY explicit_point_shape
  SUBTYPE OF (point_shape);
  boundary                  : face;
  location_and_orientation : axis_placement;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

boundary: The domain of the shape.

location_and_orientation: The local coordinate system of the shape.

4.19.7.22 GRAPH SHAPE

A simple linear **shape** that has a uniform width of which there
are two types: **open graph shape** and **closed graph shape.**
Only the "center line" of the shape is represented; the "interior" of the shape is symmetrically disposed about the center line and has a total width of **width.**

```
*)
ENTITY graph_shape
  SUPERTYPE OF (closed_graph_shape XOR
                open_graph_shape)
```

```
  SUBTYPE OF (lop_shape);
  width   : REAL;
WHERE
  width > 0;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

width: The width of the interior.

PROPOSITIONS:

1. Width must be greater than zero.

4.19.7.23 CLOSED GRAPH SHAPE

A graph shape that is defined by a loop.

```
*)
ENTITY closed_graph_shape
  SUBTYPE OF (graph_shape);
  center_line : loop;
WHERE
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

center_line: The center-line of the interior of the shape.

4.19.7.24 OPEN GRAPH SHAPE

A graph shape that is defined by a path.

```
*)
ENTITY open_graph_shape
  SUBTYPE OF (graph_shape);
  center_line : path;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

center_line: The center-line of the interior of the shape.

4.19.7.25 AREA SHAPE

A shape which has an arbitrary complex boundary that is defined explicitly by a face. The face may contain voids.

```
*)
ENTITY area_shape
  SUBTYPE OF (lep_shape);
  region : face;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

region: The definition of the interior of the shape.

4.19.8 Level 3 Model

This model is appropriate for the planning and manufacture of a Level 2 Model in the form of a Printed Wiring Board product. This model references the Level 2 Model.

4.19.8.1 PANEL

A rectangular or square base material of pre-determined size intended for, or containing one or more layered electrical products and, when required, one or more test coupons.

```
*)
ENTITY panel;
  panel_upper_right : cartesian_two_coordinate;
  panel_lower_left  : cartesian_two_coordinate;
  coupons           : SET [0 : #] OF test_coupon;
  perforations      : SET [0 : #] OF lep_passage;
  nests             : SET [0 : #] OF
                          panel_layered_electrical_product;
  components        : LIST [0 : #] OF panel_detail;
  made_from         : SET [1 : #] OF layer;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

panel_upper_right: The coordinates of the upper right end of the rectangle diagonal.

panel_lower_left: The coordinates of the lower left end of the rectangle diagonal.

coupons: The set of test coupons on the panel.

perforations: The set of perforations (through holes) in the panel.

nests: The set and location of LEPs on the panel.

components: The list of panel details (sub-assemblies) forming the panel.

made_from: The set of layers forming the panel.

4.19.8.2 PANEL DETAIL

The manufacturing identification of one or more layers. The outside layers may be processed to form a layer element, and lep passages may be processed through the panel detail. The lep passages may become blind or buried vias when the panel detail is assembled to other panel details.

```
*)
ENTITY panel_detail;
  definition : panel;
END_ENTITY;
(*
```

4.19.8.3 PANEL LAYERED ELECTRICAL PRODUCT

An occurrence of a layered electrical product in a manufacturing printed circuit or printed wiring board panel. Note that one or more products are nested in a window area of the panel.

```
*)
ENTITY panel_layered_electrical_product;
  product_offset_and_orientation : axis_placement;
  produces                       : layered_electrical_product;
END_ENTITY;
(*
```

4.19.8.4 TEST COUPON

A portion of the quality conformance test circuitry used for a specific acceptance test or group of related tests.

```
*)
ENTITY test_coupon;
  specification : UNDEFINED;
END_ENTITY;
(*
```

4.19.8.5 LEP PASSAGE

The implementation of a void in a panel detail normal to the plane of its layers. It is an implicit feature within the design topology and an explicit feature in the printed circuit or printed wiring board data. It may be drilled through the panel detail during some phase of detail processing depending on whether the element has been formed in order to be visually present or has been intended and formed to provide current conduction for plating purposes.

```
*)
ENTITY lep_passage
  SUPERTYPE OF (layer_passage XOR
                component_lead_passage XOR
                via XOR
                layer_to_layer_join_string);
  location       : axis_placement;
```

```
  usage          : OPTIONAL passage_uses_enumeration;
  diameter       : size;
  is_surfaced_by : LIST [0 : #] OF deposition;
  signal         : OPTIONAL def_elec_log_link;
END_ENTITY;
(*
```

4.19.8.6 DEPOSITION

The specification of a deposited material.

```
*)
ENTITY deposition;
  specification          : UNDEFINED;
  material_specification : lep_material;
  thickness              : REAL;
END_ENTITY;
(*
```

4.19.8.7 LAYER PASSAGE

The occurrence of a lep passage through a layer.

```
*)
ENTITY layer_passage
  SUBTYPE OF (lep_passage);
  boundary : loop;
END_ENTITY;
(*
```

4.19.8.8 COMPONENT LEAD PASSAGE

The occurrence of a lep passage intended for a component lead. The lep passage may be plated.

```
*)
ENTITY component_lead_passage
  SUBTYPE OF (lep_passage);
END_ENTITY;
(*
```

4.19.8.9 VIA

An electrical plated passage whose sole purpose is to electrically connect layer elements on two or more layers. It cannot be
associated with a component lead.

```
*)
ENTITY via
  SUBTYPE OF (lep_passage);
```

```
  via_number    : NUMBER;
  via_type      : UNDEFINED;
WHERE
  signal <> NULL;
  is_surfaced_by <> NULL;
END_ENTITY;
(*
```

PROPOSITIONS:

1. Signal must not be NULL.
2. The passage must be plated.

4.19.8.10 LAYER TO LAYER JOIN STRING

The occurrence of a lep passage as a means of achieving one or more layer to layer joins.

```
*)
ENTITY layer_to_layer_join_string
  SUBTYPE OF (lep_passage);
  join_path : path;
END_ENTITY;
(*
```

4.19.8.11 LAYER DEPOSITION

A layer whose layer elements are implemented as an additive process.

```
*)
ENTITY layer_deposition;
  is_based_on                : layer;
  deposition_specification : deposition;
END_ENTITY;
(*
```

4.19.9 Relationship With Other EXPRESS Models

This Section provides some new EXPRESS models that are required by the LEP model. These are not fully defined but appear as dotted boxes in the LEP IDEF model.

4.19.9.1 LEP MATERIAL

The specification of a material as used in the LEP model.
Lep material is the tangible substance out of which a product is made. A material is made of exactly one constituted (mixture of) chemical compound possessing consistent characteristics and properties.

```
*)
ENTITY lep_material;
  material_name              : STRING;
```

```
  material_specification   : UNDEFINED;
  electrical_conductivity  : REAL;
  thermal_conductivity     : REAL;
END_ENTITY;
(*
```

4.19.9.2 DEFINED ELECTRICAL LOGICAL LINK

This defines a LEP electrical signal.

```
*)
ENTITY defined_electrical_logical_link;
  signal_attributes : UNDEFINED;
END_ENTITY;
(*
```

4.19.9.3 LEP COMPONENT

A COMPONENT is an individual part or combination of parts that performs a specific design function(s).

A lep component is a type of product which is identified as being part of the next assembly to the subject of the LEP model. As such, it has two or more component leads. It can either be mounted on a layered electrical product in a subsequent assembly process or be implemented as a layer element of the product.

```
*)
ENTITY lep_component;
  component_description : STRING;
  specification         : UNDEFINED;
  height                : REAL;
  width                 : REAL;
  length                : REAL;
  leads                 : SET [2 : ?] OF component_lead;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

component_description: The nomenclature commonly recognized as an identifier of a component. Examples are capacitor, DIP etc.

specification: The specification of the component. For example a reference to a catalogue number or part number.

height: The vertical displacement of the top of the unit from an installation surface.

width: The installed horizontal width of the component body.

length: The installed horizontal length of the component body.

leads: The set of connections on the component.

4.19.9.4 COMPONENT LEAD

A connection "point" on a lep component. That is, an element of a **lep component** that serves as a mechanical and/or electrical connector in an assembly.

```
*)
ENTITY component_lead;
  lead_id          : INTEGER;
  lead_max_width   : REAL;
  signal           : OPTIONAL def_elec_log_link;
  abstracted_as    : OPTIONAL vertex;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

lead_id: The "pin number" of the lead.

lead_max_width: The diameter of a round lead or the diagonal of a rectangular lead.

signal: The electrical signal carried by the lead.

abstracted_as: An abstraction of the location of the lead.

4.19.10 Classification Scheme

The following indented list provides the classification scheme for the Layered Electrical Product Model.

```
COMPONENT LEAD
DEPOSITION
DEF ELEC LOG LINK
ICON
ICON PLACEMENT
LAYER
LAYER DEPOSITION
LAYER ELEMENT
LAYER ELEMENT SUBREGION
  ELEMENT JOIN SUBREGION
LAYERED ELECTRICAL PRODUCT
LEP COMPONENT
LEP JOIN
  ASSEMBLY JOIN
    COMPONENT LEAD JOIN
    LAYER TO LAYER JOIN
  INTRALAYER JOIN
LEP MATERIAL
LEP PASSAGE
  COMPONENT LEAD PASSAGE
  LAYER PASSAGE
  LAYER TO LAYER JOIN STRING
```

```
  VIA
LEP SHAPE
  AREA SHAPE
  GRAPH SHAPE
    CLOSED GRAPH SHAPE
    OPEN GRAPH SHAPE
  POINT SHAPE
    EXPLICIT POINT SHAPE
    IMPLICIT POINT SHAPE
LOGICAL SUBREGION
  INFORMATION SUBREGION
  RESTRICTED SUBREGION
PANEL
PANEL DETAIL
PANEL LAYERED ELECTRICAL PRODUCT
TEST COUPON
TEXT
TEXT PLACEMENT

*)
  END_SCHEMA;  -- end of LEP schema
END_SCHEMA; -- end of ELECTRICAL schema
(*
```

4.20 Analysis Applications

4.20.1 Introduction

There are numerous methods for analysing a continuum by discretising the continuum into regions, such as the Finite Element, Boundary Element and Finite Difference discretization and analysis methodologies.

```
*)
SCHEMA ipim_analysis_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_fem_schema);
(*
```

4.20.2 Finite Element Analysis Information Model

4.20.2.1 Introduction

```
*)
SCHEMA ipim_fem_schema;

  EXPORT EVERYTHING;

  ASSUME(ipim_shape_interface_schema,
         ipim_material_schema,
         ipim_pscm_schema,
         ipim_data_transfer_schema);
(*
```

4.20.2.2 FEM TYPE Definitions

4.20.2.2.1 ELEMENT ORDER

This is an enumeration of the types of FEM elements describing the mathematical order of an element.

```
*)
TYPE element_order = ENUMERATION OF
     (constant,
      linear,
      quadratic,
      cubic);
END_TYPE;
(*
```

4.20.2.2.2 ELEMENT SHAPE

An enumeration of the avialable FEM element shapes.

```
*)
TYPE element_shape = ENUMERATION OF
     (point,
      line,
      triangle,
      quadrilateral,
      tetrahedron,
      wedge,
      pyramid,
      hexahedron);
END_TYPE;
(*
```

4.20.2.2.3 ENUMERATION OF CONSTANT VARYING

```
*)
TYPE enumeration_of_constant_varying = ENUMERATION OF
      (constant,
       non_constant);
END_TYPE;
(*
```

4.20.2.3 FINITE ELEMENT MODEL

A **finite element model** (FEM) is a collection of finite elements, nodes, material properties, and geometric characteristics which is used to mathematically model the physical behavior of a thing.

Some means of imposing the following constraints needs to be added to this schema. This is currently under discussion with the Logical Layer Committee.

1. The dependence of the following entities on the entity **finite element model** for their existance;
2. the uniqueness of the following entities (or in some cases an attribute of the following entities) within an instance of a **finite element model**;
3. the cardinality of the 1 to many relationship between **finite element model** and the following entities i.e each of these entities may not belong to more than on "finite element analysis".

   The entities in question are:

   - **element**
   - **node**
   - **fem coordinate system**
   - **fem group**
   - **fem material property**
   - **geometric property**
   - **spring property**
   - **damper property**
   - **spring geometric property**
   - **damper geometric property**

- beam interval
- beam section

4. Coordinate system numbers must be unique within a FEM.
5. Element numbers must be unique within a FEM.
6. Node numbers must be unique within a FEM.
7. Geometric Property numbers must be unique within a FEM.
8. FEM Material Property numbers must be unique within a FEM.
9. FEM Group names must be unique within a FEM.

```
*)
ENTITY finite_element_model;
  fem_id                    : STRING;
  author                    : person_name;
  creating_software         : STRING;
  creation_date             : date;
  creation_time             : time;
  descriptor                : STRING;
  targeted_analysis_code    : STRING;
  units_ref                 : units;
  product_item_ref          : OPTIONAL SET [1:#] OF
                        product_item_version_functional_definition;
  master_coordinate_system_ref  : master_coordinate_system;
  derived_coordinate_system_ref : OPTIONAL SET [1:#] OF
                                        derived_coordinate_system;
  element_ref               : SET [1:#] OF element;
  node_ref                  : SET [1:#] OF node;
  geometric_property_ref    : SET [1:#] OF geometric_property;
  material_property_ref     : SET [1:#] OF fem_material_property;
  approval_ref              : OPTIONAL SET [1:#] OF approval;
  group                     : OPTIONAL SET [1:#] OF fem_group;
  spring_property_ref       : OPTIONAL SET [1:#] OF spring_property;
  damper_property_ref       : OPTIONAL SET [1:#] OF damper_property;
  spring_geometric_property_ref : OPTIONAL SET [1:#] OF
                                        spring_geometric_property;
  damper_geometric_property_ref : OPTIONAL SET [1:#] OF
                                        damper_geometric_property;
  beam_interval_ref         : OPTIONAL SET [1:#] OF beam_interval;
  beam_section_ref          : OPTIONAL SET [1:#] OF beam_section;
UNIQUE
  fem_id;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

fem_id: This attribute represents the unique application-defined identifier of a FEM.

author: This attribute contains the FEM creator's name. There can be only one author's name.

creating_software: This attribute contains the name of the preprocessor used to create the FEM. This is normally the name of the software used to create the FEM.

creation_date: The date the FEM was created.

creation_time: The time the FEM was created.

descriptor: Text that describes the FEM and provides supporting information supplied by the analyst.

targeted_analysis_code: This attribute contains the name of the target analysis code that the FEM was created for.

units_ref: The identifier of the fundamental units that will be used in the FEM. "Values" defined in another system of **units** (e.g **fem material property**) must be converted to this system.

product_item_ref: This attribute records the product item the FEM references.

master_coordinate_system_ref: The **master coordinate system** for the model.

derived_coordinate_system_ref: The set of **derived coordinate systems** used by the **elements** and **nodes** of the model.

element_ref: The set of **elements** comprising the model.

node_ref: The set of **nodes** that are used by the **elements** of the model.

geometric_property_ref: The set of **geometric properties** used by the **elements** of the model.

material_property_ref: The set of **fem material properties** used by the **elements** of the model.

approval_ref: The dates and approvers of the model.

group: The set of **fem groups** in the model.

spring_property_ref:

damper_property_ref:

spring_geometric_property_ref:

damper_geometric_property_ref:

beam_interval_ref:

beam_section_ref:

PROPOSITIONS:

1. **Fem id** must be UNIQUE.

4.20.2.4 ELEMENT

This entity contains data about a finite element. A finite element is the basic building block of a FEM. It defines a mathematical relationship between a FEM's nodes.

```
*)
ENTITY element;
  element_number                : INTEGER;
  geometric_property_coord_sys  : fem_coordinate_system;
  local_coordinate_system       : fem_coordinate_system;
  material_property_coord_sys   : fem_coordinate_system;
  element_purpose_descriptor    : STRING;
  element_order_number          : element_order;
  element_shape_number          : element_shape;
  node_list                     : ARRAY [1 : maxnodes] OF node;
  geometric_property_type       : enumeration_of_constant_varying;
  geometric_property_ref        : ARRAY [1 : geom_prop_list_size]
                                    OF geometric_property;
WHERE
  maxnodes = derive_maxnodes(element_order,
                                        element_shape);
  geom_prop_list_size =
     derive_geom_prop_list_size(geometric_property_type, maxnodes);
  test_geom_prop(element) = TRUE;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

element_number: An application-defined number used to identify a finite element from all others within a FEM.

geometric_property_coord_sys: The coordinate system used to define the directions of the geometric properties in the finite element.

local_coordinate_system: The coordinate system used to define the local directions of the finite element.

material_property_coord_sys: The coordinate system used to define the directions of the material properties in the finite element.

element_purpose_descriptor: Text describing the functional purpose of the finite element.

element_order_number: An enumerated value describing the mathematical order of the finite element.

element_shape_number: An enumerated value describing the generic shape of the finite element.

node_list: The list of nodes defining the location and shape of the element.

geometric_property_type: An enumerated value specifing whether the geometric properties are constant or varying within the element.

geometric_property_ref: The element's geometric properties.

PROPOSITIONS:

1. The size of **node list** is given by the result of the **derive maxnodes** function.
2. The size of **geometric property ref** is given by the result of the **derive geom prop list size** function.

4.20.2.5 NODE

A **node** is a place within a FEM. It is a point in space with attributes specific to a FEM.

Coordinate system type	Node direction		
	1	2	3
Rectangular	X	Y	Z
Cylindrical	R	Θ	Z
Spherical	R	Θ	Φ

Table 14: Node Directions in Different Coordinate Systems

The direction of nodal values depends on the type(s) of coordinate system(s) associated with the node as shown in Table 4.20.2.5.

```
*)
ENTITY node;
  node_number                          : INTEGER;
  coordinate_1                         : REAL;
  coordinate_2                         : REAL;
  coordinate_3                         : REAL;
  definition_coordinate_system_ref     : fem_coordinate_system;
  output_coordinate_system_ref         : fem_coordinate_system;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

node_number: An application-defined number used to identify a FEM **node** from all others within a FEM.

coordinate_1, coordinate_2, coordinate_3: These attributes contain the location components of a node in the node's defining coordinate system with respect to directions 1, 2, and 3, respectively. Depending on the coordinate system type, the node values are as given in Table 4.20.2.5.

definition_coordinate_system_ref: The coordinate system used to define the location of the FEM node.

output_coordinate_system_ref: The coordinate system used to define the output directions of the FEM result data.

4.20.2.6 FEM COORDINATE SYSTEM

A reference frame used to define locations of FEM **elements**, **nodes**, environments, and results in 2D or 3D space. This data entity contains information about the use (either master or definition) and type (rectangular, polar, etc...) of a coordinate system.

```
*)
ENTITY fem_coordinate_system
  SUPERTYPE OF (master_coordinate_system OR
                derived_coordinate_system);
  coordinate_system_number : INTEGER;
  system_type              : coordinate_system_type;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

coordinate_system_number: An application defined number used to identify a **fem coordinate system.**

system_type: The type (rectangular, cylindrical etc.) of the coordinate system.

4.20.2.7 MASTER COORDINATE SYSTEM

A FEM always has a **master coordinate system.** The origin of the **master coordinate system** is always at (0,0,0) and the axes are aligned in the primary directions. All other FEM definition coordinate systems are derived from the FEM's **master coordinate system.**

```
*)
ENTITY master_coordinate_system
  SUBTYPE OF (fem_coordinate_system);
END_ENTITY;
(*
```

4.20.2.8 DERIVED COORDINATE SYSTEM

This entity contains information about the coordinate system used to define the position and/or direction of some other data element. It is located, oriented and defined within a **master coordinate system.**

```
*)
ENTITY derived_coordinate_system
  SUBTYPE OF (fem_coordinate_system);
  transformation_matrix        : ARRAY [1:4] OF ARRAY [1:3] OF REAL;
  reference_coordinate_system : master_coordinate_system;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

transformation_matrix: A matrix to map from the **master coordinate system** space to the definition coordinate system space. This matrix is dimensioned 4x3.

reference_coordinate_system: The coordinate system in which the derived coordinate system is embedded.

4.20.2.9 FEM GROUP

This entity contains data about a set of FEM entities that, for convenience, can be referred to by referring to one grouped item. Groups of **elements** and **nodes** are currently defined and they may be arbitrarily combined. Thus groups (sets) of **nodes** and finite **elements** can be referenced by referring to a single **fem group** data entity.

```
*)
ENTITY fem_group;
  group_number : INTEGER;
  group_name   : OPTIONAL STRING;
  node_ref     : OPTIONAL SET [1:#] OF node;
  element_ref  : OPTIONAL SET [1:#] OF element;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

group_number: An application defined number used to identify a **fem group** from all others within a FEM.

group_name: Text that describes the **fem group**. For example, possible values of this attribute might be 'red', 'set 1', or 'wing'. It is what ever the analyst chooses as a label for group identification.

node_ref: The set of **nodes** in a group.

element_ref: The set of **elements** in the group.

4.20.2.10 GEOMETRIC PROPERTY

This entity contains data that describes the physical and geometrical characteristics of a finite element.

```
*)
ENTITY geometric_property
  SUPERTYPE OF (point_geometric_property XOR
                line_geometric_property XOR
                beam_geometric_property XOR
                membrane_shell_geometric_property XOR
                solid_geometric_property);
  geometric_property_id : INTEGER;
  property_text         : OPTIONAL STRING;
  integration_order     : OPTIONAL INTEGER;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

geometric_property_id: An application-defined number used to identify a geometric property from all others within a FEM.

property_text: This attribute supplies ancillary data about a geometric property. For example, a text string describing the geometric properties of a beam could be inserted into this entity. This human readable text string would enable the receiving analyst to determine the cross-sectional shape of an 'I' beam, as this data is not readily available from a beam's cross-sectional and second moments of area. This text may or may not be of value to the receiving analyst or computer.

integration_order: This attribute contains the integration order of a finite element.

4.20.2.11 POINT GEOMETRIC PROPERTY

This entity's data represents a FEM point geometric properties. A FEM point is a 0D reference datum.

```
*)
ENTITY point_geometric_property
  SUBTYPE OF (geometric_property);
  point_mass_half_6x6_matrix : ARRAY [1:21] OF REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

point_mass_half_6x6_matrix: A 6x6 mass matrix for a point finite element.

4.20.2.12 LINE GEOMETRIC PROPERTY

```
*)
ENTITY line_geometric_property
  SUBTYPE OF (geometric_property);
  spring_property_ref : OPTIONAL spring_geometric_property;
  damper_property_ref : OPTIONAL damper_geometric_property;
WHERE
  NOT ((spring_property_ref = NULL) AND
       (damper_property_ref = NULL));
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

spring_property_ref: Geometric spring property.

damper_property_ref: Geometric damper property.

PROPOSITIONS:

1. Spring property ref and damper property ref cannot both be NULL.

4.20.2.13 SPRING GEOMETRIC PROPERTY

This entity contains the geometric property values for a FEM spring element.

```
*)
ENTITY spring_geometric_property;
  property_list : LIST [1 : #] OF spring_property;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

property_list: A list of spring property.

4.20.2.14 SPRING PROPERTY

This entity defines a spring geometric property.

```
*)
ENTITY spring_property;
  dof_sequence_no_1            : INTEGER;
  dof_sequence_no_2            : INTEGER;
  spring_coefficient           : REAL;
  stress_recovery_coefficient  : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

dof_sequence_no_1, dof_sequence_no_2: These attributes contains the degree of freedom at which the FEM spring coefficient is defined. This degree of freedom is referenced to the local element coordinate system.

spring_coefficient: This attribute contains the value of the spring coefficient. This is the coefficient k in the spring equation

$$Force = k * displacement.$$

stress_recovery_coefficient: The value to be used in the stress recovery of a spring finite element. The spring stress recovery coefficient is k in the equation

$$Stress = k * force.$$

4.20.2.15 DAMPER GEOMETRIC PROPERTY

This entity contains the geometric property values for a FEM damper element.

```
*)
ENTITY damper_geometric_property;
  property_list : LIST [1 : #] OF damper_property;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

property_list: A list of damper property.

4.20.2.16 DAMPER PROPERTY

This entity contains a damper geometric property.

```
*)
ENTITY damper_property;
  dof_sequence_no_1   : INTEGER;
  dof_sequence_no_2   : INTEGER;
  damper_coefficient : REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

dof_sequence_no_1, dof_sequence_no_2: These attributes contain the degree of freedom which the FEM damper coefficient is applied to. This degree of freedom is referred to the local element coordinate system.

damper_coefficient: The value of the damping coefficient. This is the coefficient k in the damping equation

$$Force = k * velocity.$$

4.20.2.17 BEAM GEOMETRIC PROPERTY

This entity represents a beam finite element's geometric properties.

```
*)
ENTITY beam_geometric_property
  SUBTYPE OF (geometric_property);
  pin_array_1_2              : OPTIONAL ARRAY [1:2] OF
                                        ARRAY [1:6] OF INTEGER;
  offset_vector_1_2          : OPTIONAL ARRAY [1:2] OF
                                        coordinate_triple;
  warping_coefficients_1_2 : OPTIONAL ARRAY [1:2] OF REAL;
  beam_interval_ref          : LIST [1:#] OF beam_interval;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

pin_array_1_2: The degrees of freedom the analyst wishes to release in the finite element. This degree of freedom is referenced to the local element coordinate system.

offset_vector_1_2: Three coordinates, x, y, and z, referenced to the local element coordinate system, are used to define the amount by which the neutral axis of the beam is offset from its nodes.

warping_coefficients_1_2:

beam_interval_ref:

4.20.2.18 BEAM INTERVAL

```
*)
ENTITY beam_interval;
  beam_length_fraction_B : REAL;
  section_ref_A          : beam_section;
  section_ref_B          : beam_section;
  material_ref           : fem_material_property;
WHERE
  (0.0 <= beam_length_fraction_B <= 1.0);
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

beam_length_fraction_B: The value of this attribute is the position along a beam finite element at which the beam geometric properties are defined. The length fraction is defined as being 0.0 at the first node of the beam finite element and being 1.0 at the end opposite from the first node.

section_ref_A:

section_ref_B:

material_ref: The fem material property used by the beam finite element.

PROPOSITIONS:

1. The beam length fraction must be between 0.0 and 1.0.

4.20.2.19 BEAM SECTION

This entity contains the geometric property values that can vary along the length of a FEM beam element.

```
*)
ENTITY beam_section
  SUPERTYPE OF (beam_property_data XOR
                standard_beam_section);
  stress_recovery_coefficients : OPTIONAL LIST [1 : #] OF
                                          coordinate_pair;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

stress_recovery_coefficients: The values of this attribute contain the Y and Z coordinate of the point at which the stresses are to be calculated for output. These values are referenced to the local beam finite element coordinate system.

4.20.2.20 BEAM PROPERTY DATA

```
*)
ENTITY beam_property_data
  SUBTYPE OF (beam_section);
  cross_sectional_area          : REAL;
  second_moment_of_area_I11     : REAL;
  second_moment_of_area_I22     : REAL;
  second_moment_of_area_I12     : REAL;
  polar_moment                  : REAL;
  shear_area_K1                 : REAL;
  shear_area_K2                 : REAL;
  shear_relief_coefficient_S1 : OPTIONAL REAL;
  shear_relief_coefficient_S2 : OPTIONAL REAL;
  non_structural_mass         : OPTIONAL REAL;
  centroid_offset             : OPTIONAL coordinate_pair;
  shear_centre_offset         : OPTIONAL coordinate_pair;
  non_structural_mass_offset  : OPTIONAL coordinate_pair;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

cross_sectional_area: The value of this attribute is the cross-sectional area of a beam finite element.

second_moment_of_area_I11, second_moment_of_area_I22, second_moment_of_area_I12: The values of these three attributes contain the beam finite element's second moment of areas (I_{11}, I_{22}, and I_{12}). The directions are referenced to the local element coordinate system.

polar_moment: This attribute contains the beam finite element's polar moment of area.

shear_area_K1, shear_area_K2: The values of these two attributes contain the beam finite element's shear areas (K_1, K_2). The directions are referenced to the local element coordinate system.

shear_relief_coefficient_S1:

shear_relief_coefficient_S2:

non_structural_mass: The value of the non-structural mass of the beam finite element for the purpose of applying gravity loading conditions.

centroid_offset:

shear_centre_offset:

non_structural_mass_offset:

4.20.2.21 STANDARD BEAM SECTION

This entity refers to an external definition of the properties of a Beam Section. The data in the reference must provide sufficient data to (conceptually) create an instance of the **beam property data** entity.

```
*)
ENTITY standard_beam_section
  SUBTYPE OF (beam_section);
  property_data : external_reference;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

property_data: The reference to the external source of beam property data.

4.20.2.22 MEMBRANE SHELL GEOMETRIC PROPERTY

This entity represents a shell finite element's geometric properties. A shell finite element is a special type of 3D finite element. The membrane is a 2D special case.

```
*)
ENTITY membrane_shell_geometric_property
  SUBTYPE OF (geometric_property);
  membrane_material            : fem_material_property;
  thickness                    : REAL;
  nonstructural_mass           : REAL;
  bending_material             : OPTIONAL fem_material_property;
  shear_material               : OPTIONAL fem_material_property;
  membrane_bending_material    : OPTIONAL fem_material_property;
  second_moment_of_area        : OPTIONAL REAL;
  shear_thickness              : OPTIONAL REAL;
  z_offset                     : OPTIONAL REAL;
  stress_recovery_coefficient  : OPTIONAL SET OF REAL;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

membrane_material: Material property for a shell finite element that is used in the element's membrane characteristics.

thickness: This attribute contains the value of the shell finite element's thickness. It is measured perpendicular to the face of the shell finite element.

nonstructural_mass: The value of the non-structural mass of a shell finite element used to apply loading conditions, such as gravity.

bending_material: Material property for a shell finite element that is used in the element's membrane-bending characteristics.

shear_material: Material property for a shell finite element that is used in the element's shear characteristics.

membrane_bending_material: Material property for a shell finite element that is used in the element's bending characteristics.

second_moment_of_area: The value of the shell finite element's second moment of area.

shear_thickness: This attribute contains the value of the shell finite element's thickness for shear. It measured perpendicular to the face of the shell element.

z_offset: This attribute contains the value of the distance by which the shell finite element is offset from its nodes in the local element coordinate system's Z direction.

stress_recovery_coefficient: These are the values that the analyst wants used for stress recovery with respect to a shell finite element. The coefficient is a z offset from the element's mid-plane in the element's coordinate system.

4.20.2.23 SOLID GEOMETRIC PROPERTY

This entity represents a solid finite element's geometric properties. A solid finite element is a 3D finite element.

```
*)
ENTITY solid_geometric_property
  SUBTYPE OF (geometric_property);
  material : fem_material_property;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

material: The material used by the solid finite element.

4.20.2.24 FEM MATERIAL PROPERTY

This entity associates a user defined integer value with a material property.

```
*)
ENTITY fem_material_property;
  material_property_id : INTEGER;
  property_definition  : material_property;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

material_property_id: An integer value

property_definition: The material's properties

4.20.2.25 FEM FUNCTION Definitions

4.20.2.25.1 DERIVE GEOM PROP LIST SIZE

```
*)
FUNCTION derive_geom_prop_list_size
```

```
                                (attr1: enumeration_of_constant_varying;
                                 maxnodes: INTEGER): INTEGER;
  LOCAL
    result : INTEGER;
  END_LOCAL;
  IF attr1 = constant THEN
    result := 1;
  ELSE
    result := MAXNODES;
  END_IF;
  RETURN (result);
END_FUNCTION;
(*
```

4.20.2.25.2 DERIVE MAXNODES

This function is incomplete. It gives an idea of what is required but needs to be extended to include all possible combinations of element shape and element order

```
*)
FUNCTION derive_maxnodes(order: element_order; shape: element_shape)
                                                        : INTEGER;
 LOCAL
  maxnodes : INTEGER;
 END_LOCAL;
IF shape = point THEN
  maxnodes := 1;
END_IF;
IF (shape = line AND order < 2) THEN
  maxnodes := 2;
END_IF;
IF (shape = line AND order > 1) THEN
  maxnodes := order + 1;
END_IF;
IF (shape = triangle AND order < 2) THEN
  maxnodes := 3;
END_IF;
IF (shape = triangle AND order = 2) THEN
  maxnodes := 7;
END_IF;
RETURN (maxnodes);
END_FUNCTION;
(*
```

4.20.2.25.3 TEST GEOM PROP

```
*)
FUNCTION test_geom_prop(param : element) : LOGICAL;
  LOCAL
```

```
    result : LOGICAL;
    max_nodes      : INTEGER;
    i              : INTEGER;
  END_LOCAL;
  result := TRUE;
  IF param.geometric_property_type = not_constant THEN
    max_nodes := derive_maxnodes(param.element_order,
                                 param.element_shape);
    REPEAT i := 1 TO max_nodes WHILE result = TRUE;
      IF param.node_list[i] <> NULL THEN
        IF param.geometric_property_ref[i] = NULL THEN
          result := FALSE;
        END_IF;
      END_IF;
    END_REPEAT;
  END_IF;
  RETURN(result);
END_FUNCTION;
(*
```

4.20.2.26 FEM Classification Structure

The following indented listing provides the classification structure for the FEM Model.

```
BEAM INTERVAL
BEAM SECTION
  BEAM PROPERTY DATA
  STANDARD BEAM SECTION
DAMPER GEOMETRIC PROPERTY
DAMPER PROPERTY
ELEMENT
FEM COORDINATE SYSTEM
  MASTER COORDINATE SYSTEM
  DERIVED COORDINATE SYSTEM
FEM ELEMENT SHAPE LINK
  ELEMENT D0 SHAPE LINK
  ELEMENT D1 SHAPE LINK
  ELEMENT D2 SHAPE LINK
  ELEMENT D3 SHAPE LINK
FEM GROUP
FEM MATERIAL PROPERTY
FEM NODE SHAPE LINK
  NODE D0 SHAPE LINK
  NODE D1 SHAPE LINK
  NODE D2 SHAPE LINK
  NODE D3 SHAPE LINK
FINITE ELEMENT MODEL
GEOMETRIC PROPERTY
  BEAM GEOMETRIC PROPERTY
```

```
  LINE GEOMETRIC PROPERTY
  MEMBRANE SHELL GEOMETRIC PROPERTY
  POINT GEOMETRIC PROPERTY
  SOLID GEOMETRIC PROPERTY
NODE
SPRING GEOMETRIC PROPERTY
SPRING PROPERTY

*)
  END_SCHEMA; -- end of FEM schema
END_SCHEMA; -- end of ANALYSIS schema
(*
```

4.21 Data Transfer Applications

4.21.1 Introduction

```
*)
SCHEMA ipim_data_transfer_schema;

  EXPORT EVERYTHING;

(*
```

At this time, two capabilities are being added to the existing capabilities of the physical file structure. These are an ability to establish an index table of entity labels, and the ability to externally reference information not contained in a given exchange file.

4.21.2 Defined Types

4.21.2.1 BINDING

When references are made to entities that are external to a given exchange file, the binding attribute is used to determine the change control relationship between the referencing and referenced entities. If the value is **current version at time of postprocess**, then the referencing entity will obtain the current of the referenced item only once and then only at the time of post processing. If the value is **current version at all times**, then the referencing entity must be notified or in some fashion updated each time that the referenced item is modified.

An example of how this might be implemented when the value is **current version at time of post-process** is: the translation software retrieves the referenced item and merges a copy of that item into the current model and then does not maintain any database connection between the "copied" item and the original information.

An example of an implementation when the value is **current version at all times** is: the translation software establishes a database connection between the referencing entity and the referenced item without retrieving or merging in a copy of the referenced item at all.

```
*)
TYPE binding = ENUMERATION OF
     (current_version_at_all_times,
      current_version_at_time_of_postprocess);
END_TYPE;
(*
```

4.21.2.2 ALIASABLE ENTITY

Aliasiable entities are those entities that could sensibly be given an alternate name or entity label. It is possible that this will extend to all entities in the schema.

```
*)
TYPE aliasable_entity = GENERIC;
END_TYPE;
(*
```

4.21.3 Entities

4.21.3.1 FOR EXPORT

The for export entity identifies entities in an exchange file that are candidates to be externally referenced at some point in the future. This is done to alert translation software so that the ability to establish database links to "exported" entities is maintained.

```
*)
ENTITY for_export;
  export_entities : LIST [1 : #] OF index_entry;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

export_entities: A list of index entry entities that associate character string aliases to entities. It is through this association that entities are given their "logical" names which will be used later to externally reference them.

PROPOSITIONS:

4.21.3.2 INDEX ENTRY

The index entry associates a character string alias with an entity.

```
*)
ENTITY index_entry;
  alias       : STRING;
  step_entity : aliasable_entity;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

alias: The character string "alternate" name for an entity.

step_entity: The entity to which the alias is attached.

PROPOSITIONS:

4.21.3.3 INDEX

The index entity can be thought of as an index table of entities and their character string aliases.

```
*)
ENTITY index;
  table_entries : INTERNAL LIST [1 : #] OF index_entry;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

table_entries: The list of entries in the index table.

PROPOSITIONS:

4.21.3.4 EXTERNAL REFERENCE

A reference to information outside the current exchange.

There are no restrictions on the type of data that may be externally referenced. It should be stressed that the data being referenced is not necessarily graphical and consequently any needs to transform the data are not the responsibility of the external reference entity. Any requirements to manipulate the referenced data are the responsibility of the entity that is using the external reference entity.

```
*)
ENTITY external_reference
  SUPERTYPE OF (computer_external_reference XOR
                human_external_reference);
END_ENTITY;
(*
```

4.21.3.5 COMPUTER EXTERNAL REFERENCE

The computer external reference entity is used to refer to data that is physically external to the exchange file being processed. The external reference should be thought of as an extension of the basic entity referencing capability of the physical file structure.

```
*)
ENTITY computer_external_reference
  SUBTYPE OF (external_reference);
  version            : binding;
  referred_item      : STRING;
  library_reference  : OPTIONAL library;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

version: Indication of the change control relationship between the referencing entity and the referenced item.

referred_item: The logical name for the externally referenced item. This may only coincidentally be a valid literal name for the receiving system. If the item being referenced was translated previously within an exchange file, then the contents of this string will exactly match one of the string contents in the previously sent export entity.

library_reference: A list of libraries from which the receiving system should resolve the external reference, in the list order of library names. If this attribute is not present, then the contents of the string in the referred item must be sufficient to resolve the reference.

PROPOSITIONS:

4.21.3.6 LIBRARY

The library entity is used to establish a list of areas in which to search to resolve external references. These areas can be equally thought of as part libraries, or directories, or even databases. The contents of the library entity contribute to the creation of a logical name through which the external reference

is resolved. It is not necessary to have a library entity in order to have an external reference. It is intended to be used when a number of externally referenced items all reside in the same library and the user does not wish to repeat the library qualification in each external reference. The type of data contained within a library is unrestricted.

```
*)
ENTITY library;
  library_names : LIST [1 : #] OF STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

library_names: A list of libraries that are to be searched in order to resolve an external reference. The values are to be thought of as logical names for the libraries.

PROPOSITIONS:

4.21.3.7 HUMAN EXTERNAL REFERENCE

This is a first pass at defining the information required for referencing something that is defined externally to the standard schema. It is intended for human rather than computer consumption. Most often the reference will be to information that is contained in some paper document and which therefor requires human searching and interpretation.

```
*)
ENTITY human_external_reference
  SUBTYPE OF (external_reference);
  item_description : STRING;
  title            : OPTIONAL STRING;
  address          : OPTIONAL STRING;
  authors          : OPTIONAL LIST [1 : #] OF person_name;
  publisher        : OPTIONAL STRING;
  publication_date : OPTIONAL date;
  how_published    : OPTIONAL STRING;
  how_to_use       : OPTIONAL STRING;
  notes            : OPTIONAL STRING;
END_ENTITY;
(*
```

ATTRIBUTE DEFINITIONS:

item_description: A description of the item being referenced (e.g a part number). All the other attributes are only to enable this to be located and used.

title: The title of the document or other source which contains the definition of item description.

address: Where a copy of the definition is obtainable

authors: The originator(s) of the definition.

publisher: Who published/released the definition.

publication_date: When the definition was released.

how_published: Information about the form of publication (eg Paper, Computer file or tape, Database etc.)

how_to_use: Information about how to use the definition. For example, if the reference is to a standard parameterised part, the values of the parameters.

notes: Any other information to assist in locating the reference.

4.21.4 Data Transfer Classification Scheme

The following indented list shows the classification scheme for the Data Transfer application.

```
EXTERNAL REFERENCE
  COMPUTER EXTERNAL REFERENCE
  HUMAN EXTERNAL REFERENCE
FOR EXPORT
INDEX
INDEX ENTRY
LIBRARY

*)
END_SCHEMA; -- end of DATA TRANSFER schema
(*
```

4.22 IPIM Schema End

This ends the IPIM schemas.

```
*)
  END_SCHEMA; -- end APPLICATIONS schema
END_SCHEMA; -- end IPIM schema
(*
```

Title: Test Methods - Conformance

Owner: James Fleming

Date: 31 October 1988

Corresponding ISO Document Number: N285

ISO TC184/SC4/WG1 ANNEX A (Normative) 31 October 1988

CRITERIA FOR CONFORMANCE

DOCUMENT NUMBER: Clause 5 **VERSION:** 1.0

TITLE: Criteria for Conformance to the Standard

ABSTRACT:

This document gives a definition of the minimum capability which must be implemented in software translators which claim to read in or write out the physical file format of this standard.

KEY WORDS:

Conformance, Implementation Criteria, Testing

DATE: 31 October 1988

OWNER: James Fleming

PHONE NUMBER: (812) 377-3338

FAX NUMBER:

ISO REPRESENTATIVE: Bradford Smith

STATUS: Working

Clause 5: Test Methods

Quality software implementations of this standard must exist before digital product models can be accurately and completely represented or exchanged. This section will describe clearly defined procedures designed to support the testing, validation and certification of software translators which claim to implement the physical file format defined in Annex B. As such, these procedures define what is expected of any software implementation and form the basis for developing test cases, software testing tools and criteria for measuring success.

Conformance Criteria

A basic concept is that of conformance. This section defines the minimum conformance criteria for any implementation of a physical file format. Separate criteria are defined for preprocessor translators which generate the physical file and postprocessor translators which read the physical file. Four aspects of conformance are addressed for these translators:

Syntax
Data structure and data consistency
Degradation of entity data
Completeness of the entity set supported

The syntax criteria is concerned with the correct form of the physical file. The structure requirement addresses the preservation of correct relationships among the entities. Data consistency is concerned with processing entity data that are internally consistent. The criteria on degradation of entity data deals with preserving all of the data which describe an entity. Finally, completeness of the entity set is concerned with making known which translations are, and are not, supported by the software implementation. It is the intent of this section that any translator claiming conformance to this standard shall adhere to all four of these criteria.

Conformance Criteria for Preprocessors.

A preprocessor which claims conformance to this standard shall satisfy the following criteria.

1. A conforming preprocessor shall create syntactically correct files in accordance with Annex B. This requirement includes the syntax of the overall file as well as the syntax of each entity in the file.

2. A conforming preprocessor shall create files that are structurally correct and consistent. This requirement includes the consistent specification of data within each entity.

3. A conforming preprocessor shall map the entities of the native system into the form of this standard in such a way that all information present in the native database is mapped into either the pre-defined entities of Clause 4 or the pre-defined entities plus implementor-defined entities. Excluded from this requirement are those data which are intrinsic to the structure of the native system's database manager and which do not add meaningful information to the definition of the product model.

4. A conforming preprocessor shall be accompanied by a publicly published list of all entities and data structures in the native database along with the corresponding entity, or entities, of this standard to which they are mapped. An unsupported mapping will appear in this list as a native entity or data structure for which there is no corresponding entry for the standard entity, or entities.

Conformance Criteria for Postprocessors.

A postprocessor which claims conformance to this standard shall satisfy the following criteria.

1. A conforming postprocessor shall read syntactically correct files in the physical file format of Annex B without failing. This requirement includes the syntax of the overall file as well as the syntax of each entity in the file.

2. A conforming postprocessor shall act on all files that are structurally correct and consistent in one or both of the following ways:

   Report on the results of the processing.

   Create native data entities that are valid within the context of the native system.

   The reports required here shall deal with three cases:

   Standard entities which are mapped into identical native entities (i.e., directly mapped) shall be reported simply with entity counts.

   Standard entities which are mapped into different native entities shall be reported by displaying the entity identifier and a description of the mapping used by the postprocessor.

   Standard entities which are not processed shall be reported by displaying the entity identifier and identifying it as not having been processed.

3. A conforming postprocessor shall map all data present in the standard form which has meaning in the native form into native entities and data structures for the following standard entities:

   Entities pre-defined in Clause 4 that can be directly mapped.

   Other entities pre-defined in Clause 4 that can be approximated in the native system.

   Implementor-defined entities created by the preprocessor associated with this postprocessor, if it exists.

   Excluded from this requirement are those data which are intrinsic to the structure of the standard form and which do not add meaningful information to the definition of the native entity or construct (e.g., entity identifier).

4. A conforming postprocessor shall be accompanied by a publicly published list of all entities defined in this standard and all implementor-defined entities created by the preprocessor associated with this postprocessor, if any, along with the corresponding native entity, or entities, and data structures to which they are mapped. An unsupported mapping will appear in this list as a standard entity for which there is no corresponding entry for the native entity or data structure.