T. FİDAN

FEATURE BASED DESIGN OF ROTATIONAL PARTS BASED ON STEP

TAHİR FİDAN

METU 2004

DECEMBER 2004

FEATURE BASED DESIGN OF ROTATIONAL PARTS BASED ON STEP

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

TAHİR FİDAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

DECEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences

_____

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof.Dr. Kemal İder
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Prof. Dr. S. Engin Kılıç
Supervisor

Examining Committee Members

Prof.Dr. Metin Akkök	(METU, ME)	_____
Prof. Dr. S. Engin Kılıç	(METU, ME)	_____
Prof. Dr. Ömer Anlağan	(TÜBİTAK)	_____
Prof. Dr. Mustafa İlhan Gökler	(METU, ME)	_____
Assoc. Prof. Tayyar D. Şen	(METU, IE)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Tahir FİDAN

ABSTRACT

FEATURE BASED DESIGN OF ROTATIONAL PARTS
BASED ON STEP

Fidan, Tahir

M.S., Department of Mechanical Engineering

Supervisor: Prof. Dr. S. Engin Kılıç

December 2004, 153 pages

The implicit and low-level part definition data, provided by geometric modeling cannot be used by downstream applications. Therefore, feature based modeling concept has been introduced to integrate CAD and downstream applications. However, due to the lack of implicit and explicit standard representations for features and unmanageable number of possible predefined features without standardization, feature based modeling approach has proved to be inadequate. STEP AP224 provides a standard for both implicit and explicit representations for manufacturing features. This thesis presents STEP AP224 features based modeling for rotational parts. The thesis covers features extracted from STEP AP224 for rotational parts and their definitions, classifications, attributes, generation techniques, attachment methods and geometrical constraints. In this thesis a feature modeler for rotational parts has been developed. STEP AP224 features generated are used as the basic entities for part design. The architecture of the proposed system consists of two three phases: (1) feature library, (2) feature modeler and (3) preprocessor. Preprocessor responsible from STEP-XML data file creation. The data file created can be used in the integration CAPP/CAM systems without using a complex feature recognition process. An object-oriented design approach is used in developing the feature modeler to provide incremental system development and reusability.

Keywords: STEP, AP224, Feature Based Modeling, Rotational Parts

ÖZ

DÖNEL PARÇALARIN STEP'E DAYALI
UNSUR TABANLI TASARIMI

Fidan, Tahir

Yüksek Lisans, Makina Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Engin Kılıç

Aralık 2004, 153 sayfa

Geometrik modelleme tarafından sağlanan yüzeysel ve üstü örtülü parça tanım verisi, tasarım-imalat zincirindeki uygulamalarca doğrudan kullanılamamaktadır. Bu uygulamaları ve CAD sistemlerini entegre edebilmek için unsur tabanlı modelleme geliştirilmiştir. Unsurların açık ve örtülü gösterimleri için bir standart olmaması ve standardizasyon olmadan da önceden tanımlanması gereken unsur sayısının kontrol edilemeyecek kadar çok olması, unsur tabanlı modellemenin yetersiz kalmasına yol açmıştır. STEP (Ürün Model Verisinin Aktarımı için Standart) AP224 imalat unsurlarının hem açık hem de örtülü gösterimleri için bir standart sağlamaktadır. Bu tezde dönel parçaların STEP AP224 unsurlarına dayalı modellenmesi sunulmaktadır. Tez, dönel parçalar için STEP AP224'den seçip çıkarılan unsurları, onların tanımlarını, sınıflandırılmasını, özelliklerini, oluşturulma, eklenme tekniklerini ve geometrik kısıtlamalarını içermektedir. Bu tezde aynı zamanda dönel parçalar için STEP'e dayalı unsur modelleyici de geliştirilmiştir. Parça tasarımı için temel olarak oluşturulan STEP AP224 unsurları kullanılmıştır. Sistem yapısı ana olarak üç bileşenden oluşur; (1) unsur kitaplığı, (2) unsur modelleyici, ve (3) STEP-XML veri dosyasının oluşturulması. Oluşturulan veri dosyası, CAD/CAPP (Bilgisayar Destekli İşlem Planlama) entegrasyonunda karmaşık bir unsur tanıma işlemine gerek kalmadan, kullanılabilecektir. Artımlı sistem geliştirilebilmesini ve tekrar kullanabilirliği sağlayabilmek için unsur modelleyici geliştirilirken nesne yönelimli tasarım yaklaşımı kullanılmıştır.

Anahtar Kelimeler: STEP, AP224, Unsur Tabanlı Modelleme, Dönel Parçalar

To My Family

ACKNOWLEDGMENTS

I would like to express my gratefulness and appreciation to my supervisor Prof. Dr. S. Engin Kılıç for his guidance throughout the completion of this thesis.

I am also indebted to my colleagues in Integrated Manufacturing Technologies Research Group (IMTRG) and mechanical engineering department for their endless support all through this hard work.

Finally, my greatest thanks go to my girlfriend and to my family who shaped me with their never ending patience.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Design and Manufacturing

In the engineering aspect of production, *design* and *manufacturing* are two major components [1]. Manufacturing processes cannot be effective without a thoughtful design for manufacturing. Likewise, a product design that cannot be realized through manufacturing processes is not a good design.

Traditionally, design and manufacturing are treated as two separate stages, which are usually handled by two distinct groups of people. The design group often does not anticipate the manufacturing implications of its decisions. After the detailed design is completed, it is passed onto the manufacturing group where product knowledge is stored in annotated drawings, binders, manuals and supplier data sheets. The manufacturing group then has to determine a way to manufacture the parts based on its interpretation of the design group's drawings, which may not be the same as the designer's intent. It usually takes a few passes back and forth between the two groups in order to reach a satisfactory part. Each pass may also mean a few scrapped parts and some retooling. Moreover, the design could be modified frequently, sometimes even before an acceptable part is manufactured.

With improved computer technologies and better understanding of its usages for design and manufacturing, the methods mentioned above are largely being replaced by Computer Aided Design (CAD) and Computer Aided Manufacturing (CAM) systems and various databases [2]. CAD systems are powerful geometric modeling tools and are used to allow interactive design of geometric models. Using a CAD

system, a part design can be created, modified interactively and graphically on a computer. This increases the productivity of the designer, improves the quality of design, and creates a database for manufacturing, which leads direct or indirect use of the model data in manufacturing processes. However, internally CAD and CAM applications works independently and uses their own data representation, which means CAD language is not same with CAPP (Computer Aided Process Planning) common language. In addition, unfortunately computers are not intelligent enough to recognize the CAM common language from the CAD language.

1.2 CAD/CAM Integration

A production cycle containing distinct steps, whether two separate groups of people responsible for design and manufacturing or CAD and CAM applications working separately, is a slow and costly process. It is clear that to reduce the time and cost, it is important to achieve a good integration of CAD and CAM applications, which provides a common language for both sides in which they can interact with each other and also anticipate design changes and reflects them to the manufacturing processes.

Computer Integrated Manufacturing (CIM) was introduced to further increase the degree of automation in production processes beyond CAD/CAM systems. [3]. The CIM concept is that all of the operations related to the production could be incorporated in an integrated computer system to assist, augment, and automate the processes. The same information could be shared throughout the entire life cycle of a product. One of the major challenges of computer-integrated manufacturing (CIM) is to make the languages of CAD and CAM common through CAD/CAM integration. CIM addresses the needs for communication, data exchange, and management between design and manufacturing, as well as other aspects of a product. It integrates design and manufacturing by shortening the transitions between them. One problem of existing CAD/CAM systems is, when the resulting data exported from CAD systems are imported into CAM systems, this procedure creates many incompatibility problems. Even if the geometrical data could be

transferred successfully, the data does not include much of the necessary data required for process planning like; part properties data (material, surface finish, etc.), tolerance data, etc. These problems in data format conversion build barriers against a full implementation of the CIM concept.

CAD/CAM integration raises two major issues, which are both to be resolved in order to manage a full integration. The first of these issues relates to the methodologies used to formalize the integration process through techniques for problem decomposition, the building of integrated information models and the establishment of appropriate information sharing techniques [4]. The second issue is formation of a part representation scheme, which is capable of acting as a carrier of information. A single representation is sought that serves the purposes of Computer Aided Design and Computer Aided Manufacturing as an effective way of unifying the information needs of CAD/CAM [5].

1.3 Feature Based Approach

In the early eighties, it was widely considered that the newly emerging geometric modeling techniques, that the CAD systems use, would provide the necessary complete and unambiguous part descriptions [6]. However, it is now generally accepted that geometric models require information enhancement before they are suited to the exacting requirements of concurrent engineering. Therefore, newer approaches related to the integration of CAD/CAM systems started using feature based approach that is feature based modeling instead of geometric modeling. They provide an integrated environment and framework for concurrent product design and manufacturing process planning.

In feature based modeling, feature based design and feature recognition, mechanical parts are represented as related sets of components each of which is represented as related sets of features. These features, when supported by a geometric solid modeler, have an enhanced information content including geometric information and attributes relevant to some or many manufacturing planning activities.

1.4 Part Representation Scheme

Feature based approaches are having difficulties in the formation of a suitably rich feature representation that might be capable of supporting many of the design and process planning activities, due to the lack of a universal feature library. These difficulties causes the current research work on feature based approach to show a definite bias to a small number of activities, which conflicts with the full CAD/CAM integration objective.

International recognition of these difficulties, in CAD/CAM integration, has resulted in standardization attempts to broaden the scope of CAD systems beyond geometric modeling and one step further than the feature based approaches by providing them a comprehensive universal feature library. Over the years many product representation schemas standing as the standard product data exchange formats have been developed. The first ones were national and focused on geometric data exchange. They included SET in France, VDAFS in Germany and the Initial Graphics Exchange Specification (IGES) in the USA. Later a grand unifying effort was started under the International Standards Organization (ISO) to produce one International Standard for all aspects of technical product data and named Standard for the Exchange of Product Data (STEP) for the Standard for Product Model Data [7]. Then, ISO introduced STEP that would serve the needs of all applications as the most promising answer to the integration problems mentioned. STEP Application Protocol (AP) 224, the ISO standard for Mechanical Product Definition for Process Planning using Form Features provides a feature library consisting of 98 manufacturing features generally classified as shown in Figure 1.1. In addition to feature library, STEP AP 224 provides geometrical and topological entities required to represent manufacturing features implicitly in boundary representation format, explicit representation of manufacturing features, information necessary to identify he dimensional and geometrical tolerances of the manufacturing features, information necessary to define material, hardness, surface finish and other technological data.

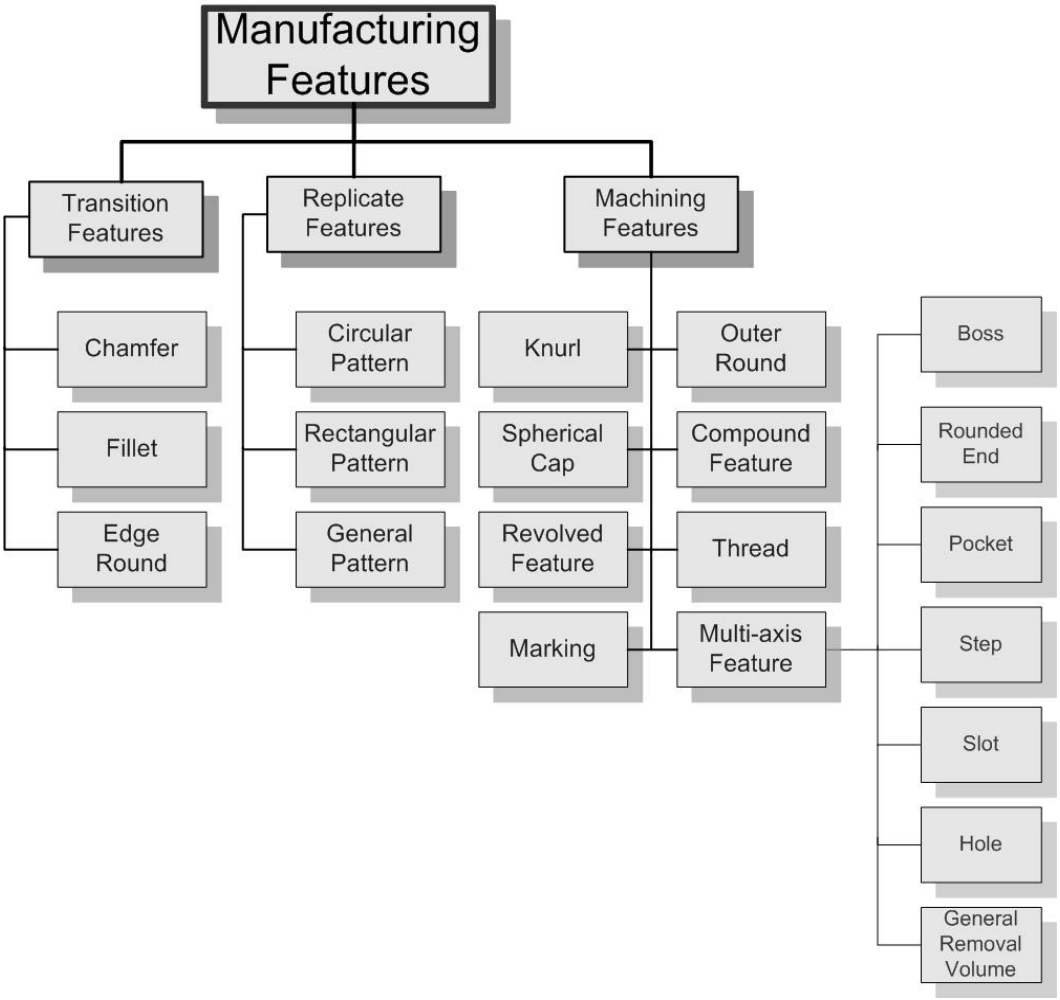

Figure 1.1 STEP AP 224 Manufacturing Features

1.5 Objective of the Thesis

CAE (Computer Aided Engineering) can be thought as a collection of computer assisted techniques each of which addresses a particular aspect of the realization of a product in its lifecycle. In a manufacturing engineering environment, this might include geometric modeling, process planning and CNC part programming. To increase productivity and cost effectiveness "Concurrent Engineering" approach should seek the integration of these functions such that there is some parallelism in their application. As it is mentioned, key to integration lies first in the methodology to formulate the integration process then in determination of representation schema

for products. Feature based modeling, considered as an indispensable tool for CAD/CAPP systems, is the methodology used to formulate the integration process. STEP AP224, providing a universal feature library for both implicit and explicit representations of manufacturing features, is the determined part representation schema for product data. Harmonizing these two, the main objective of the thesis is to develop a "STEP-Based Feature Modeler for Rotational Parts".

In order to achieve this goal, the following specific objectives have to be accomplished:

1) Developing an integrated feature library by using object-oriented approach and STEP AP224. The main tasks of this library are:
   - **a.** to provide a standard product data representation for the feature modeler,
   - **b.** to provide the feature modeler the ability to integrate with other systems in CIM environment efficiently.

2) Developing a STEP based feature modeler for rotational parts by enhancing the modeling capabilities of a conventional solid modeler, AutoCAD. Features in the developed feature library will be used as the basic entities for the part design.

3) Developing a unidirectional STEP AP224 translator. This translator includes a pre-processor, which exports data from feature modeler to STEP-XML format. The exported *neutral* STEP-XML file facilitates the generation of process plan of rotational parts by using STEP based Computer Aided Process Planning (CAPP) systems, which are being developed in Middle East Technical University Mechanical Engineering Department Computer Integrated Manufacturing Laboratory (METUCIM). STEP based CAPP system will map the product information in the neutral file generated by the feature modeler and will produce the corresponding machining operations to generate the process plan.

1.6 Scope of the Thesis

The scope of this study is to develop a feature modeler for rotational parts based on STEP. There are two main steps to be followed to develop the feature modeler: (1) building up a feature library for rotational parts, (2) development of the modeler using the features in the feature library.

A feature library based on STEP AP224 standard is developed for rotational parts in terms of manufacturing features. The feature library includes features extracted from STEP AP224 for rotational parts and their definitions, classifications, attributes, generation techniques, in every detail. Features are so selected that almost every possible rotational part feature, which is defined in STEP AP224, is included into the feature library. Figure 1.2 provides a general view for the manufacturing features for rotational parts in the feature library. EXPRESS is an object-oriented data descriptive language, which classifies and constructs product data in terms of entities. EXPRESS enables precision and consistency of product data representation and facilitates implementation. [8] By means of EXPRESS language that STEP AP224 provides for every feature, the feature library is being developed in an object-oriented manner, which makes the feature library easy to maintain and extend. By this way, the feature library is implemented as an object-oriented data type library. It is created as a "dynamic link library", called "RotSTEPFeat.dll", which can be used as the data source of the proposed feature modeler. In the "RotSTEPFeat.dll", each feature is defined as class modules, keeping the hierarchical architecture defined in the standard and the file is compiled by using Visual Basic 6.0. This feature library provides the feature modeler to be "STEP-based".

A STEP-based feature modeler is developed to use the defined feature library in an environment fulfilling the requirements of concurrent design. The feature modeler is implemented in AutoCAD environment, to implement the feature modeler in AutoCAD environment, Visual Basic 6.0 and ActiveX technology is used. By this way, AutoCAD became capable of providing "feature based design of rotational

parts based on STEP". Developed feature modeler provides the designer an easy to design environment, which is 3D, based on rotational manufacturing features, which removes the geometry based mass and complexity of traditional CAD systems. Moreover, since the part is designed by using predefined standard features, it also removes the manufacturability problems that may occur after the design process. In addition to all these, the ability to export STEP-XML file based on STEP AP224, of the feature modeler facilitates to obtain a file combining both implicit and explicit features' data with technological attributes. This file can directly be used by STEP based CAPP systems and the process plan of the designed part can be obtained in an integrated manner.

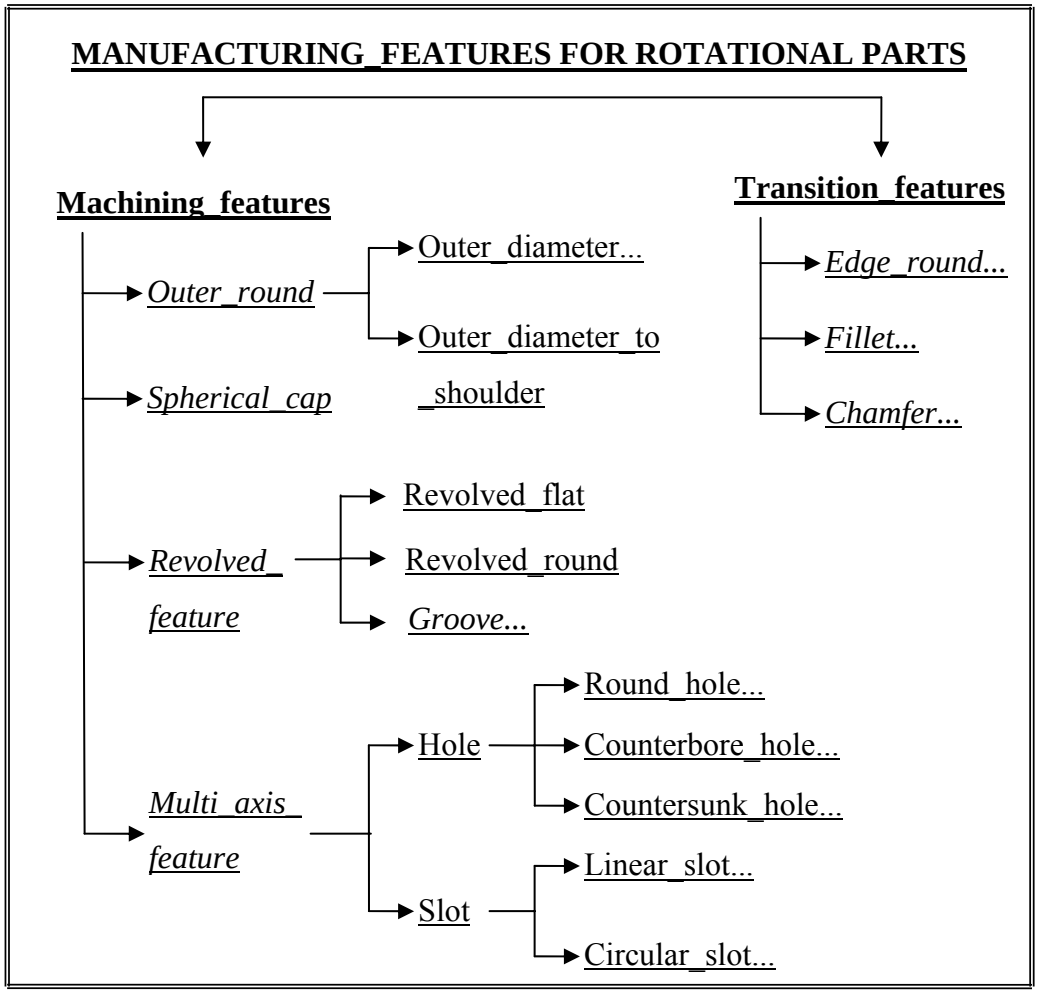

Figure 1.1 Manufacturing Features for Rotational Parts

1.7 Outline of the Thesis

In Chapter 2, a survey related to product design representation approaches, product data models and software technologies used like object oriented programming, unified modeling language (UML), extensible markup language (XML) is presented.

In Chapter 3, model of the developed system, including the architecture of both the feature library and the feature modeler, is described in every detail.

In Chapter 4, steps followed for the system developed is described in a detailed manner, proving example work done for one of the manufacturing features for rotational parts, selected form the feature library.

In Chapter 5, sample drawings and their corresponding STEP XML files are presented and the relation between these outputs is described, to demonstrate the operational performance of the developed system.

In Chapter 6, concluding remarks and possible future work plans recommended to complete the integration process are given.

In Appendix A, sample source codes are given, describing different steps of the software developed. In Appendix B, for each feature in the feature library figures describing the feature geometries are presented.

CHAPTER 2

LITERATURE SURVEY

2.1 Product Design Representation

There are two main approaches used in the product design representation: geometric modeling and feature modeling. A detailed explanation of the models will be presented in the following section.

2.1.1 Geometric modeling

Geometric models are classified as 2D or 3D models. 2D geometric models are wireframe models and 3D geometric models are classified as wire frame, surface or solid models [9].

2.1.1.1 Wireframe Models

Wireframe models are the earliest type of geometric model, dating back to 1960. Both 2D and 3D wireframe models represent objects by the edge lines, arcs, and points on the surface of the object. A wireframe model is the same as skeletal descriptions of the product being designed. It should be noted that there are no visible surfaces on the wireframe model, only geometric entities such as lines, arcs and points. Although wireframe models do not look like a solid object, they do contain an accurate geometric description of the object being modeled.

Wireframe models are practical because of the speed with which they can be displayed. Since a design workstation does not need a sophisticated color video

monitor to display complex wireframe models; it is inexpensive to model objects using the 3D wireframe technique. The display of a wireframe model is often an ambiguous representation, because it can be impossible for the viewer to determine which lines are in the foreground and which are in the background. Since wireframe models do not contain any information about the space between the edges, it can be difficult to determine, for instance, if two objects will interfere with each other. In addition, it is possible to create a wireframe model of a nonsense object, that is, an object that is a physical impossibility.

2.1.1.2 Surface Models

Surface models were first developed in the early 1960's. Surface models improve on wireframe models by including face information. They can model a 3D object without any ambiguity. In a surface model, it can be determined whether or not a point is on the surface. When several surfaces form an object, it cannot generally be determined whether a point is inside or outside the object unless some additional information is available indicating this. The mathematical representations for surface models are a set of surface equations. As far as computer representations are concerned, most of the plane surfaces can be represented or approximated using polygons. A surface model is represented in the computer by vertices, edges and faces.

An advantage of surface models is that they are easy to construct by creating plane surfaces, as well as by sweeping, revolving, or extruding entities. Surface models are also useful for finding the intersection of surfaces in space and creating models for shaded rendering. Surface modeling approach's main fault is that it cannot represent the interior of the model as solid. Therefore, surface models cannot represent properties needed to analyze a product's internal structure.

2.1.1.3 Solid Models

Solid models were developed in the early 1970's. Solid models are a complete and

unambiguous description of the object being represented. The construction procedure for solid modeling is different from these for wireframe and for surface modeling. Instead of having to generate specific lines, arcs, and surfaces that define the object, the designer uses mathematically predefined solid primitives, such as locks, cylinders, cones, wedges, spheres and son on. Most CAD modeling packages have a limited number of primitives available, but the designer can use them creatively to model very complex shapes. To create complex shapes, the designer can combine primitives using the Boolean operations: *union* (the sum of two primitives), *intersection* (the common mass shared by two primitives), and *difference* (subtracts a primitives from another). Since solid contain more information about the closure and connectivity of shapes than wireframe and surface models, they have become the most important type of model for designing, analyzing, and manufacturing products. Solid models offer a number of advantages over surface models, including the ability to calculate mass properties such as weight and center of gravity.

There are several representation schemes developed and used in the solid modeling software such as constructive solid geometry (CSG), boundary representation (B-rep), primitive instancing, cell decomposition, etc. The most popular representation schemes for CAD solid modeling packages are CSG and B-rep.

Solid models have the following inadequacies:

- ✓ *Incomplete database:* Solid models can only be used to define the nominal geometry. Information regarding surface finish, tolerances, material properties, surface conditions, etc., is important parts of the definition of mechanical parts, but these cannot be incorporated in a solid model database.
- ✓ *Mismatch in abstraction level:* Solid models store data in terms of low-level entities such as vertices, edges, faces, etc., or binary trees that contain primitives and Boolean operators. It is difficult to extract the engineering meaning of this data from the solid models database.

2.1.2 Feature modeling

CAD systems use the geometric modeling approach, several of which were discussed in the previous section, for product design representation. These models make the CAD systems powerful in geometric modeling. However, the design information provided by CAD systems is implicit and in terms of low-level primitives, which has limited use in conducting comprehensive manufacturing analysis and cannot directly support many manufacturing applications where technical, functional and other information is crucial to shape data. Therefore, the design information provided by the CAD system need to be translated into explicit manufacturing information such as part features in order to be understood by various downstream application (CAM, process planning, CNC programming, group technology, inspection, assembly etc.) . Thus, features serve as a link between the CAD and downstream applications. Features in general sense have been variously defined. Among many others, one definition of feature is "recurring patterns of information related to a part's description" [10]. Features retain a high level of abstraction of part's description that means features provide not only geometric and topological entities but also dimensions, tolerances, materials, surface finishes…etc.

Downstream applications deal with high-level manufacturing applications such as features as described, instead of pure geometric entities. In this sense, feature modeling approach has been developed towards representing high-level design information and significant research efforts have been done on feature modeling. Significant research efforts have been made towards representing the high-level design information, commonly available in engineering drawings, in a CAD system. These efforts have resulted in various types of feature representations. There is a rich literature in applying feature concepts to the integration of design, process planning and CNC programming. Feature modeling is essentially grouped into two distinct approaches, namely feature recognition and feature-based design [11].

2.1.2.1 Feature Recognition

Feature recognition, examines the topology and geometry of a part and matches them with the appropriate definition of predefined features. Part-feature recognition algorithm for rotational parts has been developed in previous researches [12]. Advantage of feature recognition is that the designers can work directly on the current CAD system, which they have been using. However, it has couple of disadvantages. Challenging feature recognition algorithms need to be developed and it is a complex and time consuming process, tough further refinements of the recognition algorithms is necessary. In case of feature interactions, incremental feature validation is required that is the partially recognized part has to be checked whether any features are interacting or not. Furthermore, feature model conversion has to be done to convert the recognized form features into manufacturing features in order to provide them as useful to downstream applications [13].

2.1.2.2 Feature Based Design

Feature based design, builds a part from predefined features where their attributes are attached. Features can capture the functional intent of the part within its geometry based representation. This property of features can facilitate high-level communication between design and manufacturing. However, feature based design limits designers' ability to create complex parts due to the limited number of predefined features that can be stored in the feature library. The existing CAD systems cannot be used by the designers, a new CAD system should be developed (or an existing one should be modified) in order to embed the feature library with predefined features and their attributes. Feature modeling approaches have their own advantages and limitations but the difficulties facing both approaches are the lack of implicit and explicit standard representations for features and unmanageable number of possible predefined features without standardization. Researchers attempted to define a comprehensive library of their own features from mechanical parts. In one research, computer integrated manufacturing system for rotational parts have been established [14]. The same part data is used in all the CAD, CAM

and CAPP modules. The basic primitives and manufacturing features are used to define the feature library. Same with the other attempts the problems are that each new part gives rise to several new features, in addition to ones previously defined and in order to deal with other CAD systems' outputs, feature recognition algorithms had to be developed due to the lack of universal feature library. Therefore, research on this subject were leaving local to the particular manufacturing plants where feature library is designed accordingly as much as the limitations of feature recognition is concerned.

2.1.2.3 Review of Feature Modeling Approaches

Presented feature modeling approaches have their own advantages and limitations, however the difficulties facing both feature recognition and feature based design approaches are:

- The lack of implicit/explicit standard representation for features, the lack of universal feature library,
- The necessity of defining a comprehensive feature library, which has its own difficulties;
  - Unmanageable number of possible features to be predefined without standardization,
  - Each new part gives a rise to several new features in addition to the ones previously defined.
- The need for feature recognition algorithms to make the developed feature modeling system compatible with other CAD systems' outputs.

These difficulties limit the developed systems to being application specific, a characteristic that is in conflict with the integration objectives. The following section provides an insight into Product Data Models that are in existence, developed to overcome these difficulties.

2.2 Product Data Models

The purpose of a product data model is to provide a means for representing and exchange information about a product gathered during, and used in, the design and manufacture of that product. Therefore, the contents of this product model must be able to support the information needs of a large variety of computerized manufacturing applications (i.e., CAPP, part programming, etc.). The popularity of using CAD systems as a means for creating, representing and exchanging product designs has created various standard product data exchange formats such as IGES, SET, DXF, etc. These standards have shown a success in transferring data between CAD systems, but they have failed to transfer product data from CAD to CAM applications. This is because current CAD systems are not able to support all the information concerning a part that is needed to support the CAM activities. As a solution, the International Standards Organization (ISO) first proposed Product Data Exchange Specification (PDES) [15]. Since PDES contains both feature model and dimension/tolerance model, it is potentially very useful in providing a link between CAD and downstream applications. PDES however is relatively complex and uses terms that are not familiar to a designer. Research work done on mapping design-feature taxonomy of rotational parts onto PDES-features in order to make terms familiar to designer [16]. Offering the most promising answer to these problems, ISO introduced the Standard for the Exchange of Product Data (STEP) that would serve the needs of all applications

In the following section, standards in existence that have been approved (either by national or international standards committees) and accepted as defacto, will be presented. A defacto standard is an unofficial standard that is widely used throughout industry.

2.2.1 Initial Graphics Exchange Specification – IGES

Efforts to define a formal standard for product data exchange began in 1979 with the development of IGES. IGES development was based upon the concepts, which

used a neutral format and half translators. Upon development, the CAD vendors implemented the standard realizing that the provision of the ability to transfer data across different systems was a valuable product feature and sales hinged upon the provision of the feature. The initial IGES standard was launched in January of 1980. Problems such as limited scope (CAD only), accuracy, and inability to certify or provide a conformance check on the software to assure consistent implementations by the various vendors. These problems arose as a consequence of an initial effort in a large domain. Research efforts began to correct these shortcomings, lead predominantly by American bodies, such as the United States Air Force, Army and Navy, and the National Aeronautics and Space Administration (NASA). As it improved, IGES use was embraced by many industries around the world as a solution to the data exchange problem. IGES became an accepted American national standard under ANSI Y14 (American National Standards Institute), it was adopted by many of the national standard bodies throughout the world. It is currently being revised in to version 6. Version 6 is likely to be the last version/upgrade to IGES. STEP is now the focus of most of the data exchange due to its increased scope and incorporation of lessons learned.

2.2.2 Standard D'Exchange et de Transfert – SET

SET was a French effort to create a standard to exchange CAD data. These efforts were driven by French industry, most notably the automotive and aerospace industries. This requirement is due to the industries' high usage of CAD systems. SET, like IGES in the US became a French national standard. In recognition of the need for a single international data exchange standard, the French efforts are now focused on developing the STEP standard and they are very active in the development of STEP.

2.2.3 Verband Der Automobilindustrie-Flachen-Schnittstella - VDA-FS

The development of the VDA-FS standard was a German effort in response to the data exchange requirements of their automobile industry in the 1980's. VDA is the

German automotive industry trade association, and was the principle developer of the VDA-FS standard. VDA focused on defining a standard that allows for transferring surface/shell data. The Germans are now also directing their data exchange standards development efforts to STEP.

2.2.4 Data Exchange Format – DXF

The use of the DXF format for data exchange has evolved as a defacto standard. It is a proprietary format published by AutoDesk, a major CAD vendor. Early versions of AutoCAD focused on drafting with later attention being focused on solid 3D models. As the most widely used CAD packages, it is used throughout industry, including, building and construction, aerospace, automotive, process, shipbuilding, electrical, industrial, and consumer products. While addressing CAD data exchange, DXF does not include the scope of product model data that is included in STEP.

2.2.5 Standard for the Exchange of Product Model Data – STEP

STEP, standing for Standard for the Exchange of Product Model Data, is officially titled ISO 10303. The aim of STEP is to provide a representation of product information along with the necessary mechanisms and definitions to enable product data to be exchanged. The exchange is among different computer systems and environments associated with the complete product lifecycle including design and manufacture. The information generated about a product during these processes is used for many computer systems, including some that may be located in different organizations. In order to support such uses, organizations must be able to represent their product information in a common computer-interpretable form that is required to remain complete and consistent when exchanged among different computer systems [17].

STEP is organized as a series of parts, each published separately. These parts fall into one of the following series: description methods, integrated resources,

application protocols (APs), abstract test suites, implementation methods, and conformance testing. STEP uses a formal specification language, EXPRESS [18], to specify the product information to be represented. The use of a formal language enables precision and consistency of representation and facilitates development of implementations. To transfer this information, STEP usually employs the neutral file approach. Transfer of data from one application to another is usually a two-step process requiring a post-processing and pre-processing.

The overall objective of STEP is to provide a mechanism that is capable of describing product data throughout the life cycle of product, independent from any particular system. The nature of this description makes it suitable not only for neutral file exchange, but also as a basis for implementing and sharing product databases and archiving. The ultimate goal is an integrated product information database that is accessible and useful to all the resources necessary to support a product over its lifecycle [17].

STEP uses application protocols to specify the representation of product information for one or more applications. The APs define the scope, the information to be exchanged, the means of testing and a user guide for implementing the application. The STEP Application Protocol of main interest for this thesis is Application Protocol 224 (Mechanical Product Definition for Process Planning using Machining Features) and Application Protocol 238 (Application Interpreted Model for Computerized Numerical Controllers) is referred for lacking information in AP224. STEP AP224 contains all the information needed to manufacture the required part, including [19]:

- Geometrical and topological entities required to represent manufacturing features implicitly in boundary representation format.
- Explicit representation of manufacturing features.
- Information necessary to identify the dimensional and geometrical tolerances of the manufacturing features.
- Information necessary to define material, hardness, surface finish and other technological data.

STEP AP238 defines the context, scope, and information requirements for numerical controlled machining and associated processes and specifies the integrated resources necessary to satisfy these requirements. It is still a draft work of ISO and is being developed. STEP AP238 provides features information from manufacturing point of view and harmonizes the data with other standards. Some lacking information in STEP AP224 about feature geometries is being extracted from STEP AP238.

2.2.6 Review of Product Data Models

The following is a brief summary of the product data models described:

- IGES, SET, DXF, VDA-FS: Shown a success in transferring data between CAD systems, however failed to transfer product data form CAD to CAPP systems. Because current CAD systems are not capable of storing CAPP applications related information concerning a part.
- STEP: Most recent and promising standard for representing part data that would serve to needs of variety of applications and as an answer to all the difficulties stated.

2.3 Object Oriented Programming

Object-oriented programming is claimed to be the software technology for the 1990s and beyond. Object-oriented programming approach is a general term for a set of analysis, design and programming methodologies. Using object-oriented programming approach, design and develop a system from the object perspective, can be analyzed. Objects are intelligent, self-contained entities responsible for performing particular system tasks [20]. Thereby, an object is defined as an identity, encapsulating some private data and a set of operations to access that data. Four main features of objects are [21]:

1. **Messaging:** A message specifies *what* is to be done and the object decides *how* it is to be done. Calling programs need not to be aware with the internal representations of the internal functions the object uses.
2. **Encapsulation:** It involves the ability to hide the implementation details of a system so that it is accessed in terms of its properties rather than of its syntactical obligations. So, that each object is an independent entity in its own, regardless of which language is used in implementing it.
3. **Dynamic Binding:** Binding is the determination of which piece of code to run for a particular task in a program. Bindings to an object can be removed instantaneously to limit the use of valuable run-time memory.
4. **Inheritance:** Objects are organized into a hierarchy of classes that share characteristics among its members. Inheritance is a technique for defining new data type differs from some pre-existing type. From a software engineering point of view, inheritance allows a developer to reuse a piece of code that incorporates only a slight modification or extension to previously written code.

The common interfaces of objects are:

- **Properties:** Changeable / retrievable features,
- **Methods:** Actions, functions which are performed by the object,
- **Events:** External inputs, to which the object is susceptible.

2.4 Unified Modeling Language

The Unified Modeling Language (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. It simplifies the complex process of software design, making a "blueprint" for construction. UML is the most widely known and used standardized notation for object-oriented analysis and design. Yet it does provide several types of diagrams that, when used within a given methodology, increase the ease of understanding an application under development. The most useful, standard UML diagrams are; use case diagram, class diagram, sequence diagram, state chart diagram, activity

diagram, component diagram, and deployment diagram. There is more to UML than these diagrams, for the purpose of this thesis, class diagrams and their notation will be described in the following section [22].

2.4.1 Class Diagrams

The purpose of the class diagram is to show the static structure of the system being modeled. A class is a collection of objects with common structure, common behavior, common relationships, and common semantics. The UML representation of a class, a class diagram, is a rectangle containing three compartments stacked vertically. The top compartment shows the class's name, pointing to the object mentioned in the Object Oriented Programming approach. The middle compartment lists the class's attributes with its variable types, pointing to the object properties mentioned in the Object Oriented Programming approach. The bottom compartment lists the class's operations, pointing to the object methods mentioned in the Object Oriented Programming approach. In Figure 2.1, a simple example of a class diagram is shown [23].

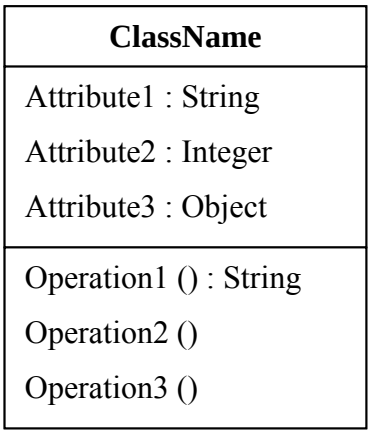

Figure 2.1 A generic class diagram showing a single class

The UML modeling elements found in class diagrams include:

- Classes, their structure and behavior.
- Association, aggregation, dependency and inheritance relationships.

The ***Inheritance*** is related modeling element of the class diagram with this thesis. As it is mentioned in the previous section, as a very important concept in object-oriented programming approach, inheritance, refers to the ability of one class (child class) to inherit the identical functionality of another class (super class), and then add new functionality of its own. To model inheritance on a class diagram, a solid line is drawn from the child class (the class inheriting the behavior) with a closed triangular arrowhead pointing to the super class. Consider types of shapes in the simple example Figure 2.2, it shows how both Circle and Square classes having their own attributes inherit methods from the Shape class.

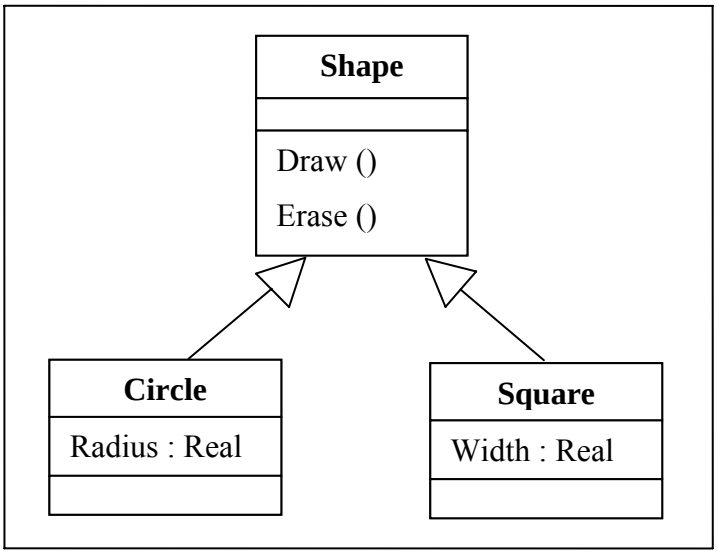

Figure 2.2 Inheritance is indicated by a solid line with a triangular arrowhead pointing at the super class

2.5 Extensible Markup Language

EXtensible Markup Language (XML) is a cross-platform, software and hardware independent tool for transmitting information. XML is a markup language much like HyperText Markup Language (HTML). However, while HTML was designed to display data and to focus on how data looks, XML is designed to describe data and to focus on what data is. XML is self-descriptive. Unlike HTML, XML tags are not predefined and XML allows the author to define his own tags and his own document structure, in other words XML tags are invented by the author of the XML document. XML uses a Document Type Definition (DTD) or an XML Schema to describe the data. [24] These bring simple syntax for XML, like; all XML tags must have an opening tag and closing tag enclosed in angle brackets, XML tags are case sensitive, XML elements must be properly nested, all XML documents must have a root element and XML tags may have attributes defined in quotation marks. Below is a simple sample XML document describing the book, to illustrate the syntax mentioned above. Book, title, chapter, paragraph are the tags used in this example. Book is the root element. Title, product and chapter are child elements of book. Book is the parent element of title and chapter. Title, product and chapter are siblings (or sister elements) because they have the same parent. Paragraph is the child element of chapter. Product has attributes like “id” and “media”, defined in quotation marks.

```
<book>
    <product id=”1-1” media=”paper”></product>
    <title>My First XML</title>
    <chapter>Introduction to XML
        <paragraph>What is HTML</paragraph>
        <paragraph>What is XML</paragraph>
    </chapter>
</book>
```

XML does not do anything, XML was created to structure, store and to send information. In the real world, computer systems and databases contain data in incompatible formats. One of the most time-consuming challenges for developers has been to exchange data between such systems. Converting the data to XML can greatly reduce this complexity and create data that can be read by many different types of applications Since XML data is stored in plain text format, XML provides a software- and hardware-independent way of sharing data. This makes it much easier to create data that different applications can work with. It also makes it easier to expand or upgrade a system to new operating systems, servers, applications, and new browsers. Other clients and applications can access XML files created as data sources, like they are accessing databases. XML data can be made available to all kinds of "reading machines" (agents) by means of processors.

CHAPTER 3

SYSTEM MODEL

3.1) General System Architecture

Three main objectives of this study were: (1) developing an integrated feature library, providing standard product data representation and facilitating the CAD/CAM integration process, (2) developing a STEP based feature modeler for rotational parts using the features in the feature library, (3) generating a neutral STEP-XML file that will directly be used by the STEP based CAPP systems. To achieve these objectives, a system architecture, which is presented in Figure 3.1, is developed. In the following sections of this chapter, each building block that constructs the whole system architecture will be defined; the idea behind the working principles of the system and their components will be described, in every detail. In the next chapter, Chapter 4, one of the manufacturing features for rotational parts in the feature library will be selected and every stage followed in the development of the system, described in this chapter, will be demonstrated on that feature in an illustrative and detailed manner. One of the features will be selected, because there are so many numbers of features with their subtypes and they show similarities in application. Following is a brief description of the main system components shown in Figure 3.1:

Feature Library: This integrated feature library will serve as the basis of the whole system, fulfilling the first objective of the study. "Feature Definitions Library" together with the "Feature Dynamic Link Library (dll)" builds up the "Feature Library". STEP AP 224 documentation, object oriented approach, MS Visual Basic

6.0 and EXPRESS Schemas are the tools used in an organized manner to bring the feature library up to a state satisfying the requirements of being a comprehensive library. By comprehensive library, at this stage, what is meant is a library facilitating the whole system to be “feature based”, “STEP based” and “object oriented”.

Feature Modeler: Feature Modeler is a software package enabling high-level 3D solid manufacturing features based design of rotational parts, fulfilling the second objective of the study. The feature modeler uses the manufacturing features in the developed feature library as the basic entities of the rotational part design. MS Visual Basic for Applications, ActiveX Automation, AutoCAD 2000i environment and error handling methods are the technologies used to develop the feature modeler provided that an easy to use, efficient and 3D environment is implemented. The inside structure of the feature modeler will be described in the following sections of this chapter.

Preprocessor: This is a unidirectional translator, which takes features data from the feature modeler as an input and creates STEP-XML file as an output. EXPRESS Schemas, used to determine the hierarchical structure, relations and attributes of the features while the feature library is developed, is now used in the same way to reflect this structure to the output file. This way most appropriate XML representation for each EXPRESS definition can be chosen and mapping of feature data into XML file can be achieved. For each feature in the feature library, algorithms are created to provide the creation of corresponding output data including all of the feature attributes. Preprocessor appends the outputs of each algorithm in a logical order and results with an overall STEP-XML file.

STEP-XML File: This file is the final output of the system and is created in XML format invented for STEP, which is descriptive and which will pioneer the data exchange to other systems, especially STEP based CAPP systems, from the feature modeler. The output file is generated in XML format due to the emergence of XML as standard for describing data exchange files.

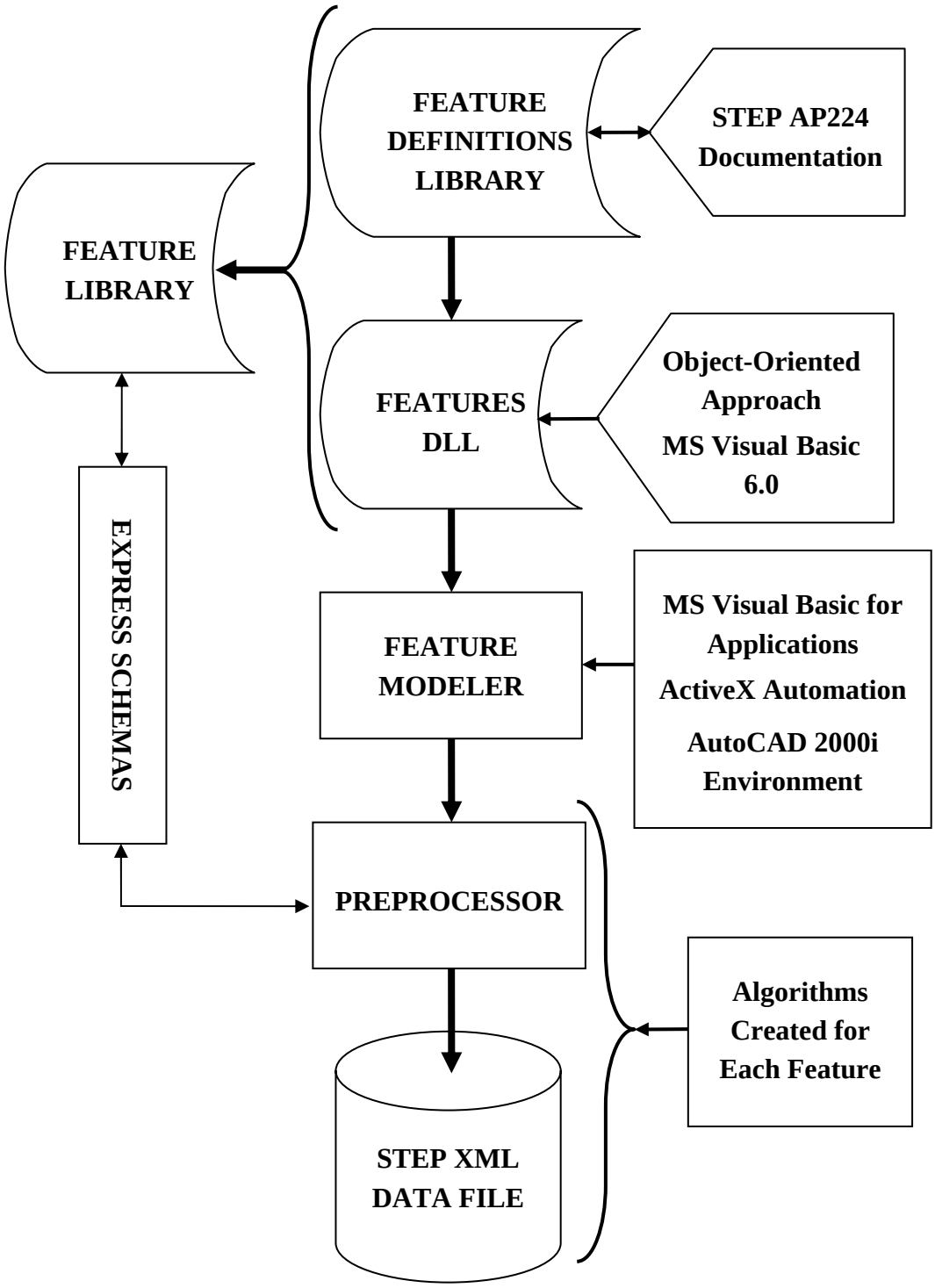

Figure 3.1 General System Architecture

3.2) Features Library

The structure of the feature library is shown in Figure 3.2. Feature library, in general, enables rotational parts to be designed by three types of data namely: (1) manufacturing features geometry data, (2) tolerance data and (3) part properties data by means of its "Features Definitions Library" component. While the manufacturing features geometry data is included into the feature definitions library STEP AP 224, STEP AP 238 documentations and related EXPRESS Schema are used. While the tolerance and part properties data are included into the feature definitions library STEP AP 224 documentation and related EXPRESS Schemas are used to construct the general data structure.

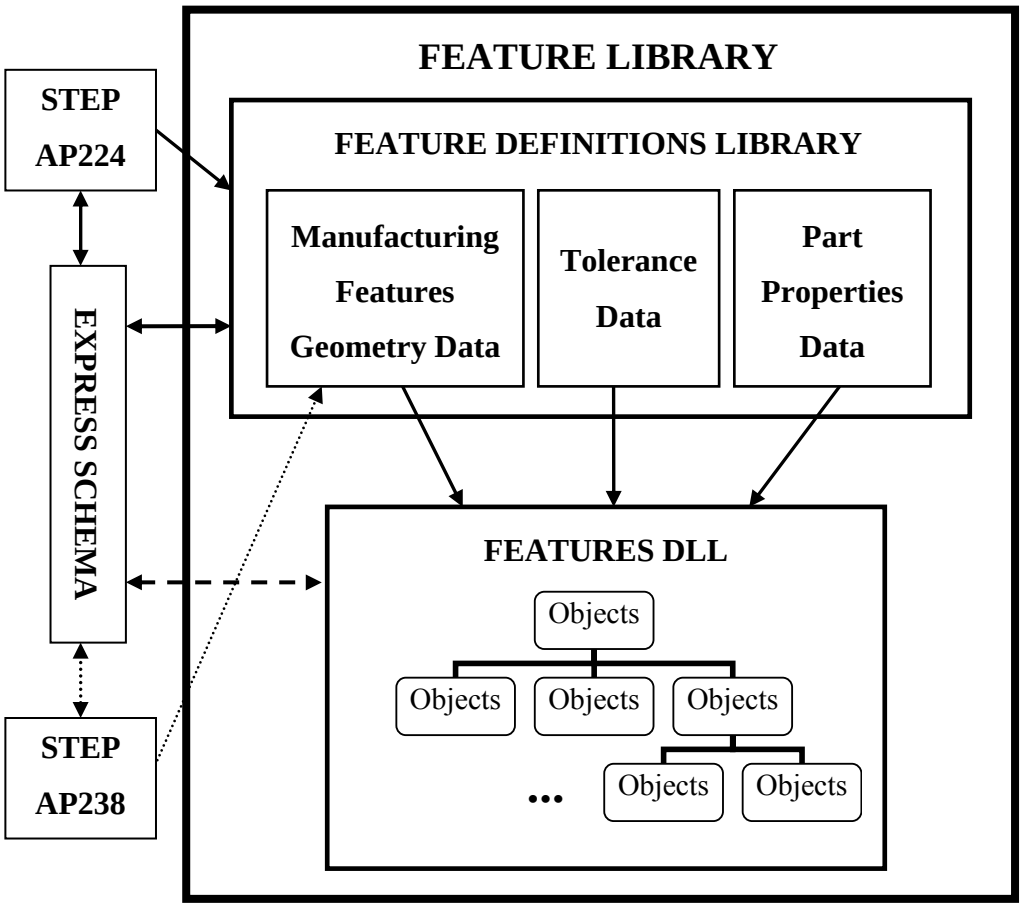

Figure 3.2 Structure of the Feature Library

Feature library is also a predefined library of objects, which remain resident during a session started in the system by means of its “Features Dll” component. Within the modeled feature definitions library, every feature is considered as an object, but these can be categorized as classes, superclasses, subclasses or instances. In terms of manufacturing features, a class is a generic description of one or more similar objects (features) defined in the feature library. In fact, these definitions are validated through EXPRESS Schema that is used in the definitions of the features and their attributes, during the development of the features dll component. This approach provides the developed system to be “object oriented”, which will be detailed in the following subsections.

3.2.1) Feature Definitions Library

Feature definitions library is a library in which every feature that will construct the primitives for the development of features dll, though the feature modeler is explicitly defined. The feature definitions library is created:

- To provide a good documenting that will guide throughout the study.
- To provide a database for Features Dll file creation and facilitate programming.
- To facilitate unambiguous and effective design for the feature modeler.

In feature definitions library, a product data model based on STEP is developed. It includes almost all the information required to design rotational parts and some information required to facilitate the development of a CAPP system for rotational parts (this content of the feature definitions library should be enhanced to achieve full integration). STEP AP 224, STEP AP 238 documentation as reference standard and data modeling language called EXPRESS as product data modeling language are used to create the feature definitions library. By this way, product data model capable of storing three types of data:

1. manufacturing features geometry data,
2. tolerance data,
3. part properties data

are developed in the feature definitions library. The content of these three components of data types will be described in the following subsections. A brief insight into both the STEP documentation and EXPRESS is provided in this section.

STEP Documentation: The STEP AP 224 and AP 238 documentations are quite long and most of them describe the Application Interpreted Model (AIM). Before, becoming familiar with AIM, Application Reference Model (ARM) parts of the documentation are covered. ARM describes the basic application objects. Application objects stands for manufacturing features or other objects used defining tolerance or part properties data, in the scope of this study. In general, what each application object represents and where the information contained in it will come from or go to, is reviewed as much as the ARM parts of the documentation is concerned.

Once ARM and application objects are understood the related parts of AIM is considered. The AIM is considerably more involved than the ARM, but it represents the same basic information. The AIM is an EXPRESS information model that formally describes the application objects in terms of a library of pre-existing definitions, called the generic resources or integrated resources. This highly normalized representation contains the structures as well as the constraints that those structures must obey. The AIM is used as the basis for the implementation and data exchange. EXPRESS-G diagrams, for which an example on one of the manufacturing features will be represented in Chapter 4, in AP document, provided an easier way to cover AIM. EXPRESS-G is a formal diagrammatic form for the EXPRESS language. EXPRESS-G diagrams contain the data structures, inheritance relationships, attributes and relationships between structures in the EXPRESS information model. The EXPRESS-G diagrams are very good at conveying the data structures associated with an information model.

By following the methodology defined, the necessary standardized features data could be extracted from the related STEP documentation.

EXPRESS: EXPRESS is a data modeling language defined by ISO and published as ISO 10303-11 [18]. The EXPRESS language allows defining a data model in terms of data structures constraints, and rules. The EXPRESS language is not case sensitive. As a convention, EXPRESS keywords are often written in uppercase to improve readability. EXPRESS is not a programming language; it is a data modeling language. EXPRESS is readable and fully computer interpretable. A brief introduction to EXPRESS is made below, which will describe the structure that is claimed to lead the whole system to be object oriented. This introduction will guide through the examples presented in the system development chapter.

EXPRESS Schema: EXPRESS specifications are organized into schemas. An EXPRESS schema is a name space of named data types. Data types may be simple types such as strings and integers or entity types, representing more complex collections of attributes (properties). Schemas can be related together to form models.

Entities: An entity is analogous to an object in object oriented programming. Each entity has a name and a set of attributes. Each attribute has a name and a data type. An entity, in EXPRESS schema, definition has the following form:

SCHEMA *SchemaName**;** -- Schema declaration*
***ENTITY** Entityname**;**-- Entity declaration*
*a1: data_type**;***
*a2: data_type**;***
*(*Number of, attributes may go up to any number *)*
*aN: data_type**;***
***END_ENTITY**; -- End of entity declaration*
END_SCHEMA; -- *End of schema declaration*

As it is seen above, there are two types of comments in EXPRESS. A *tail remark* is

written at the end of a line. Two consecutive hyphens "--" start the remark and a new line terminates it. An *embedded remark* begins with the character pair "(***"** and ends with the same character pair "*****)".

Data Types: The data type of an attribute can be either *a simple type, an aggregate type, an entity type, a defined type, an enumeration type,* or *a select type*. The first three data types are the mostly related ones in the scope of this study. These three data types are described below:

Simple Data Types: EXPRESS has several implicitly defined primitive data types. These are **integer, real, Boolean, logical, string,** and **binary**. These are analogous to ordinary variables in programming languages. Throughout the examples, integer and real data types are represented under the same name *"numeric_parameter"* to provide uniformity.

Aggregate Data Types: An aggregate is a container that holds multiple elements of the same type. They are defined in closed square brackets with a lower and upper bound values. The EXPRESS aggregate types are:

Bag: Bag is an unordered collection, in which duplicate values are allowed, but null values are not allowed.

List: List is an ordered collection, in which duplicate values are allowed, but null values are not allowed.

Set: Set is an unordered collection, in which neither null values nor duplicate values are allowed.

Array: Array is an ordered collection of fixed size, in which both null values and duplicate values are allowed.

Entity Data Type: Any entity declared in a schema can be used to specify the data type of an attribute. Using an entity as an attribute's data type establishes a relationship between the two entities. For instance, using an Entity named "*Point*"; another Entity named "*Line*" can be defined as it is presented in the example that follows.

ENTITY *Point*;
x: *numeric_parameter*;
y: *numeric_parameter*;
END_ENTITY;

ENTITY *Line*;
start_point: *Point*;-- *defined as Entity Point*
end_point: *Point*;-- *establishes the relation between Line and Point*
END_ENTITY;

Inheritance: Subtypes in EXPRESS allow new types to be derived from existing types. The derived types are "almost like" other existing types, with some incremental changes. They inherit attributes and functionality from their supertypes. Subtypes can define additional attributes and functionality, thereby extending or restricting the existing data types.

The EXPRESS language supports several types of inheritance relationships. The following example shows one type of inheritance relationship supported by EXPRESS:

ENTITY *Point3D*
SUBTYPE OF (Point);
z : numeric_parameter;
END_ENTITY*;*

The Entity "*Point3D"* will have three attributes: "*x"* and "*y"* which are *inherited* from the Entity "*Point"*, which was presented at the top of this page, and "*z"*, which is declared locally.

Overall mentioned functions of EXPRESS, facilitates the development of the feature library by means of clear and hierarchical definition it provides for the manufacturing features in STEP documentation.

3.2.1.1) Manufacturing Features Geometry Data

Manufacturing features geometry data contains the information necessary to identify shapes, which represent volumes of materials that shall be removed from a part by machining or shall result from machining.

To extract rotational part manufacturing features with their attributes from the STEP AP224 documentation, to make the features a member of the feature library, to bring feature definitions up to a state that can be used in the dynamic link library (dll) file and most generally to construct the manufacturing features and part geometry data component of the feature definitions library the following preliminary work has to be done for each feature in the feature library:

1. Selecting and extracting the feature from the standard, STEP AP 224 documentation, with its definitions and attributes.
2. Referring to related EXPRESS documentation in order to find out inheritance structure of the feature.
3. Combining the collected data in order to completely define the geometry of the feature.
4. Referring to the STEP AP 238 documentation, if there are lacking attributes during the definition of the geometry of the feature,
5. Creating a 2D profile that will result as the 3D feature geometry after being revolved or extruded along or around an axis, after collecting all the necessary parameters to completely define the feature geometry,
6. Determining the generation technique for each feature that is, extrusion or revolution.
7. Determining the attachment Boolean operation for each feature that is subtraction, addition or intersection.
8. Determining the insertion point for each feature in FCS (Feature Coordinate System)
9. Determining the geometrical constraints that will be used
    a. in the definition of 2D profile,
    b. in the placement of 3D feature on the part

to avoid creation of meaningless geometries and possible designer errors that may occur during the use of feature modeler.

10. Creating test interfaces to determine all the necessary geometrical constraints and to test the constraints designed on paper and make trial designs to find out new constraints. According to the results of the trial designs made, modifying the geometrical constraints.

11. Documenting the:

- ✓ Selected STEP AP224 rotational part features,
- ✓ Hierarchical and object-oriented structure of selected features,
- ✓ Parameters necessary to completely define the 3D solid geometry for each feature including;
  - ➢ Attributes to define the 2D profile,
  - ➢ Geometrical definition of the 2D profile (points, driven parameters…),
  - ➢ Generation technique and attributes for 3D solid feature creation (extrusion or revolution),
- ✓ Geometrical constraints for each feature,
- ✓ Attachment technique (addition, subtraction, intersection) and insertion point for each feature,
- ✓ Geometrical constraints related to both 2D profile creation and feature interactions to provide the precision of design process.

Following each step stated above, "manufacturing features and part geometry data" component of the feature definitions library documentation is prepared. Resulting documentation stands as the building block for the dynamic link library, thus for the proposed feature modeler and covers features and their subtypes extracted from STEP AP224 for rotational parts and their definitions, classifications, attributes, generation and attachment techniques, in detail. Since, the resulting work and documentation adds up to a huge amount, it will only be demonstrated on one of the selected manufacturing features, in the next chapter, Chapter 4, to illustrate the developed methodology clearly. In the following part of this chapter, a brief summary about the results of the documentation will be presented. Manufacturing

features for rotational parts will be classified into its subtypes and each feature that is a member of the feature library will be defined in a few words. 2D sketches of features, their geometrical attributes, generation techniques and insertion points of each feature in the library, which is a part of feature dentitions library documentation, will be presented in the figures included in Appendix B. While defining the features the corresponding figure for that feature will be referenced to the Appendix B.

A *manufacturing_feature* identifies the types of features necessary to manufacture a machined rotational part. Each *manufacturing_feature* is either a *machining_feature* or a *transition_feature*. A *machining_feature* is a type of *manufacturing_feature* that identifies a volume of material that shall be removed to obtain the final part geometry from the initial stock. *Machining_features* requires both direction and location in placing them on a part. Each *machining_feature* may be one of the following: *outer_round*, *spherical_cap*, *revolved_feature*, or a *multi_axis_feature*. A *transition_feature* is a type of *manufacturing_feature* that is a transition area between two surfaces. This feature differs from *machining_feature* objects in that it requires no orientation for placement instead a feature or two features are selected to attach the transition feature to a position that has no other alternative. Each *transition_feature* is either an *edge_round*, *fillet*, or a *chamfer*. Below, types of *machining_features* and *transition_features* will be defined respectively.

An *outer_round* is a type of *machining_feature* that is an outline or significant shape that is swept through a complete revolution about an axis. Each *outer_round* is either an *outer_diameter* or an *outer_diameter_to_shoulder*. The *outer_diameter* is a subtype of *outer_round* and may have a constant diameter around the axis of rotation that is *straight_outer_diameter*, or it may be tapered that is *tapered_outer_diameter*. *Straight_outer_diameter* is a subtype of *outer_diameter* that is not tapered. *Tapered_outer_diameter* is a subtype of *outer_diameter* that describes a continual transition from one diameter to another diameter across a certain *feature_length*. An *outer_diameter_to_shoulder* is a subtype of *outer_round* that is a sweeping of a shape one complete revolution about an axis. The shape shall

be specified by two lines that connect at a point and extend finitely defined by diameters or lengths. 2D sketches, geometrical attributes, generation techniques and insertion points for the types of *outer_round* feature are presented in the Appendix B, in Figures B.1 to B.4.

A *spherical_cap* is a type of *machining_feature* that is circular about an axis of rotation. A *spherical_cap* consists of all points a given distance from a point constituting its center. 2D sketch, geometrical attributes, generation technique and insertion point for the *spherical_cap* feature is presented in the Appendix B, in Figure B.5.

A *revolved_feature* is a type of *machining_feature* that is a sweeping of a planar shape one complete revolution about an axis. Each *revolved_feature* is one of the following: *revolved_flat*, *revolved_round* or a *groove*. A *revolved_flat* is a subtype of *revolved_feature* that is the sweeping of a straight line about an axis. A *revolved_round* is a subtype of *revolved_feature* that is the sweeping of an arc about an axis. The *groove* is a type of *revolved_feature* that is a narrow channel or depression that is swept through one complete revolution about an axis. *Grooves* are classified into *square_u_groove*, *rounded_u_groove*, *partial_circular_groove*, *tee_ groove*, *and vee_groove* depending on their sweep shape. The *groove* feature may be defined on different faces of a part depending on the orientation of the profile and material side as shown in Figure 3.3 and *groove* can be further classified as *outer_groove* and *inner_groove*, accordingly. At the end of these two level classifications a *groove* may be one of, *inner_square_u_groove*, *outer_square_u_ groove, inner_rounded_u_groove*, *outer_rounded_u_groove, inner_partial_circular _groove*, *outer_partial_circular_groove, inner_tee_groove*, *outer_tee_groove, inner_vee_groove* or *outer _vee_groove*. 2D sketches, geometrical attributes, generation techniques and insertion points for these types of *revolved_ feature* are presented in the Appendix B, in Figures B.6 to B.17.

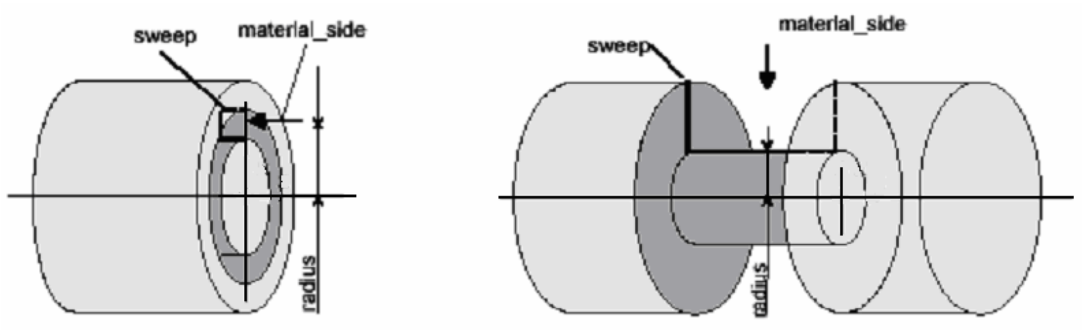

Figure 3.3 Inner_groove (left) and Outer_groove (right)

A *multi_axis_feature* is a type of *machining_feature* that identifies milling features for rotational parts. A *hole* is a type of *multi_axis_feature* that is the removal of a cylindrical volume from a part. Each *hole* is either a *round_hole*, *counterbore_hole*, or a *countersunk_hole*. A *round_hole* is a type of *hole* that is a removal of a volume of cylindrical shape from a part, which is represented by a *circular_closed_profile* swept along a *linear_path.* A *counterbore_hole* is a type of *hole* that is a combination of two round_holes. The first *round_hole* shall have either a *through_bottom_condition* or *a blind_bottom_ condition*; the second shall have a *blind_bottom_condition*, and a larger diameter than the first *round_hole*. A *countersunk_hole* is a type of *hole* that is a combination of two round_holes. The first *round_hole* shall have either a *through_bottom_condition* or *a blind_bottom_ condition* and it is not tapered; the second shall have a *blind_bottom_condition*, and it is tapered. Firstly, depending on the *bottom_condition*, *holes* are classified into subtypes, and then each subtype is further classified depending on their *change_in_diameter*. Each *bottom_condition* may be one of the following: *blind_bottom_condition* or *through_bottom_condition.* Each *blind_bottom_ condition* is either a *flat_hole_bottom*, *flat_with_radius_hole_bottom*, *flat_with_ taper_hole_bottom*, *conical_hole_bottom*, or a *spherical_hole_bottom*. 2D sketches, geometrical attributes, generation techniques and insertion points for the resulting types, after the two level classifications, of *holes* are presented in the Appendix B, in Figures B.18 to B.47.

A *slot* is a type of *multi_axis_feature* that is a channel or depression with continuous direction of travel. Firstly, depending on *the course_of_travel* the *slot* follows, *slot* is classified into subtypes such as *linear_slot* and *circular_slot*, secondly these two subtypes are classified first depending on open_profile, and then each is further classified depending on the *first_end_type* and *second_end_type.* The subtypes are listed and defined in the following pages. *Linear_slot* is a subtype of *slot*, which has *linear_path* as *course_of_travel.* A *linear_path* is a direction of travel along a line. Depending on the open_profile, *linear_slot* is either a *square_ linear_slot*, *square_u_linear_slot*, *round_u_linear_slot*, *partial_circular_ linear_ slot*, *tee_linear _slot*, or *vee_linear_slot*. *Circular_slot* is a subtype of *slot*, which has *circular_path* as *course_of_travel.* Depending on the open_profile, *circular_ slot* is also classified in the same way like the *linear_slot*. All *slot* types other than the *square_u_linear_slot*, have *open_slot_end_type* at both sides of the *slot*. Depending on *first_end_type* and *second_end_type*, different possible *square_linear _slot* subtypes are summarized in Table 3.1. 2D sketches, geometrical attributes, generation techniques and insertion points for the resulting types, at the end of classifications, of *slots* are presented in the Appendix B, in Figures B.48 to B.66.

Table 3.1 Square_linear_slot subtypes depending on end_condition

First_end_type	***Second_end_type***	***Feature(Subtype) Name***
open_slot_end_type	*open_slot_end_type*	*square_linear_slot_type1*
open_slot_end_type	*flat_slot_end_type*	*square_linear_slot_type2*
open_slot_end_type	*radiused_slot_end_type*	*square_linear_slot_type3*
open_slot_end_type	*woodruff_slot_end_type*	*square_linear_slot_type4*
flat_slot_end_type	*open_slot_end_type*	*square_linear_slot_type5*
flat_slot_end_type	*flat_slot_end_type*	*square_linear_slot_type6*
flat_slot_end_type	*radiused_slot_end_type*	*square_linear_slot_type7*
radiused_slot_end_type	*open_slot_end_type*	*square_linear_slot_type8*
radiused_slot_end_type	*flat_slot_end_type*	*square_linear_slot_type9*
radiused_slot_end_type	*radiused_slot_end_type*	*square_linear_slot_type10*
woodruff_slot_end_type	*open_slot_end_type*	*square_linear_slot_type11*
woodruff_slot_end_type	*woodruff_slot_end_type*	*square_linear_slot_type12*

An *edge_round* is a type of *transition_feature* that is a convex circular arc transition between two intersecting surfaces. The blend surface is tangent to both of the adjacent surface edges. Depending on the *edge_round_feature* and outer contour of the feature, different possible *edge_round* subtypes are summarized in Table 3.2. 2D sketches, geometrical attributes, generation techniques and insertion points for the resulting types, shown in the Table 3.2, of *edge_rounds* are presented in the Appendix B, in Figures B.67 to B.71.

Table 3.2 *Edge_round* subtypes

Edge_round_feature	**Outer Contour**	**Feature (Subtype) Name**
Straight_outer_diameter	Both	*Edge_round_type1*
Tapered_outer_diameter (decr. diameter)	Right	*Edge_round_type2*
Tapered_outer_diameter (decr. diameter)	Left	*Edge_round_type3*
Tapered_outer_diameter (incr. diameter)	Right	*Edge_round_type4*
Tapered_outer_diameter (incr. diameter)	Left	*Edge_round_type5*
Outer_diameter_to_shoulder	Right	*Edge_round_type6*
Outer_diameter_to_shoulder	Left	*Edge_round_type7*
Revolved_flat (decr. diameter)	Right	*Edge_round_type2*
Revolved_flat (decr. diameter)	Left	*Edge_round_type3*
Revolved_flat (increasing diameter)	Right	*Edge_round_type4*
Revolved_flat (increasing diameter)	Left	*Edge_round_type5*

A *fillet* is a type of *transition_feature* that is a concave circular arc transition between two intersecting surfaces. The blend surface may or may not be tangent to both of the adjacent surface edges. Firstly, depending on the answer of the question, "Does *fillet* require additional manufacturing operation or does it result from the geometry of the tool?"; *fillet* is classified into subtypes *m_fillet* and *g_fillet*. Secondly these two subtypes are classified first depending on *first_feature* and *second_feature*, and then each is further classified depending on whether blend surface is tangent to both of the adjacent surface edges or not. The results of this

classification is summarized in Table 3.3 and Table 3.4. Instead of every *tapered_outer_diameter, outer_diameter_to_shoulder or revolved_ flat* can be written in the tables. 2D sketches, geometrical attributes, generation techniques and insertion points for the resulting types, shown in the Table 3.3 and Table 3.4, of *fillets* are presented in the Appendix B, in Figures B.72 to B.87.

Table 3.3 *M_fillet* subtypes

First_feature	**Second_feature**	**Blend Surface**	**Feature (Subtype) Name**
Straight_outer_diameter	*Tapered_outer_diameter*	Tangent	*m_fillet_type1*
Straight_outer_diameter	*Tapered_outer_diameter*	Not Tangent	*m_fillet_type2*
Tapered_outer_diameter	*Straight_outer_diameter*	Tangent	*m_fillet_type3*
Tapered_outer_diameter	*Straight_outer_diameter*	Not Tangent	*m_fillet_type4*
Tapered_outer_diameter	*Tapered_outer_diameter*	Tangent	*m_fillet_type5*
Tapered_outer_diameter	*Tapered_outer_diameter*	Not Tangent	*m_fillet_type6*

Table 3.4 *G_fillet* subtypes

First_feature	**Second_feature**	**Blend Surface**	**Feature (Subtype) Name**
Straight_outer_diameter	*Straight_outer_diameter*	Tangent (+)	*g_fillet_type1*
Straight_outer_diameter	*Straight_outer_diameter*	Not Tangent	*g_fillet_type2*
Straight_outer_diameter	*Straight_outer_diameter*	Tangent (-)	*g_fillet_type3*
Straight_outer_diameter	*Straight_outer_diameter*	Not Tangent	*g_fillet_type4*
Straight_outer_diameter	*Tapered_outer_diameter*	Tangent	*g_fillet_type5*
Straight_outer_diameter	*Tapered_outer_diameter*	Not Tangent	*g_fillet_type6*
Tapered_outer_diameter	*Straight_outer_diameter*	Tangent	*g_fillet_type7*
Tapered_outer_diameter	*Straight_outer_diameter*	Not Tangent	*g_fillet_type8*
Tapered_outer_diameter	*Tapered_outer_diameter*	Tangent	*g_fillet_type9*
Tapered_outer_diameter	*Tapered_outer_diameter*	Not Tangent	*g_fillet_type10*

3.2.1.2) Tolerance Data

Tolerance data component of the feature definitions library defines the data of the tolerances information for a part specified by the STEP AP224. Tolerance data consists of two types: dimensional tolerances and geometrical tolerances. Types of tolerance data is shown in Figure 3.4.

1. *Dimensional tolerance* is the total amount a specific dimension permitted to vary, which is the difference between maximum and minimum permitted limits of the size.
2. *Geometrical tolerance* is the maximum or minimum variation from true geometric form or position that may be permitted in manufacturing. Geometric tolerance should be employed only for those requirements of a part critical to its functioning.

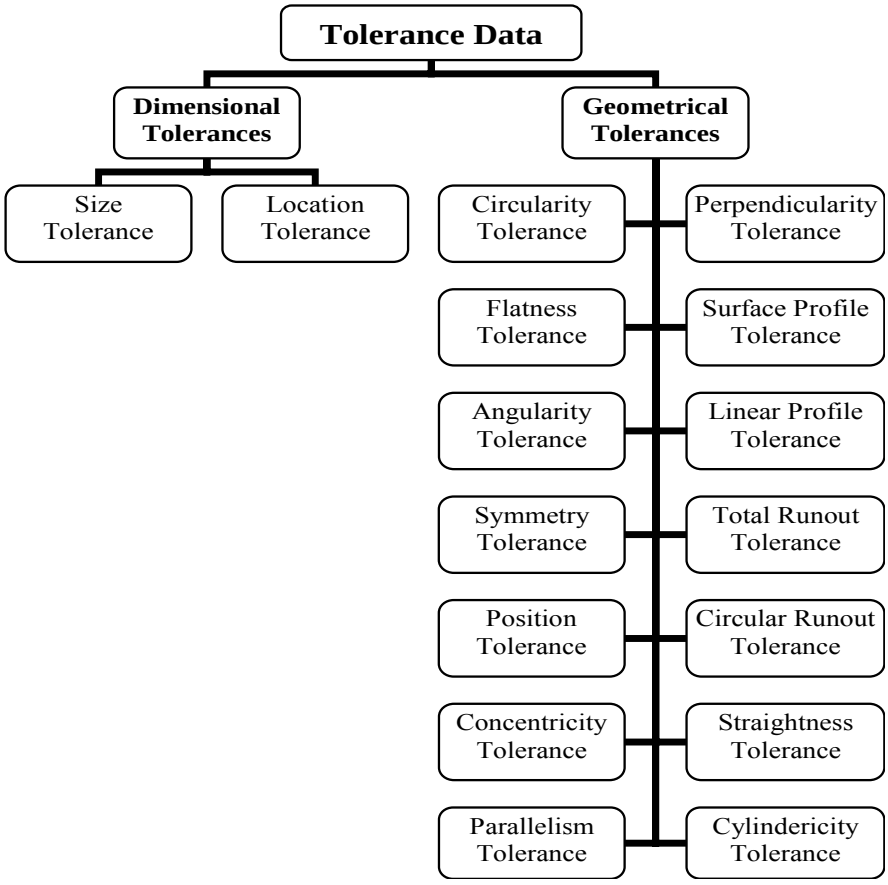

Figure 3.4 Types of Tolerance Data

3.2.1.3) Manufacturing Part Properties Data

The manufacturing part properties data component of the feature definitions library contains the description of characteristics of the rotational part that is being designed. These characteristics specify requirements of manufacturing that apply to, either the state of the part at a particular time prior or after the manufacture of the part, or a process that is required to be executed during the manufacture of the part. To fully define part information scheme and integrate it into the features definitions library a huge work has to be done, as it has been conducted for manufacturing features geometry data. It is also required to have a process planning strategy already developed in order to define an integrated part properties data library accordingly, at this stage. Therefore, within the scope of this study part properties data is just described in the sake of completeness of the definition of the feature library. To fully integrate the part properties data into the developed system, future work on this topic has to be conducted. As an example, besides many others some part characteristics covered in the standard as manufacturing part properties data is described below:

- ✓ *Material* is the identification of the raw stock from which a part is produced. Material identifies primary and substitution material. The material information is very important to be included in the part data that is resulting STEP-XML file. Because for process planning activities, it significantly affects the selection of the cutting tools, cutting parameters, etc…Material related data is defined by *material_id, material_description* and *stock_size* attributes in the standard. In addition, an alternate material with *alternate_ranking* and *material_substitute* attributes can be defined.
- ✓ The *surface information* related to representation of the surface of the part, such as surface finish, hardness and heat treatment conditions. These factors should be included in the part data that is resulting STEP-XML file. Because, while selecting manufacturing operations in process planning this data should be considered seriously. Hardness has *scale, high_value, low_value* and *nominal* attributes, surface property has *surface_finish* and *parameter_name* attributes to be defined in the standard.

3.2.2) Features Dynamic Link Library

Up to this point, Application Interpreted Model (AIM) and Application Reference Model (ARM) of STEP AP 224, has been covered and resulting feature definitions library has been developed for rotational parts. The important problem raised at this stage was, how the feature modeler will be tied to the EXPRESS information model of STEP AP 224 thus to the product data model developed in the feature definitions library. It was the question, which prompted the development of Feature Dynamic Link Library (Dll) in an object oriented manner. At the end, the product data model developed in the feature definitions library is implemented as an object oriented data type library, that is Features Dll, called "RotSTEPFeat.dll".

The following subsections describe the object oriented approach used in constructing the structure of the Features Dll, methodology developed to create Features Dll and capabilities of Features Dll respectively.

3.2.2.1) Object Oriented Structure of Features Dll

In general, object oriented concepts are accepted to be well suited to engineering activities because object structures are readily able to model the real world, support communication and provide interfacing and manipulation of different data types [25]. From the creation of feature modeler point of view, which is the main objective of this study, the object oriented representation gives the flexibility to define the system in a hierarchical manner, such that:

- Features in the feature definitions library can be represented as objects,
- Functions of the features in the feature definitions library can be represented as methods on objects such as drawing functions, export to STEP-XML functions…,
- The relationship between features in the feature definitions library can be represented as messages passed between objects,
- System architecture can be represented by the use of UML diagrams and class hierarchy.

By using the mentioned benefits gained using object oriented approach, manufacturing features defined in the feature library are transformed into solid objects, which will facilitate the implementation during the feature modeler development. In the following parts of this subsection, the general structure of the Features Dll and object oriented approach used in the development of the structure will be described.

Object oriented approach can be viewed as a software design methodology or programming style with particular disciplines. The basic principle of the object oriented approach used developing the Features Dll is, every element of the product data model environment (manufacturing features, tolerances defined in the feature definitions library for rotational parts) must be regarded as an object, which contains properties and methods. The properties of the objects stand for the attributes of the features (like attributes defining feature geometry) and methods of the objects stand for both the necessary actions that the features should perform (like creating 3D solids, storing tolerance information) and implemental behaviors of features (like exporting STEP XML file, returning necessary information about its definition). An object can only be accessed by activating one of the methods defined for that object.

In object oriented systems, by going one step further, classes are allowed to be defined in terms of other classes. Using this terminology within the Features Dll, although everything is considered as an object, objects are categorized as *"classes"*, *"instances", "superclasses"* or *"subclasses"*.

- ✓ A class is a generic software blueprint for one or more similar objects, such as *outer_diameter, groove, spherical_cap, hole* etc. in terms of manufacturing features. Objects are defined in terms of classes; it means that a lot is known about an object by knowing its class. Even what a "*tapered round hole*" is not known, if it is told that it is a member of "*hole*" class, it would be known that it had *diameter*, *hole depth* etc. as its attributes. A typical definition of a class object includes its class name, superclass, properties and methods.
- ✓ An instance is a "real" representation of an object, that is 3D CAD solid

representation for manufacturing features in the scope of this study, such as *tapered round hole with a conical bottom, outer square-u groove* etc., which has specific values for their definition. Class definition describes the location for specifying data storage and method. Two kinds of variables are supported: *class variables* and *instance variables*. The class variable is used to hold information shared by all instances of that class. (All *holes* have a *diameter* and *hole depth*) The instance variable contains information, which is specific to a particular instance. (A *tapered round hole with a conical bottom* would have a *bottom tip radius* and *bottom tip angl*e as its instance variable.)

- ✓ Superclass is the highest category that groups objects at the most generic level. Each subclass inherits properties and methods from its superclass. However, subclasses are not limited to the state and behaviors provided to them by their subclasses. Subclasses can add properties and methods to the ones they inherit from the superclass. The structure is not limited to just one level of inheritance. The inheritance tree, or class hierarchy, can be as deep as needed. Methods and properties are inherited down through the levels. In general, the farther down in the hierarchy a class appears, the more specialized its behavior. By this way, subclasses provide specialized behaviors form the basis of common elements provided by the superclass. Using inheritance, the code in the superclass can be used many times.

From this description of objects, the object oriented structure of the Features Dll can now be explained. First, the general view of the Features Dll will be presented, and then the four hierarchical levels will be explained. In most general means, using object oriented approach described and the data provided in the feature library has resulted the hierarchical UML structure shown in Figure 3.5 and Figure 3.6. Due to the extensive inheritance structure of the system and excessive number of classes, the entire system structure can not be presented in a UML diagram. Therefore, UML diagrams in Figure 3.5 and Figure 3.6 only show the uppermost structure in the hierarchy of features and tolerance classification. Notice that, each hierarchical structure shown in this subsection is validated through EXPRESS Schemas besides using the documentation provided by the features library.

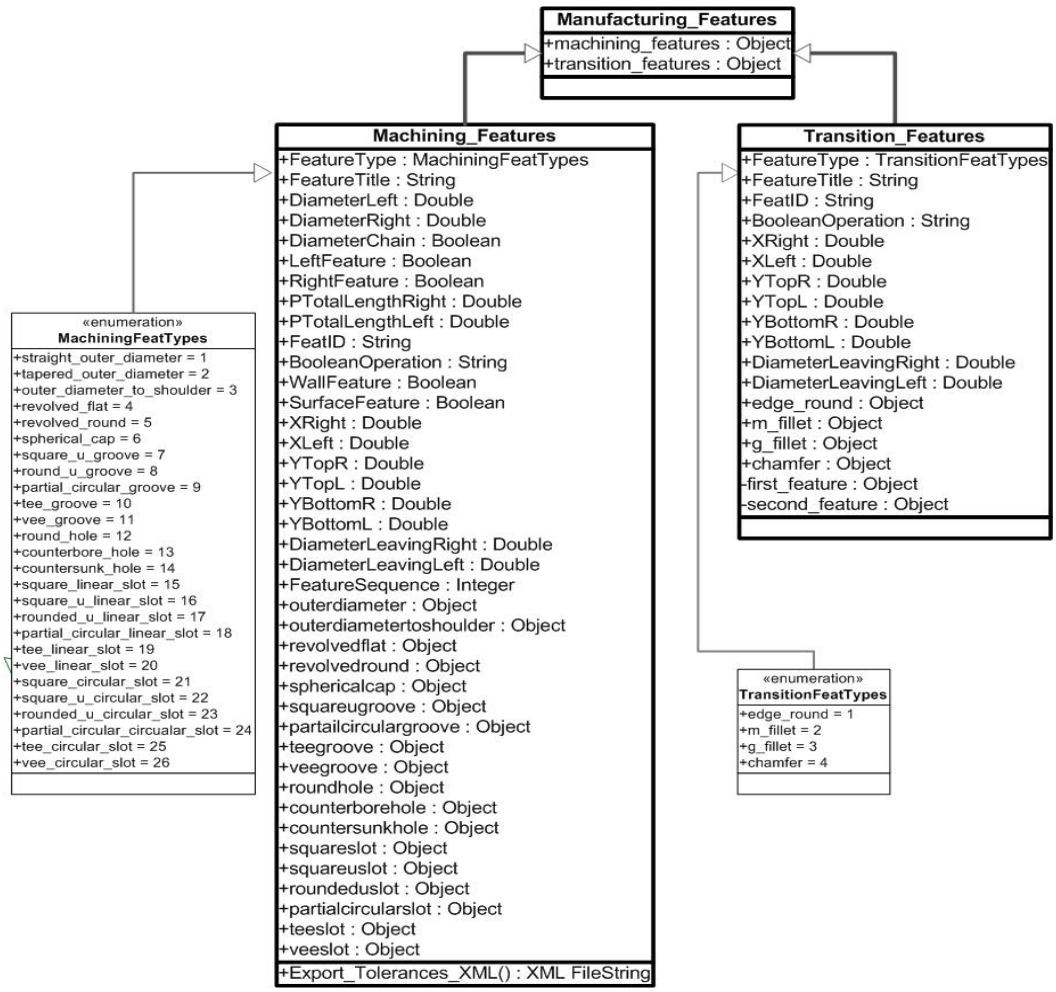

Figure 3.5 UML Diagram of Manufacturing Features for Rotational Parts

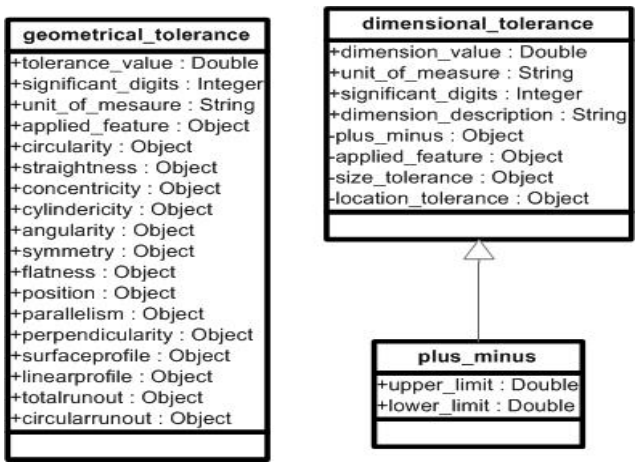

Figure 3.6 UML Diagram of Tolerance Data

UML diagram shown in Figure 3.5 exactly reflects the data provided in the manufacturing feature geometries data component of the feature library. In the diagram, *manufacturing_features* for rotational parts superclass takes its place in the highest category, the first hierarchical level. *Machining_features* and *transition_features* are subclasses of *manufacturing_ features* superclass. However, *machining_features* also act as superclass of many other subclasses such as o*uter_round, revolved_ feature, spherical_cap* and *multi_axis_feature*; *transition_features* act as superclass of *edge_round, fillet* and *chamfer*, in the second hierarchical level. The major difference of the whole system structure from the general one shown in Figure 3.5 is that some of the subclasses in the second hierarchical level (e.g. *outer_round* and *groove*) also act as superclasses of various subtypes of that features (e.g. *outer_diameter_to_shoulder* and *outer_groove),* in the third hierarchical level. Therefore, a further fourth hierarchical level classification is performed. The resulting classification in the third hierarchical level may lead to creation of instances or new superclasses. For instance while *outer_diameter _to_shoulder* is an instance in third hierarchical level , *outer_ groove* is a superclass of its subtypes, *outer_square_u_groove*, *outer_round_u_ groove*, *outer_partial_ circular_groove, outer_tee_groove* and *outer_vee_groove* in the fourth hierarchical level. Thus, for some classes the hierarchical level goes up to four, until the instances are created. This structure for rotational manufacturing features was defined in the manufacturing feature geometries data subsection of feature library section in this chapter. To avoid complexity, the detailed UML diagram, including the inheritance relation between classes, of only one selected feature will be presented in system development chapter, Chapter 4.

UML diagram shown in Figure 3.6 exactly reflects the data provided in the tolerance data component of the feature library. *Dimensional_ tolerances* and *geometrical_tolerances* superclasses take their place in the highest category. These are subclasses of many other tolerance types as shown in Figure 3.6 and defined in tolerance data subsection of feature library section in this chapter. The relation of tolerance classes with the *manufacturing_feature* subclasses will be shown in the example UML diagram provided in Chapter4, for a particular feature selected.

By using the terminology described, features dynamic link library (Dll) a file of code containing objects that can be called from other executable code (either an application or another dll) is developed. In general, Dll is used to provide code to be reused and to parcel out distinct jobs. Unlike an executable (exe) file, a dll file cannot be directly run. Features Dll must be called from other code that is already executing, that is feature modeler software in this case. Sample code containing the implementation and structure of one the most general class "*machining_features*" library is given in the Appendix A.1. This code consists of predefined classes of objects, which have been structured to form a hierarchical class library, which will be available within the feature modeler at run-time.

Generally, Features Dll is a predefined library of objects, which remains resident during a session of the feature modeler. The objects are class level objects as defined, including methods and variables applicable to the class, allowing instances of the object to be generated. In the objective of this study, class objects map directly to manufacturing features in the feature definitions library. The following subsection defines the methodology developed to create the features Dll.

3.2.2.2) Methodology to Create Features Dll

To create an integrated feature library for the feature modeler, for which the structure is described, a dynamic link library (dll) file should be created. The following are the steps required to generate the dll file:

1. Creating UML diagrams for each feature depending on the related EXPRESS documentation and feature definition, which includes the inheritance structure of the feature.
2. Validating UML diagrams through the EXPRESS Schemas in order to double check and avoid any errors.
3. Depending on the UML diagrams and the feature library documentation created, programming each feature by means of classes using Visual Basic 6.0 and the object oriented approach described.

4. Using Visual Basic 6.0 creating properties, methods and events for each feature class, which will help to;
   - ✓ assign values to feature attributes,
   - ✓ create 2D profiles from given points,
   - ✓ create 3D features from 2D profiles,
   - ✓ modify feature (for ex: rotation in necessary cases),
   - ✓ calculate, return required data about feature's geometry,
   - ✓ check geometrical constraints at each step and warn the designer about possible errors,
   - ✓ create STEP-XML file during creation of feature to append it to the resulting STEP-XML file of the designed rotational parts (the details of this process will be explained in the preprocessor section of this chapter),
   - ✓ store manufacturing features and part geometry, part properties and tolerance data.
5. Generating an algorithm for each feature that uses the EXPRESS class files as reference to generate the required STEP feature data format
6. Compiling "RotSTEPFeat.dll", dynamic link library file which will be used as the integrated feature library and the heart of the feature modeler in AutoCAD ActiveX Automation

The steps for generating the dll file have been followed for each feature and the "RotSTEPFeat.dll" file has been created.

3.2.2.3) Capabilities of Features Dll

- Creates 2D profiles from the points defined in feature definitions library and converts them into 2D regions,
- Creates 3D features from 2D regions created as the basic 3D entities of rotational parts in feature modeler,
- Has the ability to modify features if rotation or move functions are required in placement, movement and orientation depending on whether a feature is, right or left, or inner or outer,

- Returns geometrical information about feature geometry to facilitate error handling, which will be conducted to check feature interactions in the feature modeler,
- Provides features to store part geometry and all described types of tolerance data that designer will attach on it, which will upgrade them from being basic entities to high level entities,
- Checks for errors during the design process using the geometrical constraints defined in the feature definitions library,
- Facilitates creation of STEP-XML file by means of the algorithms developed and embedded in the dll for each feature, which will pioneer the development of the preprocessor,
- Most generally, integrates the feature library component of the overall system with the feature modeler component, playing the most critical bridging role throughout the system.

3.3) Feature Modeler

In previous sections, the first building block of the overall system, Feature Library, is described in detail by covering its components, Feature Definitions Library and Features Dll. When generalized, Feature Definitions Library is recognized to be the first fundamental component, which makes the overall system "STEP Based" and Features Dll is recognized to be the second fundamental component, which makes the overall system " Feature Based" and "Object Oriented", thus "Integrated". At this stage, after creating a complete and integrated feature library, the third fundamental component, which is the opening window of the system to the world, that means makes the overall system "Functional" and "Useful".

By feature modeler, what is meant is a software package, which is required to:

- cover as much manufacturing features as possible,
- enable the creation of 3D manufacturing features in AutoCAD 2000 design environment,
- bring them together facilitating the creation of the whole 3D rotational part,

- provide interfaces that are user friendly and isolates the designer from the complicated processes running behind the scene,
- permit attachment of tolerance attributes to features, presents an easy to design feature based design environment,
- avoids errors by means of comparing them with predefined cases,
- ensure that no feature interactions occur and no illogical or unmanufacturable parts can be designed
- warn the user by predicting the possible errors occurred during the design process,
- offer as various design alternatives as possible,
- present a continuous design environment by offering the designer to chain the diameters between succeeding features,
- provide flexible design environment for rotational parts by providing them both left hand and right hand feature attachment opportunity.

With the aim of developing a feature modeler, which is able to satisfy all the stated requirements, feature modeler architecture is developed and that architecture is implemented in AutoCAD 2000i environment by using MS Visual Basic for Applications, ActiveX Automation and error handling methods. The following sections provide insight to the feature modeler architecture and feature modeler implementation, respectively.

3.3.1) Feature Modeler Architecture

The feature modeler architecture is mainly organized in three major components:

1. Features Creation Phase,
2. Part Creation Phase,

and one mechanism working in between these components:

3. Error Handling.

The general architecture of the feature modeler is presented in Figure 3.7.

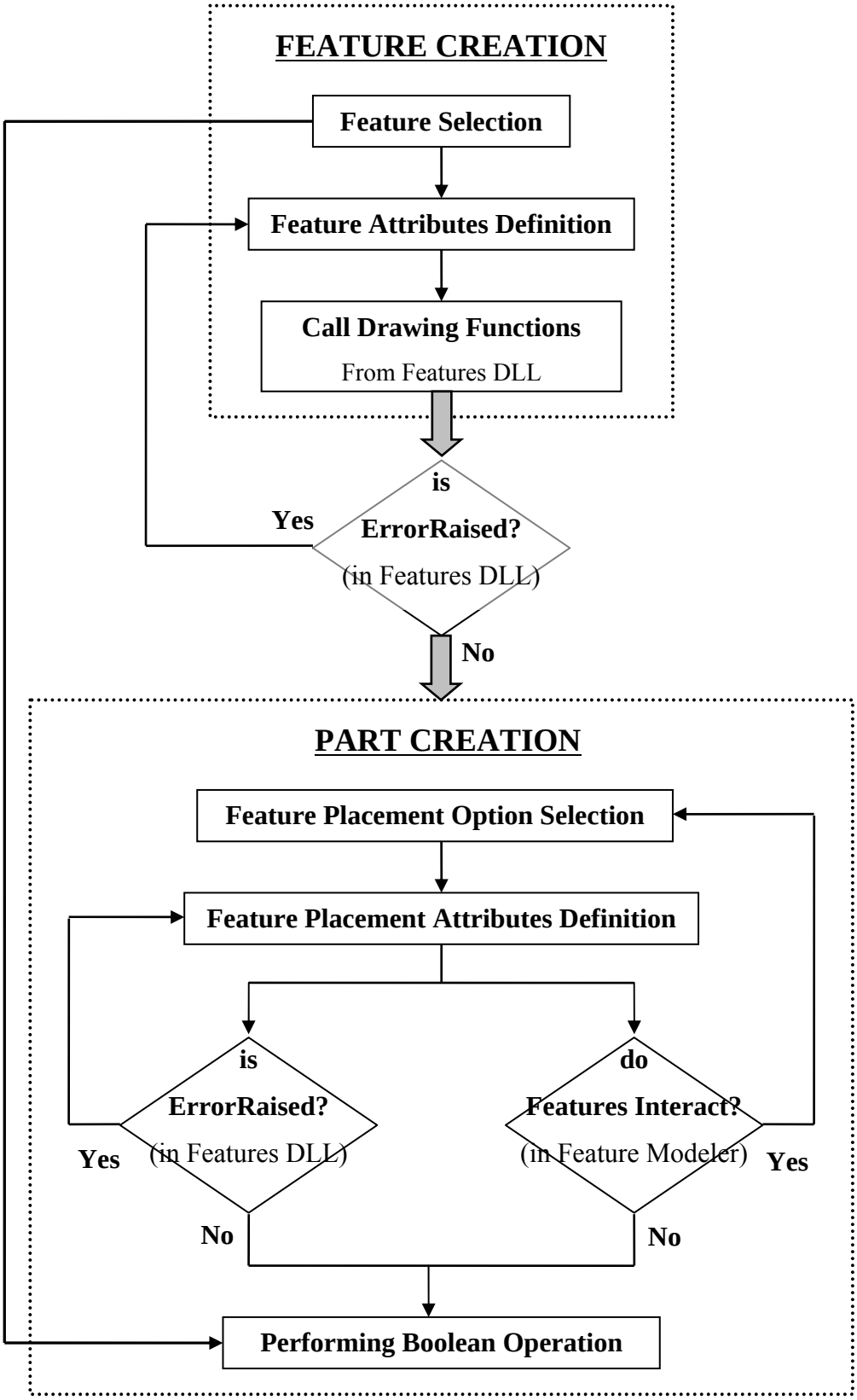

Figure 3.7 Feature Modeler Architecture

At first glance, feature creation is generation of manufacturing features one by one and part creation is bringing the generated manufacturing features together to form the resulting rotational part. In between these two stages and before the Boolean operation is performed, there are error handling mechanisms to justify the required reliable design in the feature modeler. Feature and part creation components of the feature modeler will be detailed in the following subsections. The working principle of the error handling used will be explained also in the following subsections.

3.3.1.1) Feature Creation

As it is shown in Figure 3.7, three steps are required to create a feature, namely: (1) Feature Selection, (2) Feature Attributes Definition and (3) Call Drawing Functions from Features Dll

Feature Selection: Feature selection is done by pointing to the required feature from a pull-down menu, which is already generated in the feature modeler's AutoCAD interface, by mouse or keyboard. A collapsed view of the pop-up menu is shown in Figure 3.8.

The selected feature corresponds to the matching class in Features Dll. By selecting a feature the corresponding Boolean operation, which will then be used while performing Boolean operation in part creation phase, related to that feature is also selected.

Feature Attributes Definition: By clicking on the required feature in the pull-down menu, a macro is executed and a pop-up form interface appears. An example pop-up form interface, for the feature *revolved_flat*, is shown in Figure 3.9. The appearing form includes input form elements like text boxes, combo boxes, check boxes and option buttons and frames grouping these elements. There exist also labels that inform the user about related actions and preview images, which illustrate the required parameters on a feature sketch, to guide and help the designer in selecting the right attributes.

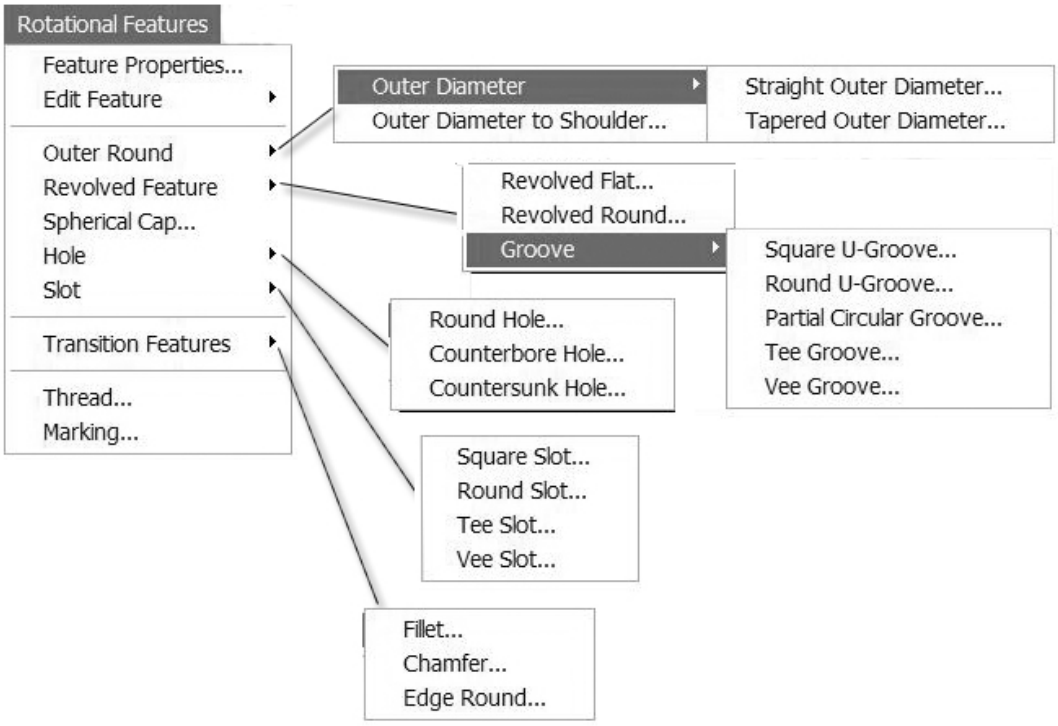

Figure 3.8 Exploded View of Pull-down Menu Designed in AutoCAD for Rotational Manufacturing Features

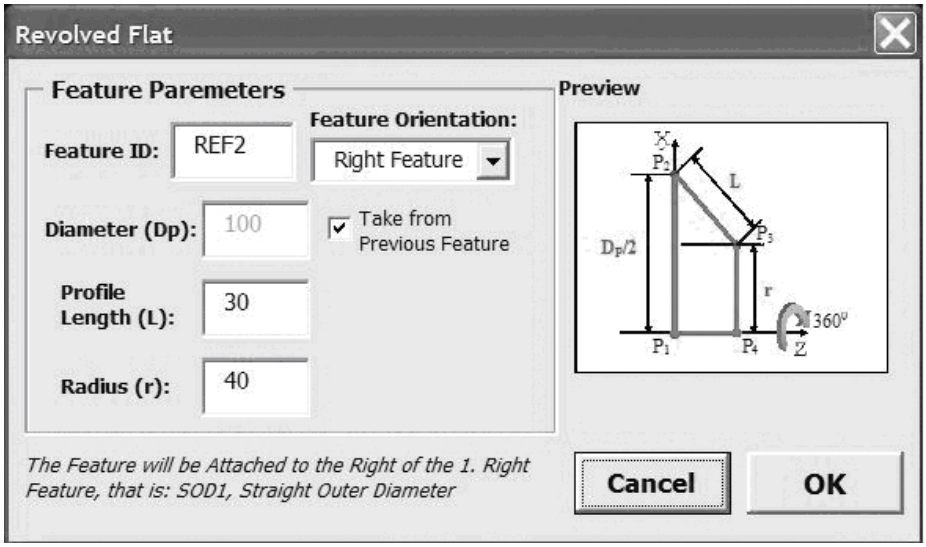

Figure 3.9 Example Pop-up Form Interface designed in AutoCAD for one of Rotational Manufacturing Features, *Revolved_flat*

Each input form element in the pop-up form stands for a property of the selected feature (class) necessary to completely define its geometry. The interface includes some default values for each attribute to give designer an idea about the possible values. The interface also offers some design alternatives for the attribute definitions, for example, taper condition may be selected by entering a taper angle or final diameter, which presents flexible design options to the designer. In the interface depending on these design alternatives, some input form elements may be enabled or disabled to prevent possible errors before they occur. There are command buttons to perform the required actions in the interface. Cancel Button closes the form and returns the process to the features selection phase.

After entering all the required parameters on the interface and clicking on the OK Button feature attributes definition phase is completed.

Call Drawing Functions: When the OK Button on the interface is clicked, the following process, behind the mouse click, is performed in the feature modeler respectively:

1. Feature attributes defined are set to the corresponding class properties, considering what the different design alternatives provide,
2. To create an instance of selected feature, the corresponding function, which creates the solid feature, is called from the Features Dll. The attributes, which are already set, are sent to the class in the Features Dll. They are processed and the feature is created.

As it was mentioned in Features Dll section, when one of its methods is called the corresponding object is activated and it creates a 2D region from the basic dimensions of feature, which are also sent. Then it creates a 3D feature solid by revolving or extruding the 2D region created, in AutoCAD' active document. However, if it finds any errors while checking each attribute with the predefined constraints it raises en error and sends an error description and an error message back to the feature modeler, which will take the whole process back to feature attributes definition phase.

3.3.1.2) Part Creation

Once the feature is generated, it has to be attached to the main rotational part. Part creation phase is composed of three main steps as it is shown in Figure 3.7, namely: (1) Feature Placement Option Selection, (2) Feature Placement Attributes Definition and (3) Performing Boolean Operation.

The manufacturing features for rotational parts were classified as *machining_ features* and *transition_features* in the features library section. At this stage, in the feature model a further classification of the *machining_features* into *parent_ machining_features and member_machining_features*, is performed to develop an effective method for part creation. *Machining_features*, which creates the outer contour of the rotational part, in other words which are protrusion features are defined as "*parent_machining_features*", these are *outer_diameter*, *outer_diameter _to_shoulder, revolved_flat, revolved_round* and *spherical_cap.* The other *machining_features*, which are subtraction features, in other words, which should be subtracted from the *parent_machining_features*, are defined as "*member_ machining_features*"; these are *grooves*, *holes* and *slots.* However, this should not be taken as a classification in the feature library, which only depends on the STEP. This classification is made just to facilitate implementation; it is only a logical classification. While simplifying part creation, this classification may limit the flexible design environment, for example a *hole* going through more than one feature was not allowed, since it can be a member of only one *parent_machining_ feature* at the design stage. These kinds of limitations are overcome by making it a member of the other *parent_machining_feature* it passes through in run time. *Transition_features* are treated as they are defined in the feature library. They may be protrusion or subtraction feature depending on their definitions in the feature definitions library. *Parent_machining_features*, *member_machining_features* and *transition_features* show differences in three steps of part creation.

In this subsection, three steps of part creation and how differently the types of different manufacturing features are handled in these steps, will be described.

Feature Placement Option Selection: Feature modeler provides different feature placement options to different types of manufacturing features. Providing placement options simplifies the design process, prevents errors and makes the feature modeler more flexible.

For *parent_machining_features*, feature placement options provided are selecting the feature to be left feature or right feature and taking its starting diameter from previous feature or not. Left or right feature selection is made through combo boxes placed in the interface and the diameter of the previous feature is automatically offered to the designer as its starting diameter. These options can be seen in the example pop-up form shown in Figure 3.9. For *member_machining_features*, feature placement option provided is selecting the *parent_machining_feature* on which the feature will be placed (if the *member_machining_features* will be placed to wall surface of the *parent_machining_feature,* left or right wall surfaces are also provided as options). For *transition_features*, feature placement option provided is selecting the *parent_machining_feature* on which the feature will be placed (if the *transition_features* is a *fillet*, two *parent_machining_features* between which the *fillet* will be placed are provided as options). In last two cases, the possible alternative *parent_machining_features* are automatically loaded to the design interfaces and the designer just makes a selection among them. This intelligent loading process examines the whole part and offers the designer every possible option for placement, for example, *inner_grooves* may be placed to the wall surfaces left between two *parent_machining_features* having different diameter values at the transition. By selecting one of the defined options, the feature placement option selection, step is completed.

Feature Placement Attributes Definition: Feature placement attributes are the attributes needed to exactly place the feature to the desired position on the rotational part. Feature modeler provides different feature placement attributes to different types of manufacturing features.

After selecting the feature placement options, for *parent_machining_features* and

transition_features there is no need to define any additional attributes to correctly place the feature on the part. Because the positions for placement are already known, such that the insertion point of the *parent_machining_features* is moved the center of the final diameter of the previous *parent_machining_feature* and the insertion point of *transition_ features* is moved to the point of transition. However, for *member_machining_ features*, definition of the feature placement attributes with respect to the *parent_machining_feature* selected may be required. It is "may be" because while this definition is not required for *holes,* for *grooves* and *slots* it is required to find out the distances and the orientation with respect to Feature Coordinate System (FCS) of the *parent_machining_feature.* These are the feature placement attributes and when they are defined according to FCS, feature insertion point is moved to the calculated placement point to complete the attachment process. To correctly place the feature on the part, feature placement attributes should be defined in the required cases and then feature placement attributes definitions step is completed.

Performing Boolean Operation: As soon as the feature insertion point is moved to the placement point, after the first steps are completed, Boolean operation is performed at that intersecting point, provided that the feature does not interact with any other previously created feature on the part or the feature dimensions does not exceed the part dimensions. Boolean operation is either addition or subtraction depending on the manufacturing feature type selected. Boolean operation type for each feature is defined in feature definition library and when the feature is first selected in the feature creation phase; Boolean operation information is stored to be used at this step. Performing the Boolean operation, this step and part creation phase is completed.

Feature creation and part creation phases are performed simultaneously for each feature provided that the designer is shielded from the explained complexities and details of the system and the designer only interacts with the interface elements and resulting 3D rotational part.

3.3.1.3) Error Handling

In the feature modeler, there are two stages of error handling where error handling mechanisms are placed to track for possible errors, as it is shown in the feature modeler architecture Figure 3.7. The first stage is performed in the Features Dll, which is associated with the geometrical constraints of the feature, defined in the features definitions library. When feature modeler calls drawing functions, it sends feature geometrical attributes to the Features Dll as class properties, Feature Dll verify this data by checking it against the feature geometrical constraints, predefined "if" statements checking attributes according to geometrical definitions. If it passes from this verification, then feature can be delivered to the part creation phase. However, if this verification fails, means if the feature attributes are incorrectly defined, Features Dll raises a public error and sends this error back to the feature modeler attaching the error description in an error message. Error descriptions are created so that the designer can be informed about the possible problem. The feature modeler sets every variable, property etc. back to its initial state when it receives the public error, to avoid any errors because of cached definitions and the overall process goes back to feature attributes definition step.

The second stage of error handling is performed before the Boolean operation. It has two levels of error tracking mechanisms. The first level is again performed by the Features Dll, verify feature placement attributes by checking the created feature geometry against part geometry this time, for example if the feature exceeds the boundaries of the part etc. If the feature passes from this verification, then the Boolean operation is performed. However, if this verification fails, means if the feature exceeds the part, Features Dll raises a public error and sends this error back to the feature modeler attaching the error description in an error message and the overall process goes back to feature placement attributes definition step. The second level is performed by the feature modeler this time; the feature modeler verifies the feature by checking its rightmost, leftmost, topmost and bottommost coordinates against the same coordinates of the previously created features to ensure that no feature interactions occur. If the feature passes from this verification, then the

Boolean operation is performed. However, if this verification fails, means if there is feature interaction, feature modeler raises an error and pops up an error message including the error description and the overall process goes back to feature placement options selection step. Some of the example error messages are shown in Figure 3.10.

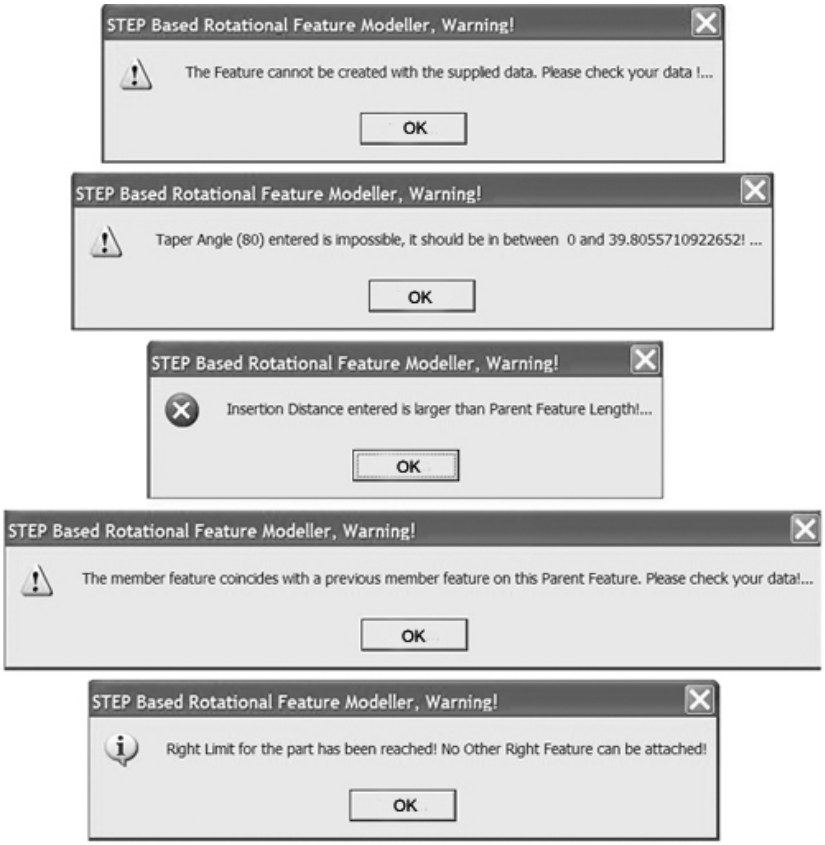

Figure 3.10 Example Error Messages

There also other error tracking mechanisms placed in the feature modeler. For example, if rightmost or leftmost diameter of the rotational part becomes zero, feature modeler pops up an error with description and does not allow the user to add more features at that direction or if there are no proper placement options for *member_machining_features,* feature modeler warns the designer about this condition and does not allow that features form to appear.

3.3.2) Feature Modeler Implementation

To implement the proposed feature modeler in AutoCAD 2000i environment, Visual Basic for Applications and ActiveX technology is used. By this way, AutoCAD became capable of providing "feature based design of rotational parts based on STEP". To achieve this goal following steps have to be fulfilled:

1. Uninstalling the AutoCAD default menus, toolbars, and creating, on purpose, AutoCAD menus including the pull-down menu shown in Figure 3.8
2. Designing the user interfaces in AutoCAD environment, using Visual Basic for Applications for AutoCAD, like the pop-up form interface shown in Figure 3.9. They will provide different design alternatives, to the designer for each feature and they will provide designers an easy to design tool for defining necessary feature attributes.
3. Making the feature modeler, able to design complex rotational parts in a reliable manner, using features in the feature library, by means of programming necessary feature creation, part creation and placement program in the AutoCAD environment and placing necessary error tracking and handling mechanisms at defined levels. Developing this program satisfying the requirements and covering the capabilities defined in the three phases of feature modeler architecture.
4. Integrating the tolerance data related interfaces and code within the "feature modeler for prismatic parts" developed in METU CIMLAB, by Saleh Amaitik [26, 27, 8], to the feature modeler for rotational parts.

These steps to create the feature modeler are followed and the feature modeler is created within the scope of this study. In feature modeler architecture section, details about first step, second step and error tracking and handling mechanisms have been provided. In the following part of this section, the general methodology used to develop the program mentioned in the third step and general messaging protocols used to achieve the exchange of data throughout the system will be explained, a more detailed picture of the developed program in the feature modeler will be drawn.

There are two types of messages used in the feature modeler. The first one is between the pull-down menus and pop-up forms, and the second one is in between the interface and the features Dll. The first type of messages is send by means of macros created in AutoCAD environment. Once a selection on the pull-down menu is made, a macro generating a message is executed and this message is send to public module in the program, which calls the show function of the corresponding pop-up form, and the form appears. The second types of bidirectional messages are sent by means of function calls, forward and error messages, backwards. The function call messages activate the classes in the Features Dll, and class properties are attached in these messages. Error messages sent from Features Dll to the feature modeler facilitate correct design and error descriptions are attached to these messages. The interface is so designed that it automatically generates the necessary messages protocol and perform to establish the required communication throughout the system, saving the designer from composing messages.

In the part creation subsection of the feature modeler architecture, it was mentioned that according to the general logic behind the rotational part design, manufacturing features are logically classified into three: *parent_machining_features*, *member_machining_features* and *transition_features* from the part creation point of view. *Member_machining_features* are accepted to the members of *parent_ machining_features.* To be able to transfer this "belonging" behavior" to the programming environment, *parent_machining_features* and *member_machining_ features* are defined in two dimensional arrays, like it is shown below: (assuming maximum number of twenty five right and left features and twenty member features on a parent feature)

Public MFeatures(-25 To 25, 0 To 20) As New Machining_Features

In this variable definition each *machining_feature* instance that will be created in the feature modeler is defined in terms of the *machining_feature* class in the feature library, in a two dimensional array. The first index of the array indicates the *parent_ machining_feature* and the second index indicates the *member_machining_feature.*

Parent features index has the same structure like in the line scale shown in Figure 3.11. At the starting point of the design process, if a the designer chooses the alternative of designing a right feature then the parent feature's index will be (1,0) in the array and the following right feature will have (2,0) as the index number. (vice versa is valid for left features that are (-1,0), (-2,0)…) If a member feature is attached on the first right feature created, member feature's index will be (1,1) and the following member feature on the same parent feature will have the index (1,2) (vice versa is valid for the left features (-1,1), (-1,2)…)

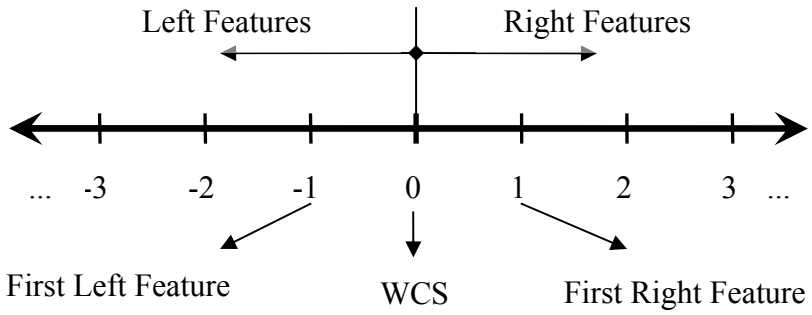

Figure 3.11 *Parent_machining_features* index definition

Transition_features are defined in an other array; the upper bound of the array is one less then the upper bound index of the array that indicates *parent_machining_ feature.* The first member of this array indicates the *transition_ feature* between the first and second right *parent_machining_features.* One dimensional *transition_ feature* array definition is shown below:

Public TFeatures(-24 To 24) As New Transition_Features

These array definitions for manufacturing features facilitate to store and exchange manufacturing features design information data throughout the system. Once the array elements are set with the corresponding features data, this data is stored in the array during run time, until the program is terminated or new part is started to be designed.

The described approach in developing the feature modeler provided a clear and natural way to organize data in the feature modeler. It also allows users to interact easily with the AutoCAD environment. Relating the feature modeler with the Features Dll and feature definitions library, the feature modeler gain the functionality to be “STEP Based”, “Feature Based” and “Object Oriented”, which improves its organization and clarity. By this way, the main objective of the thesis has also been achieved.

3.4) Preprocessor and STEP-XML File

The main goal of developing a preprocessor is to create self-descriptive, clear, reliable, neutral output file including all the manufacturing features used in the entire rotational part design, their placement location coordinates, feature types and their identifications, data defining feature geometry and additional attributes attached to the feature like tolerance data. While including all these data, to include absolute dimensions, especially for the placement coordinates, with respect to World Coordinate System (WCS) used in feature modeler, is important to facilitate the appropriate and efficient use of these dimensions in CAM or CAPP systems.

XML and EXPRESS, for which the basic concepts were described in Chapter 2 and in this chapter respectively, are now used as the fundamental technologies behind the preprocessor. The raising characteristics of these technologies and other standard technical data formats, in the scope of selection of a standard output file format, can be summarized as follows. EXPRESS is a very comprehensive language, when its inheritance and rule-based definition model is concerned. However, it is hard to learn and due to the extensive inheritance relationships between geometric entities, it is not fully implemental. Unlike languages such as XML, EXPRESS is rarely used in other domains [28]. This means, an output file directly in EXPRESS format is not feasible and not acceptable from the preprocessor objectives point of view. Therefore, it was decided that a new language was desirable as a standard output file format and EXPRESS Schema has been converted into that format. As an alterative, STEP’s Part 21 file format that

uses a style, which writes the information one at a time, avoiding the possibility of any contradictions in the data is selected. However, the style assumes that the data will only be processed by software that designers will only look at the data to create test examples or find bugs, and that making the data more easily readable by designers is less important than eliminating redundancies. This minimalist assumption contradicts with the design requirements of the output file style and it was popular before the advent of XML, which is created as a descriptive format for technical data that could be understood with less difficulty than other technical data formats. XML is a standard for describing data exchange files, nowadays. Since there is a semantic gap between EXPRESS and XML, simply converting EXPRESS data into XML data adds a lot of additional tags without making XML easier to be understood. Therefore, additional technologies have to be used to bridge this gap, like STEP Part 21. However, STEP Part 21 also puts many tags into XML definition. International recognition of this problem and the success of XML, resulted in the introduction of STEP Part 28, which best fits with the objectives of the preprocessor defined. Therefore, the preprocessor uses the STEP Part 28 standard, included recently into STEP, to map EXPRESS Schema into XML Schema and to create the STEP-XML output file.

In this section first, the structure of the XML Schema invented, which establishes the basic working methodology of the preprocessor and stands for the standard format of the STEP-XML output file, will be described. Then, how the processor creates the STEP-XML output file in the invented format will be explained.

To map EXPRESS Schema to STEP-XML output file, using the rules set in STEP Part 28, a mapping algorithm is developed. According to this simple mapping algorithm the main idea is, each element is the XML equivalent of an entity in EXPRESS. First owner elements and child elements are identified among the EXPRESS entities defining the feature, to establish a nesting relationship. Pick a tag name for each element from the available EXPRESS names. Finally, place the nested sequence rows in a logical manner, taking the nesting relationships created into account for each feature forming the entire rotational part.

Each STEP-XML file starts with a *STEP-XML* tag with an attribute defining the related STEP Part standard used while creating it. In the entire STEP-XML file, nothing can appear outside of this tag. Then file *file_schema* and *file_description* tags defining the output file are placed, respectively as shown below:

-<**STEP-XML** *xmlc="ISO 10303-28"*>

<**file_schema**>*feature_based_design_of_rotational_parts*</**file_schema**>

<**file_description**>*AP224 file*</**file_description**>

</**STEP-XML**>

Then, according to the STEP AP 224 EXPRESS schema used, the outermost element appearing after the STEP-XML tag will be the tag for one of the *machining _feature* or *transition_feature* elements. This means that every possible feature used in the feature modeler is one child element of one the owner elements, *machining_feature* or *transition_feature.* These element's tag names are selected so that they are the same with their EXPRESS entity names to ease understanding, this way of naming the tags are preferred throughout the entire STEP-XML output file. Inside one of these elements, placement coordinates, feature geometrical attributes definition and tolerance information elements are placed, respectively. Placement coordinates are the first appearing elements and they are placed in between *placement* tags, which act as an owner of the *location* element this time, with *location* tag. The only thing appears different from the traditional XML format, is the use of Object Serialization Early Binding (OSEB), a rule set defined in STEP Part 28. OSEB is used to simplify the STEP-XML output file by taking all the child elements of an element as its attributes inside the tag, and closing the tag without any tag name declaration. The features of STEP-XML file mentioned above are presented in a simple sample illustration part taken from the output file; the use of OSEB is shown for the element *location*:

-<**machining_feature**>

-<**placement**>

<**location** *x="0" y="0" z="50" />*

</**placement**>

</**machining_feature**>

At this stage, in the STEP-XML output file, elements mapping to the feature geometry attributes appear. These elements and their tags show differences for different features. Therefore, this part of the STEP-XML file will be illustrated on a real example in the next chapter, Chapter 4. The most general tags that can be seen at this stage are, a comment telling which feature is defined and a feature name tag describing the feature id as its attribute. A simple example for these general tags is shown below:

<!-- Round Hole Feature Definition -->
-<**round_hole** *id="RDH1"*>
</**round_hole**>

Lastly, elements mapping the tolerance data attached on the feature appear in the STEP-XML output file. The tolerance representation starts with an *its_tolerance* tag and continues with a comment explaining the type of the tolerance attached on the feature. Then according to the tolerance type attached on the feature, corresponding EXPRESS entities are mapped as tolerance elements. In the example below, *cylindricity_tolerance*, one of the *geometrical tolerances* and its child elements are shown:

-<**its_tolerance**>
<!-- Geometric Tolerance -->
-<**cylindricity_tolerance**>
<**significant_digits**>*1*</**significant_digits**>
<**unit_of_measure**>*Millimeter*</**unit_of_measure**>
<**tolerance_value**>*0.1*</**tolerance_value**>
</**cylindricity_tolerance**>
</**its_tolerance**>

The "-" signs shown in all of the examples before the tag names of owner elements means that they are expanded views of that element. When viewed in any of the browsers or editors, XML provides the ability to be displayed with color coded owner and child elements. A plus (+) or minus sign (-) to the left of the elements can be clicked to expand or collapse the element structure.

Once the STEP-XML output file structure is developed, the resulting rotational part design data should be translated into STEP-XML data, by means of a unidirectional preprocessor. In the preprocessor STEP-XML output file creation process, is carried out in two stages. First, features data related to its geometrical definition is translated into STEP-XML format, by analyzing each feature forming the resulting rotational part and translating their geometrical entities into the described format. Then, additional attributes, tolerance data, attached to the features are analyzed for each feature and translated into STEP-XML format. By translation, it is meant that mapping the EXPRESS format for each feature to the defined STEP-XML format. The following part of this section, describes the two stages of output file creation in a detailed manner

When the rotational part design in the feature modeler is completed and the designer selects the “Export STEP-XML AP224”, pull-down menu option placed under the “File Menu” in AutoCAD interface, output file creation process is triggered with a message sent to the Preprocessor by means of execution of a macro developed in the feature modeler. As soon as the preprocessor receives this message, it first creates the header part, describing the output file, of the STEP-XML file. Then, for each feature exists in the rotational part, it calls the corresponding “EXPORT_XML” function of that features, in the Features Dll. Preprocessor attaches the STEP-XML output file name selected by the designer and a tab position indicating the last position of the cursor in the nested sequence. As it was mentioned in previous sections, in Features Dll “EXPORT_XML” functions were created for each feature using the STEP-XML mapping algorithm defined in this section. At this stage, when the feature class in Features Dll is activated by calling one of its method, “EXPORT_XML” function, receiving the STEP-XML file name and tab position, opens the output file and appends the features geometrical data XML elements to the end of the file at its correct position in the nested sequence. A similar process is performed for the tolerance data. This time, a general “EXPORT_Tolerances_XML” called for each feature, it first examines if any tolerance data is attached on that feature or not. Then, for each type of tolerance data attached on the feature, it opens the output file and appends the

features tolerance data XML elements to the end of the file at its the correct position in the nested sequence. Once, this process is completed the STEP-XML output file, representing the same structure with the STEP AP224, is successfully created. A sample part of the developed code to implement the preprocessor is given in the Appendix A.2, including the two stages of messaging described in this section.

When a CAPP system, STEP-NC or G-Code generator need to be developed for rotational parts, the resulting STEP-XML file will be a highly appropriate input data for integration, which is self-describing and a programmer can parse it without the need for understanding EXPRESS. This way the data exchange, thus the integration between Feature Modeler, CAM and CAPP systems for rotational parts, has been facilitated.

CHAPTER 4

SYSTEM DEVELOPMENT

In the previous chapter, Chapter 3, a comprehensive system model has been presented. Each component, acting as a building block in constructing the entire system, is described in every detail, in a logical order. The architectures of the components, the methodologies developed to build up those components, the capabilities of the components, the benefits that the system gain through the existence, functions or outputs of those components and the developed implementation methods were the main topics in describing each component. Even though the system components are explained in every detail, while doing this in some cases the necessity of giving illustrative examples out from the work done, arouse. However, due to the extensive number of features and the sequential and inheritably conjunctive structure of the work done, for the sake of readability, understandability and completeness it is found more appropriate not to break the chain in the structure of the work done. Therefore, work done in developing the system is collected under this chapter to present it in the same order followed while the system components are described in Chapter 3. In addition, by this way it will be possible to come up a task list that guides through the system development, which means the sections of chapter also provides a checklist used during the development of the system.

In Chapter 3, at each section describing the methodology related to the illustrative material, presented in this chapter, Chapter 4 is referred to establish a relationship between the work done in system development and the corresponding methodology and functionality developed.

In this chapter, to explain the work done to develop the entire system, an example manufacturing feature for rotational parts, that is *outer_round*, will be selected and the sample work done for each feature, during the system development will be illustrated on that feature. To present the system development steps, this method is selected because most of the steps followed in the system development are repeating each other and to prevent the resulting document adding up to a huge amount. Therefore, projection of the total work done on *outer_round* will not cause any information lack for the sake of completeness. The following first two sections in this chapter will take the *outer_round* feature from just being a definition in STEP AP 224 documentation to becoming a superclass in the hierarchy. Then, the latter section makes it a member of the designed rotational part in the feature modeler. Lastly, it will be represented in STEP-XML output file.

4.1) Feature Definition

In previous sections, it was emphasized that the STEP AP224 is chosen as the best alternative product data model for the feature library, which will provide the standard definitions for feature based design in feature modeler. After this decision, Application Reference Model (ARM) and Application Interpreted Model (AIM) of the AP 224 are reviewed. This study lead to an understanding about, how the information located in the standard could be used to develop the system and how to track the information related to features throughout the system. Then, the comprehensive process of extracting features with their definitions from the standard is completed. Finally, the collected feature definitions data is documented in features definitions library, for each feature. A brief summary of the documentation about the feature classifications and their definitions is provided in Chapter 3. Also in Appendix B, feature geometries library is presented which is a part of the documentation created. However, it is certain that this documentation includes much more details for every feature. In the following subsections, the explained process will be illustrated on the example *outer_round* feature step by step.

4.1.1) STEP AP224 Definition

First, ARM representation of outer_round is found out in the STEP AP 224 documentation, as an application object. ARM representations lists application objects alphabetically, they are not listed feature by feature and application objects may be used to define the attributes of other application objects. Therefore, to understand the flow of information between application objects and identify the where the appropriate data is coming from or going to, AIM representations of each feature is referred at this stage. The EXPRESS-G Diagram, located in the AIM representation of STEP AP 224, for *outer_round* is shown in Figure 4.1 [19]. In the EXPRESS-G Diagram, it is first recognized that *outer_diameter_to_shoulder* and *outer_diameter* are subtypes of *outer_round* feature. *Diameter*, *feature_length* and *reduced_size* are the attributes defining the feature *outer_diameter*. It is also seen that, while *diameter* and *feature_length* are the attributes defined directly under the ARM description of *outer_diameter, reduced_size* attribute refers to another application object, "*taper_select*". Thus, the attributes defining the *taper_select* application object should also be included into the definition of *outer_diameter*. In the same way, *v_shape_boundary* and *diameter* are the attributes defining the feature *outer_diameter_to_shoulder.* While *diameter* attribute is defined directly under the ARM description of *outer_diameter_to_shoulder, v_shape_boundary* attribute refers to another application object, "*vee_profile*". Thus, the attributes defining the *vee_profile* application object should also be included to the definition of *outer_diameter_to_shoulder*. At the end applications objects, which are used to define the *outer_round*, are brought together with the help of associated EXPRESS-G Diagram, to completely understand its definition. By this way, all of the attributes defining *outer_round* are identified and defined.

The output of the work stated above, for *outer_round*, will be presented in the next subsection in the feature definitions library documentation, together with the additional attributes defined for it to facilitate programming in the later stages of system development.

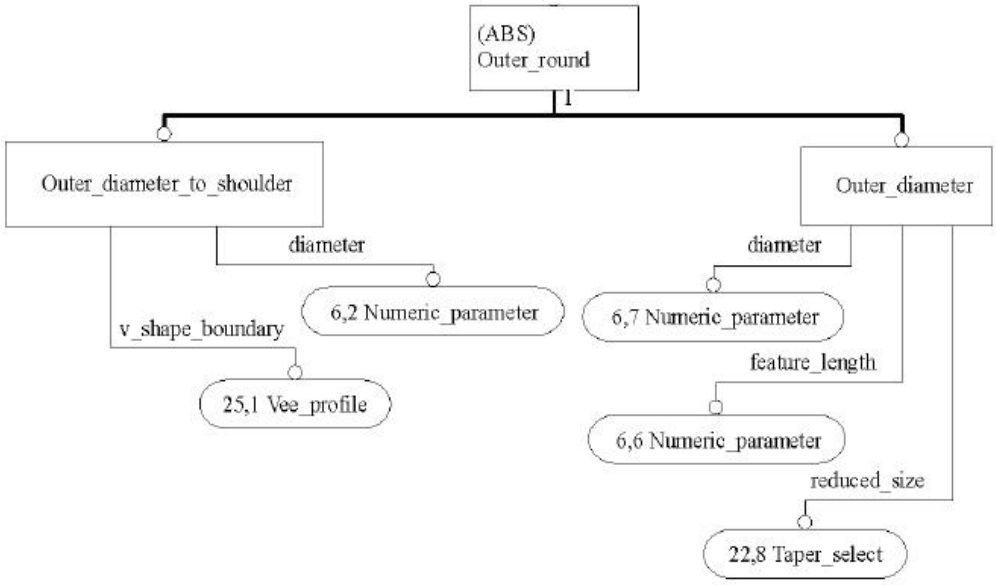

Figure 4.1 EXPRESS-G Diagram for the feature *outer_round*

4.1.2) Feature Library Definition

After discovering the STEP definition of the features with all of its attributes resulting from the inheriting structure of STEP AP 224, by including some additional attributes to the definition of the features and by identifying the EXPRESS Schemas for the features, feature definitions library documentation is prepared. In this subsection, as a part of the entire documentation, documentation for *outer_round* will be presented. By additional attributes, what is meant is, the attributes created for each feature to facilitate its representation and use during the development of Features Dll and Feature Modeler. Additional attributes include:

- 2D sketches with geometrical definitions attached to be able to create 2D regions in Features Dll,
- Generation techniques to create the 3D solid from 2D region in Features Dll,
- Insertion points to stand as the base point while feature placed on the part in Feature Modeler,
- Boolean operations to identify if the feature will be added to or subtracted from the part in Feature Modeler,

- Geometrical constraints to prove the reliable design environment in feature Modeler by means of providing bases for the error tracking and handling mechanisms placed throughout the system.

The following documentation harmonizes both the STEP definition and feature library definition of *outer_round*.

An *outer_round* is a type of *machining_feature* that is an outline or significant shape that is swept through a complete revolution about an axis. Each *outer_round* is either an *outer_diameter* or an *outer_diameter_to_shoulder*. These words can be translated into EXPRESS language as follows:

ENTITY outer_round;

ABSTRACT SUPERTYPE OF (ONEOF *(outer_diameter, outer_diameter _to _shoulder));*

SUBTYPE OF *(machining_feature);*

END_ENTITY; *-- Outer_round*

The *outer_diameter* is a subtype of *outer_round* that is a sweeping of an outline specified by a line segment one complete revolution about an axis. The line is finite in length and coplanar with the axis. The *outer_diameter* may have a constant diameter around the axis of rotation that is *straight_outer_diameter*, or it may be tapered that is *tapered_outer_diameter*. *Diameter*, *feature_length* and *reduced_size* are the parameters necessary to completely define the *outer_diameter* geometry. The STEP AP 224 ARM representation of *outer_diameter* feature is shown in Figure 4.2 [19]. The *diameter* (D) specifies the maximum diametric size of an *outer_ diameter*. The *feature_length* (L) specifies the size of an *outer_diameter* feature, measured along the feature's axis. *Reduced_size* can be selected between two possibilities. If a *diameter_taper* is selected, this is the diameter of the opposite side of the feature's placement co-ordinate system that is *final_diameter* (D_F). If an *angle_taper* is selected for describing the cone, this angle is the *taper_angle* (α) between the negative x-axis and the line on the positive y-side of the x-axis defined by the intersection of the cone with the xy-plane of the feature, extended to meet the

x-axis. An angle greater than 0 degrees and less than 90 degrees indicates a *tapered_ outer_diameter* with decreasing diameter for increasing x-values, an angle between 90 degrees and 180 degrees indicates a *tapered_outer_diameter* with increasing diameter for increasing x-values. The way of saying all these words in EXPRESS language is given below:

ENTITY *outer_diameter*

SUBTYPE OF *(outer_round);*

reduced_size: OPTIONAL taper_select;

feature_length: numeric_parameter;

diameter: numeric_parameter;

END_ENTITY*; -- Outer_diameter*

TYPE *taper_select =* ***SELECT*** *(angle_taper, diameter_taper);*

END_TYPE*; -- taper_select*

ENTITY *angle_taper;*

angle: numeric_parameter;

END_ENTITY*; -- Angle_taper*

ENTITY *diameter_taper;*

final_diameter: numeric_parameter;

END_ENTITY*; -- Diameter_taper*

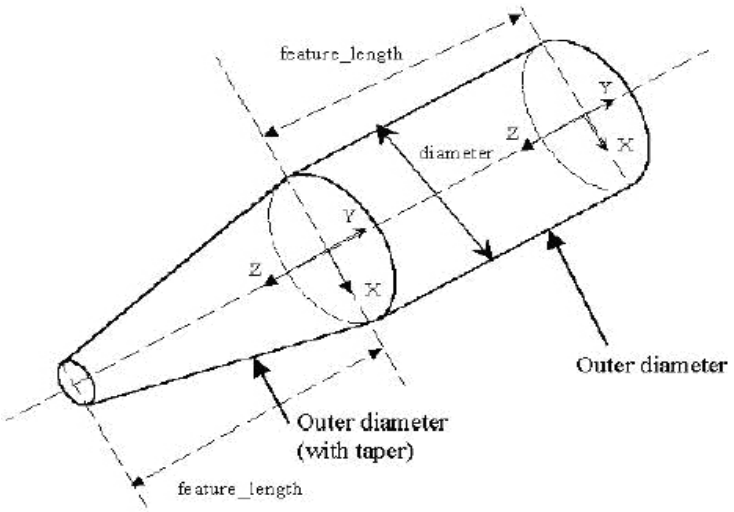

Figure 4.2 STEP AP 224 ARM Representation of outer_round

In 3D *outer_diameter* feature creation, a planar profile enclosed by points P_1, P_2, P_3, and P_4 is created and a feature volume is generated by revolution of the planar profile 360^0 about x-axis. Figure 4.3, Figure 4.4 and Figure 4.5 demonstrates the generation process for the different types of *outer_diameter* and the required parameters for generation. The resulting feature will be added to the part for all there types of *outer_diameter.*

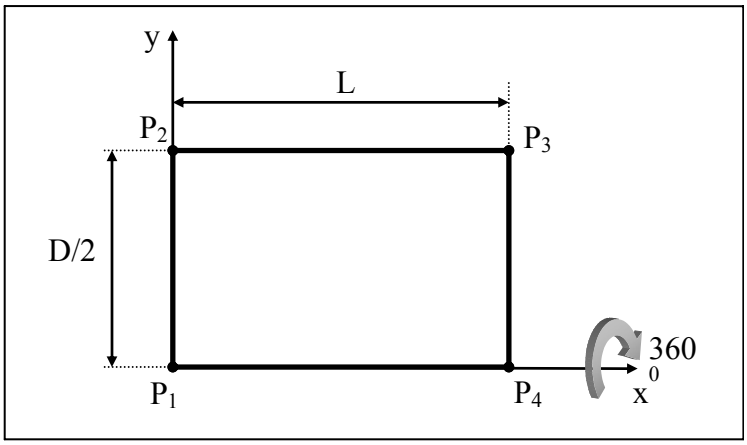

Figure 4.3 Straight_outer_diameter

Geometrical definitions for the *outer_diameter* feature in Figure 4.3:

P_1 (0, 0, 0) (insertion point)

P_2 (0, D/2, 0)

P_3 (L, D/2, 0)

P_4 (L, 0, 0)

Geometrical constraints for the *outer_diameter* feature in Figure 4.3:

$D > 0, L > 0$

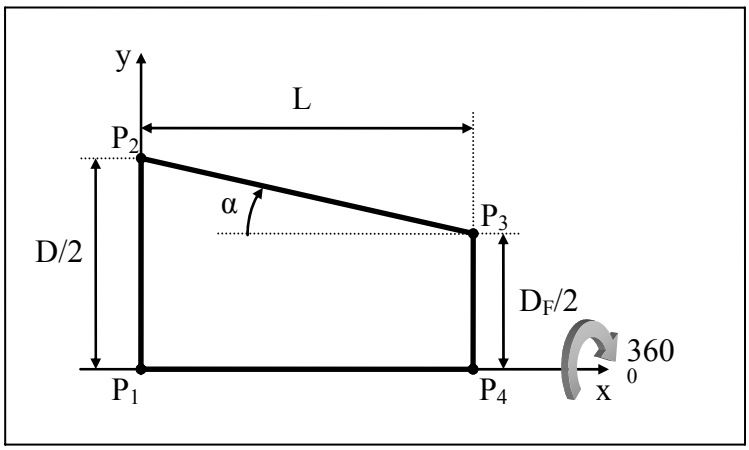

Figure 4.4 Tapered_outer_diameter, decreasing diameter

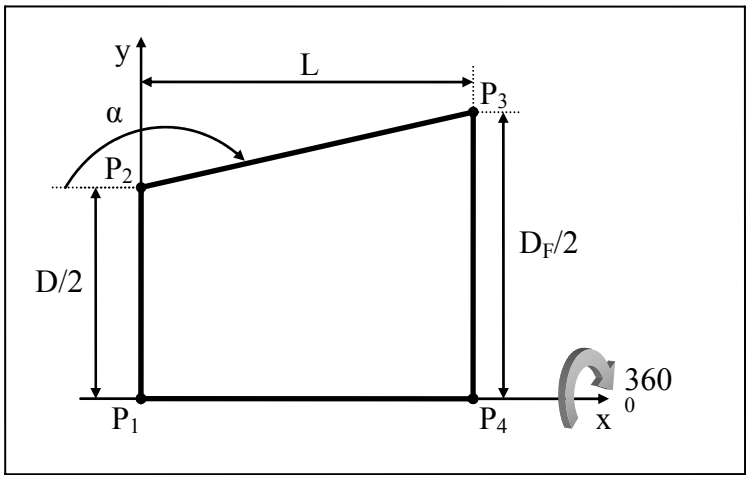

Figure 4.5 Tapered_outer_diameter, increasing diameter

Geometrical definitions for the *outer_diameter* feature in Figure 4.4 and Figure 4.5:

P_1 (0, 0, 0) (insertion point)

P_2 (0, D/2, 0)

P_3 (L, D_F/2, 0)

P_4 (L, 0, 0)

$$D_F = D - \left(2 \times \tan(\alpha) \times L\right)$$

$$\max(\alpha) = \arctan\left(\frac{D}{2 \times L}\right)$$

Geometrical constraints for the *outer_diameter* feature in Figure 4.4 and Figure 4.5:

$D > 0, L > 0$

$(90 < \alpha \leq 180^0)$, for increasing diameter case

$(0 \leq \alpha \leq \max(\alpha) < 90^0)$, for decreasing diameter case

An *outer_diameter_to_shoulder* is a subtype of *outer_round* that is a sweeping of a shape one complete revolution about an axis. The shape shall be specified by two lines that connect at a point and extend finitely defined by diameters or lengths. The enclosed angle shall be smaller than a straight angle. The intersection of the two lines need not be blended with a radius. *Vee_profile* specifies the *v_shape_ boundary* of the *outer_diameter_to_shoulder*. The STEP AP 224 ARM representation of *vee_profile* is shown in Figure 4.6 [19]. *Diameter* and attributes inherited from *vee_profile* those are, *profile_radius*, *tilt_angle*, *profile_angle* (optional) are the parameters necessary to completely define geometry. However, the definition for *vee_profile* is lacking information to completely define the *outer_diameter_to_shoulder* feature geometry. Thus, STEP AP 238 is referenced in this case, as it is mentioned in previous chapters. The STEP AP 238 ARM representation of *vee_profile* is shown in Figure 4.7. As it is seen from Figure 4.7, AP 238 adds *first_side_length* (*diameter_previous_feature*), *and second_side_ length* (*final_diameter*) attributes to the definition. These attributes are adapted to more feasible ones, for the designer to input without further calculations. Nevertheless, in the output file, the attributes are exported in the represented way. The *diameter* (D) specifies the size of the part at the point of the vee, or where the two sides come together, swept about an axis of rotation. The *profile_angle* (α) specifies the size of the angle between the two sides of the *vee_profile*. The angle shall be greater than 0 and not more than 180 degrees. The *profile_radius* (r) specifies the size of the blend radius at the point of the vee, or where the two sides come together (profile origin, I). The *tilt_angle* (β) specifies the size of the angle between one side of the *vee_profile* and the x-axis of the local coordinate system that defines the *vee_profile* orientation on the part. The *first_side_length* (L_1) indicates the distance, as measured from the profile origin, along the side of the vee

located by the sum of the *tilt_angle* and *profile_angle*. The *first_side_length* parameter can be calculated when the *diameter_previous_feature* (D_P) parameter is known. The *second_side_length* (L_2) indicates the distance, as measured from the profile origin, along the side of the vee located by the *tilt_angle* parameter. The *second_side_length* parameter can be calculated when the *final_diameter* (D_F) parameter is known.

Design using diameters or design using lengths are the two possible choices to complete the geometry definition of feature *outer_diameter_to_ shoulder*. The STEP AP 224 ARM representation of *outer_diameter_to_shoulder* feature is shown in Figure 4.8 [19]. The way of saying all these words in EXPRESS language is given below:

ENTITY *outer_diameter_to_shoulder*
    SUBTYPE OF *(outer_round);*
    diameter: numeric_parameter;
    v_shape_boundary: vee_profile;
END_ENTITY*; -- Outer_diameter_to_shoulder*

    ENTITY *vee_profile*
        SUBTYPE OF *(open_profile);*
        profile_radius: OPTIONAL numeric_parameter;
        tilt_angle: numeric_parameter;
        profile_angle: numeric_parameter;
        first_side_length: numeric_parameter;
        second_side_length: numeric_parameter;
    END_ENTITY*; -- Vee_profile*

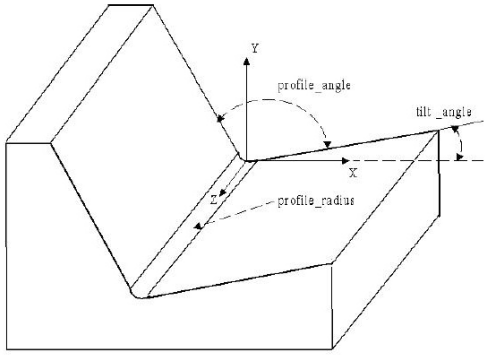

Figure 4.6 STEP AP 224 ARM Representation of Vee_profile

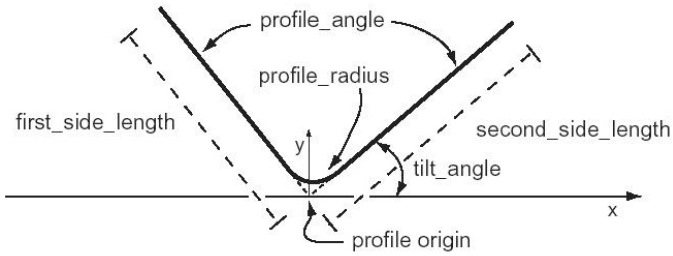

Figure 4.7 STEP AP 238 ARM Representation of Vee_profile

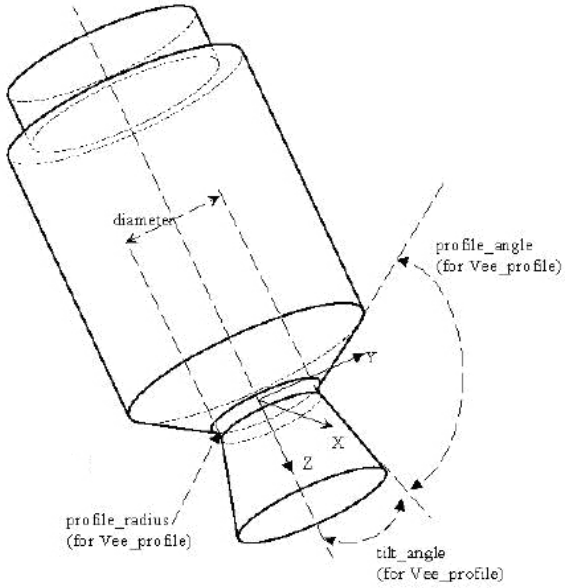

Figure 4.8 STEP AP 224 ARM Representation of Outer_diameter_to_shoulder

In 3D *outer_diameter_to_shoulder* feature creation, a planar profile enclosed by points P_1, P_2, P_3, P_4, P_5, P_6 and P_7 is created and feature volume is generated by revolution of the planar profile 360^0 about x-axis. Figure 4.9 demonstrates the generation process and the required parameters for generation. The resulting feature will be added to the part.

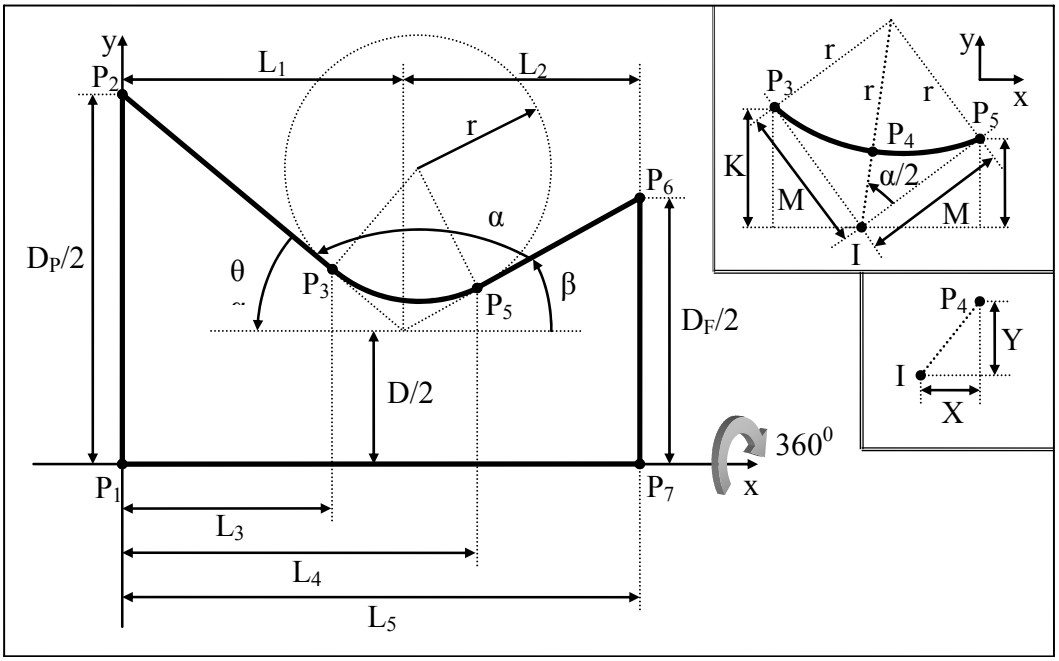

Figure 4.9 Outer_diameter_to_shoulder

Geometrical Definitions for the *outer_diameter_to_shoulder* feature in Figure 4.9:

P_1 (0, 0, 0) (insertion point)

P_2 (0, D_P/2, 0)

P_3 (L_3, D/2+K, 0)

P_4 (L_1 + X, D/2 + Y, 0)

P_5 (L_4, D/2 + N, 0)

P_6 (L_5, D_F/2, 0)

P_7 (L_5, 0, 0)

Case a: Design Using Diameters

$$L_1 = \frac{(D_P - D)}{-2 \times \tan(\alpha + \beta)}$$

$$L_5 = L_1 + \frac{(D_F - D)}{2 \times \tan(\beta)}$$

Case b: Design Using Lengths

$$D_P = D - 2 \times \tan(\alpha + \beta) \times L_1$$

$$D_F = D + 2 \times \tan(\beta) \times L_2$$

$$L_5 = L_1 + L_2$$

For Both Cases:

$$M = \frac{r}{\tan(\frac{\alpha}{2})}$$

$$N = M \times \sin(\beta)$$

$$K = M \times \sin(\alpha + \beta)$$

$$L_3 = L_1 + \frac{K}{\tan(\alpha + \beta)}$$

$$L_4 = L_1 + \frac{N}{\tan(\beta)}$$

$$Y = (\frac{r}{\sin(\frac{\alpha}{2})} - r) \times \sin(\frac{\alpha}{2} + \beta)$$

$$X = (\frac{r}{\sin(\frac{\alpha}{2})} - r) \times \cos(\frac{\alpha}{2} + \beta)$$

Geometrical Constraints for the *outer_diameter_to_shoulder* feature in Figure 4.9:

$$D > 0, L_1 > 0, L_2 > 0, r \geq 0$$

$$D_F > D, D_P > D$$

$$0 \leq \beta < 90^0,\ 0 \leq \theta < 90^0,\ 0 < \alpha < 180^0$$

$$\alpha + \beta + \theta = 180^0$$

$$0 < (\alpha + \beta) \leq 180^0$$

$$(180^0 - (\alpha + \beta)) < 90^0$$

$$0 \leq r \leq \left(\tan(\frac{\alpha}{2}) \times \min(|P_2 I|, |IP_6|) \right)$$

4.2) Feature Class Creation

To create the feature class with respect to the object oriented approach explained in Chapter 3, objects, their properties and methods, classes, superclasses and subclasses has to be identified out of the documentation presented in the previous section. As it was explained, the best way to do it is to create UML Diagrams, representing the inherited structure of the classes with properties and methods attached on them. For each feature, UML Diagrams are created to facilitate the class creation corresponding to that feature. In Figure 4.10, UML diagram for *outer_ round* is shown, which is created by using the documentation presented in the previous section, especially EXPRESS Schemas for *outer_round.* Some other properties and methods other than the ones in the EXPRESS schemas are also included in the UML Diagram, because they are required when the feature is placed on the part and when the STEP-XML output file is created.

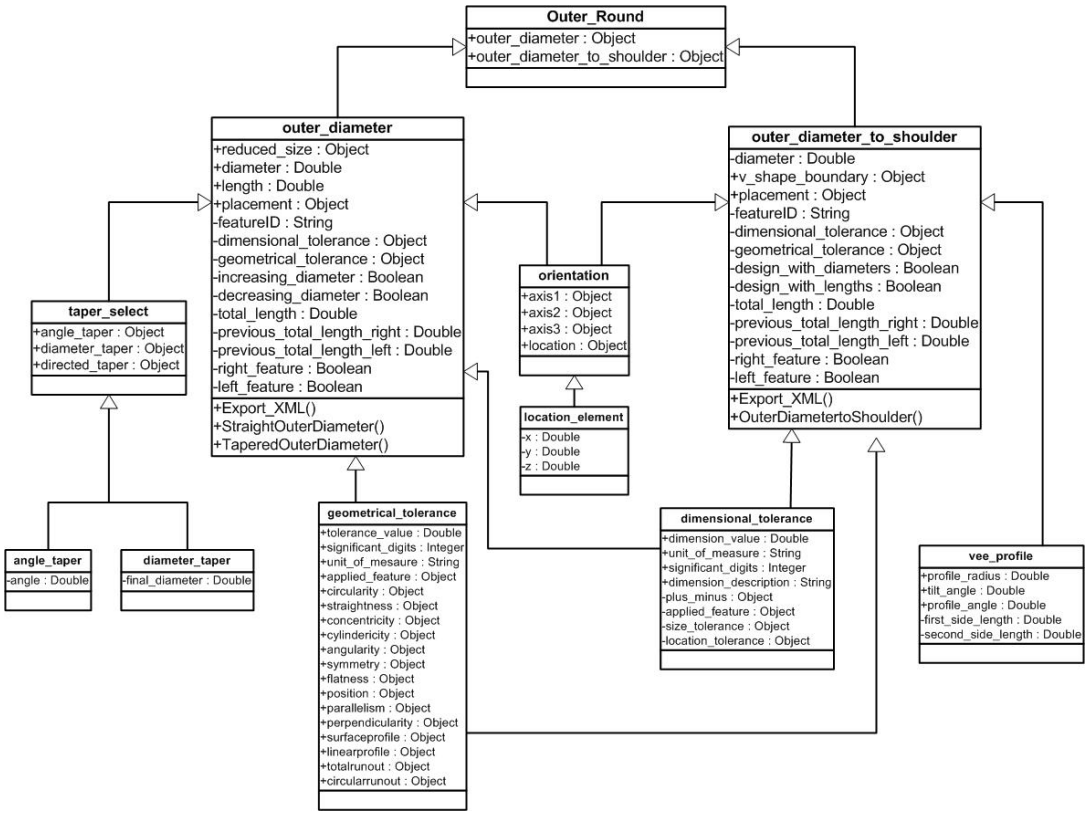

Figure 4.10 UML Diagram for the feature *outer_round*

The object oriented approach used in developing the Features Dll file and the functionality of the Features Dll file has been covered in every detail, in Chapter 3. That general methodology is valid for each feature in the features library. After carrying out the necessary steps explained in Chapter 3, *outer_diameter* and *outer_diameter_to_shoulder* classes are created, taking the presented UML Diagram as basis reference for this programming process. Sample code out of the *outer_ diameter* class in the Features Dll, including 2D feature region creation and 3D feature solid creation using geometrical definitions presented, is given in Appendix A.3. The created classes can be used throughout the entire system, especially from the feature modeler.

Figure 4.11 demonstrates the use of *outer_diameter* class in one of the system components. When a variable, "OuterDiameter", is defined in terms of the *outer_diameter* class, that variable gain access to the properties and methods of that class. Each time "OuterDiameter" variable *calls* any of the methods of the *outer_ diameter* class, the messaging protocol defined in Chapter 3, play its role and the necessary action is performed. This process provides the basic answer to the question, "How does the Features Dll and its classes are used from the feature modeler?".

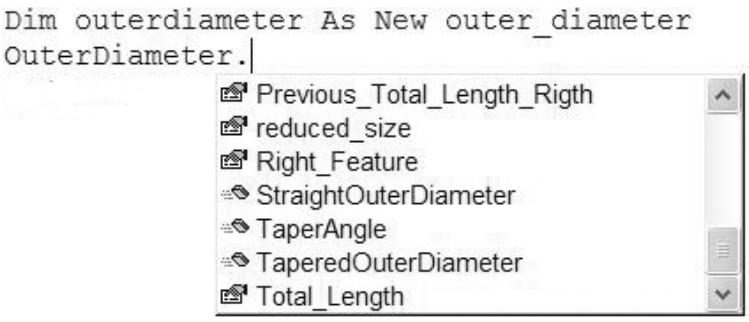

Figure 4.11 Outer_diameter class usage

4.3) Implementation in Feature Modeler

Feature modeler first creates the feature in World Coordinate System (WCS) of the active document in AutoCAD, by calling the associated drawing function of the feature class and sending the feature attributes defined by the designer. Then, the feature is automatically moved to the position according to the feature placement options selected and feature placement attributes defined by the user. This process is extensively defined in Chapter 3. In this section the creation of an instance, *tapered_outer_diameter,* feature will be demonstrated on an example run scenario, without going deep into the processes executing behind, which have already been described in Chapter 3.

The design process in the feature starts with the selection of the feature from the pull-down menu developed and installed into AutoCAD environment. Therefore, to create the *tapered_outer_diameter* feature, the pull-down menu selection shown in Figure 4.12 has to be done first. When this menu option is selected a macro, calling the pop-up form is executed. The macro is presented below:

Sub *tapered_outer_diameter0*()
   tapered_outer_diameter_form.Show
End Sub

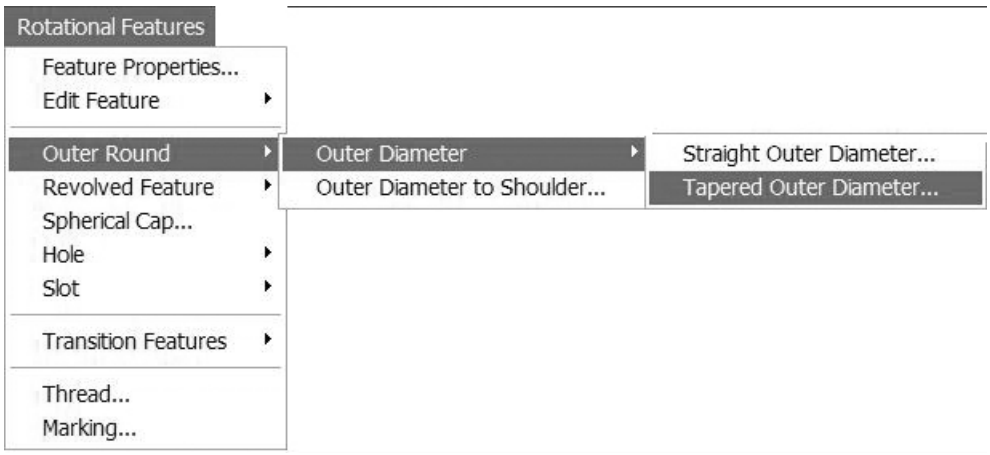

Figure 4.12 Pull-down Menu in Feature Modeler to Select *outer_round*

When called by the macro, *tapered_outer_diameter_form* appears, shown in Figure 4.14 and waits for designer input. The attributes defined for *tapered_outer_ diameter* in the feature definition section are all designed on the form interface. Additionally, attributes related to part creation are included. According to the logical classification done in the feature modeler section of Chapter 3, *tapered_ outer_diameter* is a *parent_machining_feature.* Therefore, placement options provided are to be a right/left feature and taking its starting diameter from previous feature or not. There is no need to define the placement attributes (these properties of *parent_machining_features* were explained in Chapter 3). When the designer inputs all the desired field and press "OK", the code embedded into the interface is activated. The feature modeler first sets the entered variables to the properties of *outer_diameter* class. One example is shown below:

MFeatures(FeatSequence, 0).OuterDiameter.feature_length = Val(Length)

How the MFeatures array is set to *machining_features* class and the meaning of the indexes of the array were described in Chapter 3. Here, *feature_length* property of the *outer_diameter* class is set to the value of the input in the "Length" textbox.

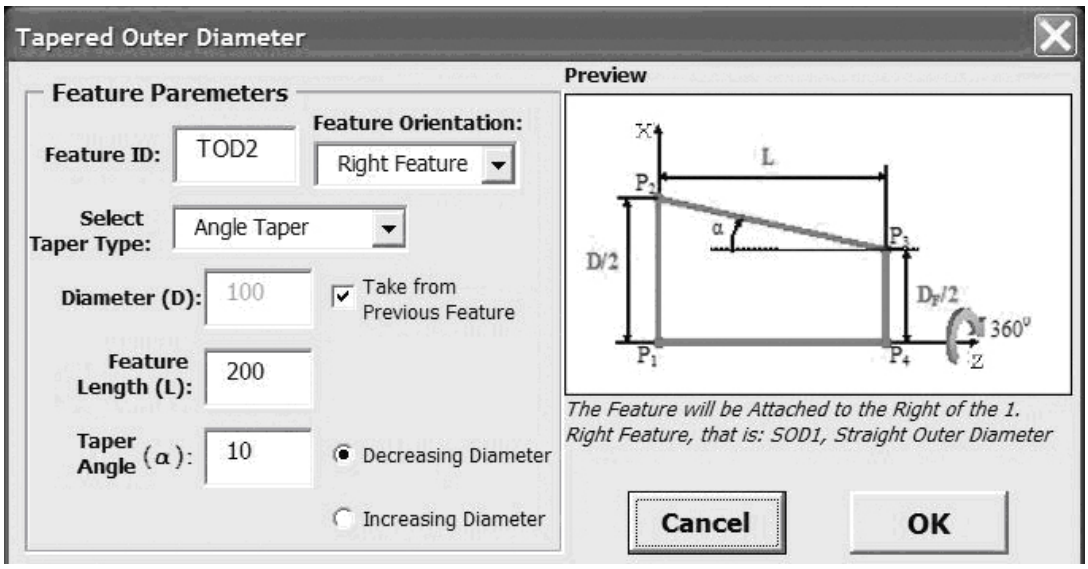

Figure 4.14 Pop-up Form Interface used to create *tapered_outer_diameter,* named *tapered_outer_diameter_form*

Setting all the properties of *outer_diameter* class, the feature modeler calls the *TaperedOuterDiameter* function of the *outer_diameter* class, which will first create the feature and then place it on the part according to the options selected. Below example shows how this function is called:

Set *CurrPFeat=MFeatures(FeatSequence,0).outerdiameter.TaperedOuterDiameter*

Set *FeatureSolids (FeatSequence, 0) = CurrentPFeat*

CurrPFeat (current parent function), which is an AutoCAD 3D solid object, is made to the store *tapered_outer_diameter*, by setting it to the function. Then, the feature is taken into an array into a 3D solid objects array, with a correct index. This will make the *tapered_outer_diameter* accessible whenever necessary.

By this way, the *tapered_outer_diameter* feature is created and placed on the part, by the feature modeler. Additionally, the data related to the feature is stored in arrays to be used in the creation of other features, first to calculate placement location automatically and then to prevent feature interactions.

4.4) Exporting STEP-XML

In Chapter 3, the format of STEP-XML output file, the structure of the preprocessor creating the STEP-XML file and the elements and their tags appearing in the output file has been described. However, elements mapping to the feature geometry was lacking. In this section a simple output file including the XML elements and their tags, which maps to the definition of the feature *outer_diameter* is shown. The sample STEP-XML output file is given below:

-<**STEP-XML** *xmlc="ISO 10303-28"*>

<**file_schema**>*feature_based_design_of_rotational_parts*</file_schema>

<**file_description**>*AP224 file*</file_description>

-<**machining_feature**>

-<**placement**>

<**location** *x="0" y="0" z="100"* />

</**placement**>

```
        <!--  Outer Diameter Feature Definition  -->
        -<outer_diameter id="SOD2">
            <feature_length length="200" />
            <diameter diameter="100" />
        </outer_diameter>
    </machining_feature>
</STEP-XML>
```

In this file, an *outer_diameter* with a feature ID, “SOD2” is represented. By looking at the file, it is easily understood that the feature is designed with a *feature_length* attribute value of “200 mm” and with a *diameter* value of “100 mm”. These definitions are placed in between the tags of the child elements of *outer_diameter* element, they are *feature_length* and *diameter*, as it was defined for a *straight_outer _diameter*. It is also recognized that, it is a right feature, but not the first right feature and placed “200 mm” away from the WCS. This information is placed in between the tags of the child element of *placement,* which is *location.*

CHAPTER 5

SAMPLE APPLICATIONS

In this chapter, two sample rotational parts, which are designed by using the developed feature modeler and some parts out from their correspondent STEP XML file generated using the developed preprocessor, will be presented. These two examples are so chosen that they enable to figure out the design capabilities of the feature modeler, how effective and rather complex rotational parts could be designed by means of using variety of features in an orderly manner. First sample rotational part consists of an outer (parent) manufacturing feature and a number of inner (member) manufacturing features and the second one consists of number of outer and inner manufacturing features, provided that most of the design alternatives that the feature modeler offers are covered. The STEP XML outputs for selected features in the part will provide an understanding about how the output file is organized and how to read the data in the output file. In the following two sections, sample parts and their output files will be presented.

5.1) First Sample

First sample provides an understanding about the wide variety of possible inner features and their orientations those can be used in rotational part design. In Figure 6.1 and Figure 6.2, dimensions necessary to design the rotational part in the feature modeler are given in two different views. These dimensions are given considering the feature geometric attributes and absolute placement attributes of features with respect to World Coordinate System (WCS). Placement attributes are given considering machining and tool aspects. In Figure 6.3, each feature constructing the overall sample part is labeled. The sample part is fully designed using the feature

modeler and the layout drawings are obtained using the functionality of AutoCAD, in which the feature modeler is embedded.

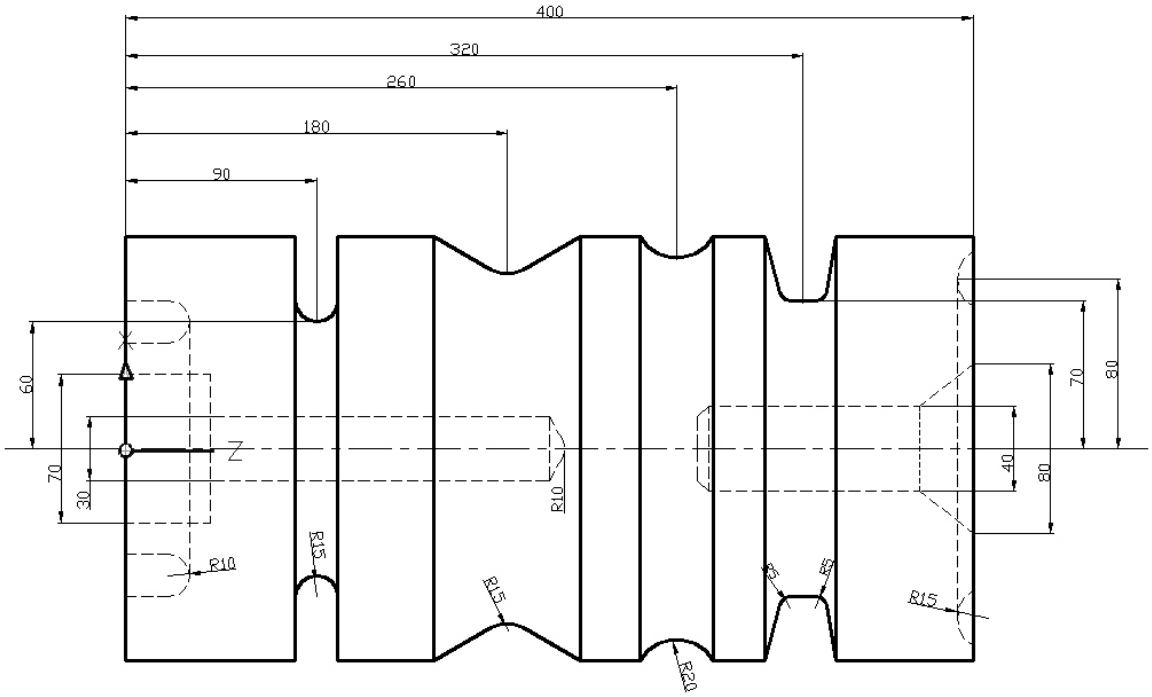

Figure 6.1 Dimensions to Design Sample Part 1 in Feature Modeler (front view)

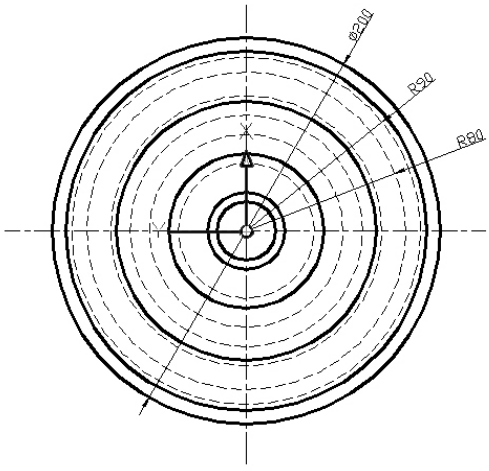

Figure 6.2 Dimensions to Design Sample Part 1 in Feature Modeler (right view)

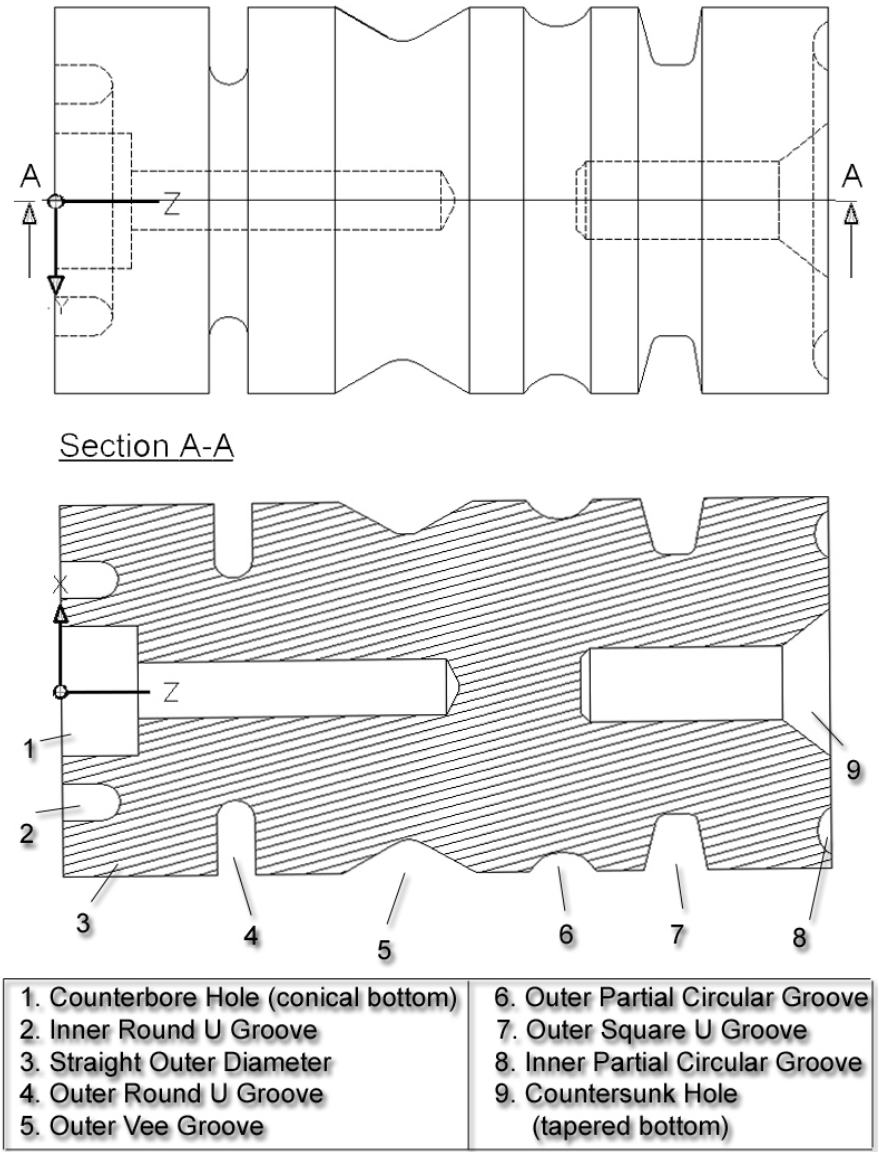

Figure 6.3 Top(upper) and Section(lower) Views of Sample Part 1

Figure 6.4 presents the 3D solid and isometric views of the sample part to provide a better understanding about its 3D geometry. As is it seen in the sample part figures, the rotational part designed may consist of an outer manufacturing feature and many inner features attached on it. Inner features may be outer or inner grooves those may have different types of profiles (like partial circular, tee, vee profiles...) or axial round, counterbore or countersunk holes those have a variety of bottom and taper conditions. The inner features may be attached to any orientation on the part, unless it interacts with a pre-designed feature on the part.

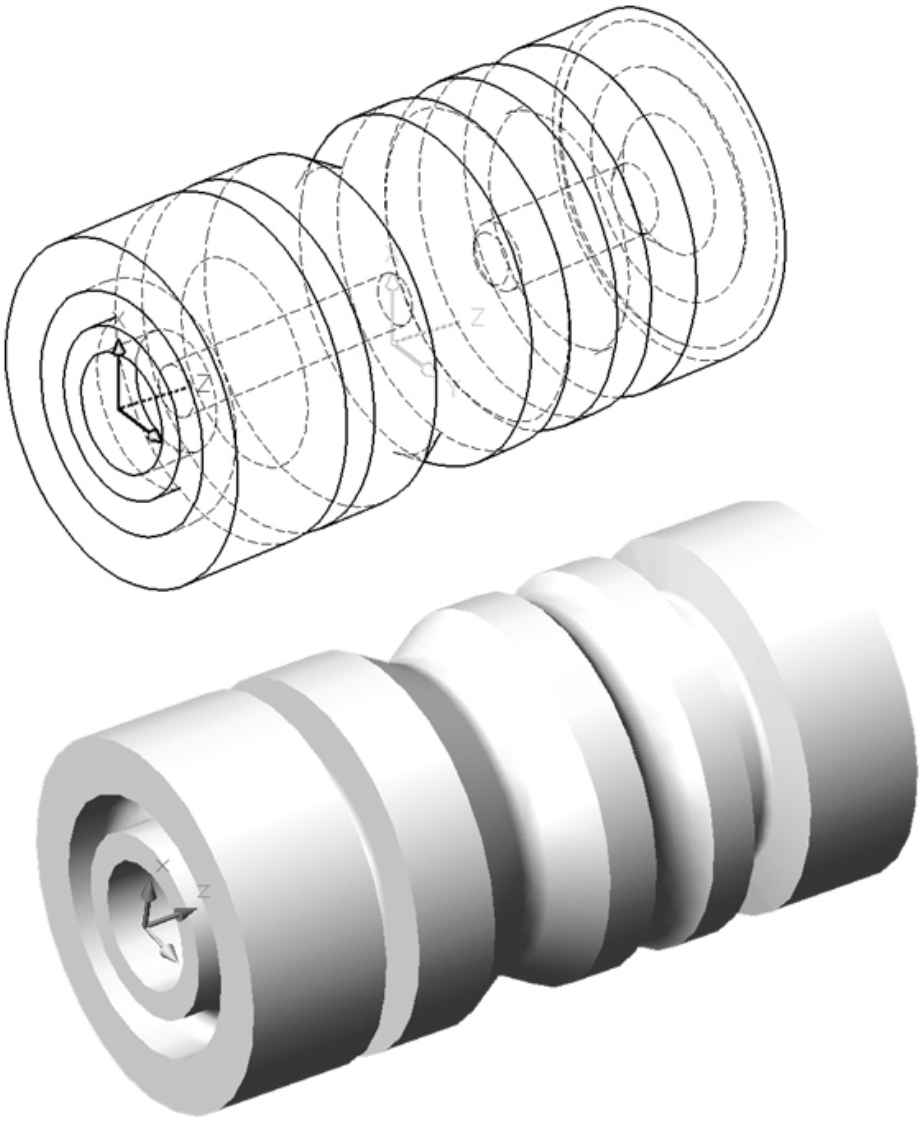

Figure 6.4 Isometric (upper) and Solid (lower) Views of Sample Part 1

The developed error handling mechanisms mentioned in previous chapters, check feature interactions as each feature is created to avoid design of meaningless part geometries. In case of feature interactions, the feature modeler warns and informs the designer about the condition by means of error messages and by showing the interacting case on the screen. When, the designer is made to recognize the interacting case the newly attached interacting feature is deleted from the overall part.

Once the design process is successfully completed, the feature information may be exported as an STEP XML design output file within the feature modeler. By means of triggering the preprocessor with a button and entering a name for the output file the related data is exported into a neutral file. The preprocessor first places the outer feature definition in the output file, and then places the inner features in the order of their creation on the outer feature. The STEP XML file including every necessary feature data is fairly long therefore, in the following part of this section, sample portion of the output file will be presented. The sample STEP XML output representation includes definitions for machining feature straight outer diameter and vee groove, which are placed on the first sample rotational part. (third and fifth features in Figure 6.3, respectively)

```
- <machining_feature>

- <!-- Straight Outer Diameter, SOD1 -->
                - <placement>
                  <location x="0" y="0" z="0" />
                - <!-- Right Feature  -->
                - <axis>
                  <x angle_x="0" angle_y="90" angle_z="-90" />
                  <y angle_x="-90" angle_y="0" angle_z="90" />
                  <z angle_x="90" angle_y="-90" angle_z="0" />
                  </axis>
          </placement>

- <!-- Outer Diameter Feature Definition  -->
        - <outer_diameter id="SOD1">
          <feature_length>400</feature_length>
          <diameter>200</diameter>
          </outer_diameter>

  </machining_feature>
```

```
- <machining_feature>

- <!-- Vee Groove, VEG1,(1)-->
        - <placement>
                <location x="80" y="0" z="180" />
              - <!-- Outer Groove, Parent Right -->
              - <axis>
                <x angle_x="90" angle_y="-90" angle_z="0" />
                <y angle_x="0" angle_y="90" angle_z="-90" />
                <z angle_x="-90" angle_y="0" angle_z="90" />
        - <!--Outer Groove Orientation, indicates alignment if necessary -->
          </axis>
          </placement>

- <!-- Vee Groove Feature Definition -->
        - <vee_groove id="VEG1,(1)">
              - <material_side>
                      <direction_element x="-1" y="0" z="0" />
                    - <!-- Outer Groove  -->
                </material_side>
                <radius>80"</radius>
              - <vee_profile>
                      <profile_radius>15</profile_radius>
                      <profile_angle>120</profile_angle>
                      <tilt_angle>30</tilt_angle>
                      <depth>20</depth>
                </vee_profile>
          </vee_groove>

  </machining_feature>
```

5.2) Second Sample

Second sample provides an understanding about the wide variety of possible outer features and their placements those can be used in rotational part design. Second sample also presents how inner features are oriented when more then one outer features exists in the part designed. In Figure 6.5 and Figure 6.6, dimensions necessary to design the rotational part in the feature modeler are given in two different views. These dimensions are given considering the feature geometric attributes and absolute placement attributes of features with respect to WCS. Placement attributes are given considering machining and tool aspects. In Figure 6.7, each feature constructing the overall sample part is labeled. The second sample part is fully designed by using the feature modeler. Figure 6.8 presents the 3D solid and isometric views of the sample part to provide a better understanding about its 3D geometry. As is it seen in the figures, the rotational part designed may consist of a number of outer manufacturing feature and many inner features attached on the selected outer manufacturing features, unless it interacts with a pre-designed feature on the part. Variety of outer manufacturing features can be used in the feature modeler to obtain the desired outer contour of the part. These feature are placed to the left or right of each other, and the feature modeler offers to keep the diameter same by default. However, depending on the requirements outer features with different diameters may also be attached to each other. Feature modeler also has the ability to place grooves on the wall surfaces, resulting from diameter differences.

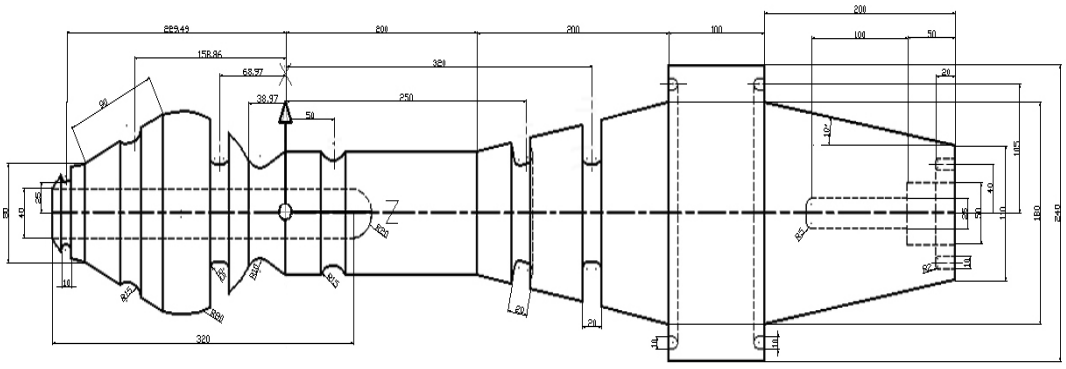

Figure 6.5 Dimensions to Design Sample Part 2 in Feature Modeler (front view)

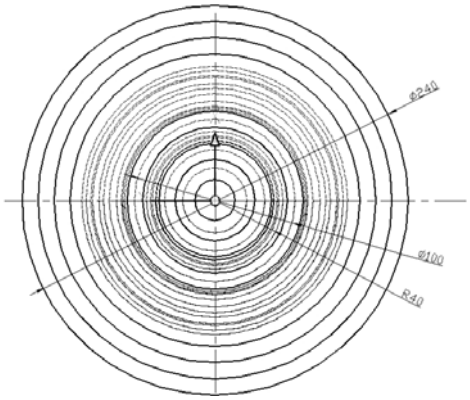

Figure 6.6 Dimensions to Design Sample Part 2 in Feature Modeler (right view)

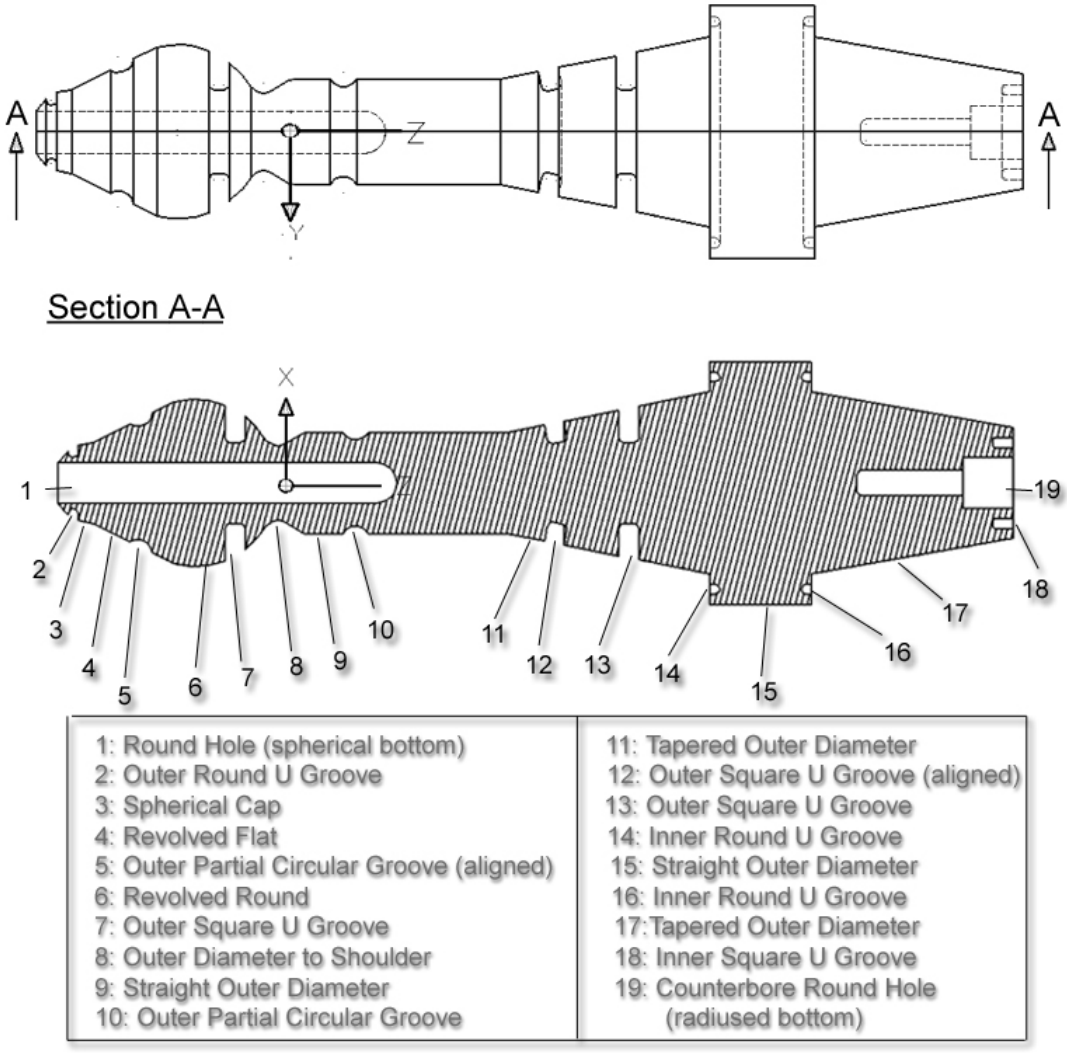

Figure 6.7 Top(upper) and Section (lower) Views of Sample Part 2

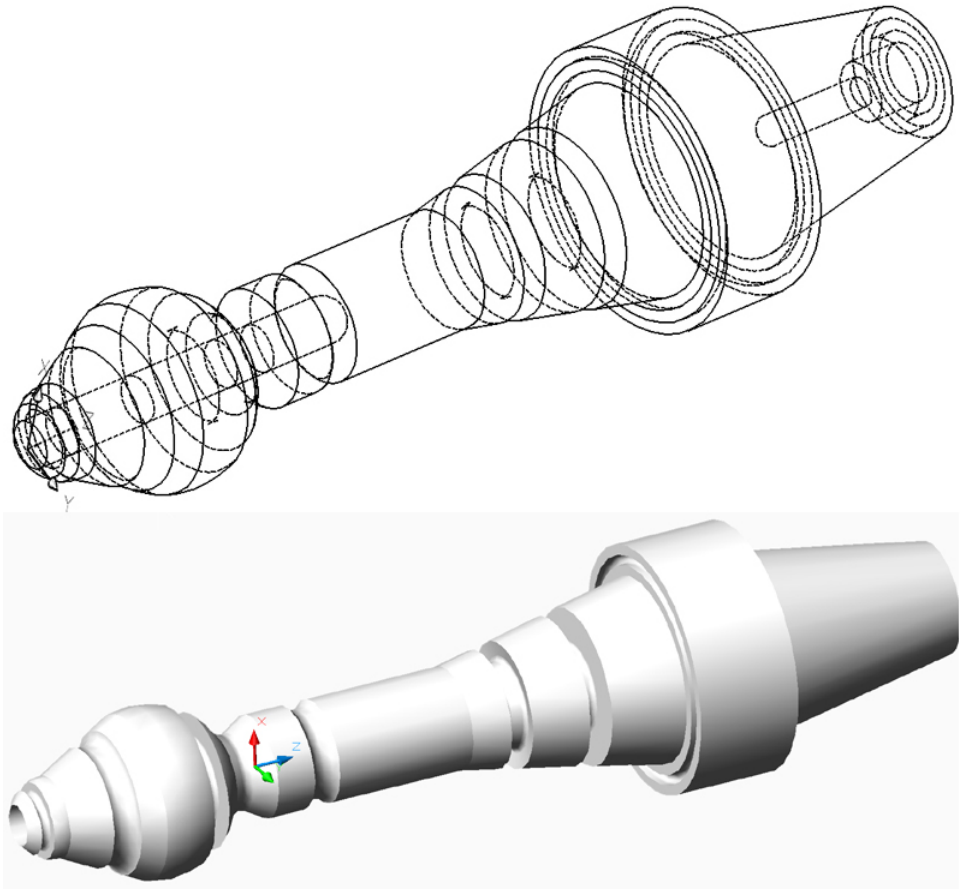

Figure 6.8 Isometric (upper) and Solid (lower) Views of Sample Part 2

Error handling mechanisms work same as it was mentioned for the first sample feature. However, if an inner feature exceeds an outer feature (this is allowed for holes) the feature modeler automatically creates the exceeding part of this feature as a member of that feature it goes through. This makes the check of the feature interactions for the whole inner feature possible.

As it was mentioned for the first sample, after completing the design process STEP XML file can be exported. In general, preprocessor arranges the outer features from left to right and places the inner features attached on the outer features in the order of their creation in between the outer feature. The STEP XML file for second sample part is very long, so just a portion of the file for the inner square u groove feature is presented: (eighteenth feature in Figure 6.7)

```
- <machining_feature>

- <!-- Square U Groove, SUG2,(4)  -->
        - <placement>
                <location x="40" y="0" z="680" />
              - <!-- Right Inner Groove, Parent Right  -->
              - <axis>
                <x angle_x="-180" angle_y="-90" angle_z="90" />
                <y angle_x="90" angle_y="-90" angle_z="0" />
                <z angle_x="-90" angle_y="0" angle_z="90" />
              - <!-- Right Inner Groove Orientation -->
                </axis>
          </placement>
- <!-- Square U Groove Feature Definition -->
        - <square_u_groove id="SUG2,(4)">
              - <material_side>
                      <direction_element x="0" y="0" z="-1" />
                    - <!-- Right Inner Groove -->
                </material_side>
                <radius>40"</radius>
              - <square_u_profile>
                      <first_angle>90</first_angle>
                      <first_radius>2</first_radius>
                      <second_angle>90</second_angle>
                      <second_radius>2</second_radius>
                      <width>10</width>
                      <depth>20</depth>
                </square_u_profile>
          </square_u_groove>

  </machining_feature>
```

CHAPTER 6

CONCLUSION

6.1) Conclusion

In design and manufacturing, many systems are used to manage technical product data. Each system has its own data formats so the same information has to be entered multiple times into multiple systems leading to redundancy and errors. The novel idea of feature based solid modeling by using STEP AP224 manufacturing features for rotational parts, which stands as the main objective of this study arouse after the recognition of this problem. With the purpose of developing a feature modeler offering STEP AP224 manufacturing features as an aid to design rotational parts, a methodology, first collecting the standard manufacturing features for rotational parts in a feature library in a hierarchical manner and bringing them up to a functional state, then providing an efficient design environment using these features, has been developed. Following the methodology developed, feature library and feature modeler are developed respectively in the scope of this study.

The developed feature library covers features extracted from STEP AP224 for rotational parts, their definitions, classifications, attributes, generation techniques, attachment techniques and constraints for their geometries. Object-oriented approach has been used in the definition of features, by means of EXPRESS language, which makes the system easy to be implemented, extended and reused. By this way, each feature is represented by class libraries and a dynamic link library (dll) file is created, facilitating to transfer the feature definition structure of features from STEP AP224 documentation to programming environment. By means of the dll file created, each feature and resulting part is provided to store manufacturing

.

features and their attributes. The dll file will also assist the integration of developed system for rotational parts with the feature modeler for prismatic parts developed by Saleh Amaitik in Middle East Technical University, Mechanical Engineering Department, Computer Integrated Manufacturing Laboratory (METUCIM), Ankara, Turkey.

The design environment is developed by embedding STEP based feature modeler for rotational parts as a design tool in AutoCAD. The feature modeler for rotational parts uses STEP AP224 manufacturing features in the feature library as the basic entities for part design. Features offer designers a higher level of graphical entity than points, lines and arcs. If properly selected, features can have additional information associated with them. Features also offer a ready means of linking to manufacturing. By this way, designers can consider design and manufacturing aspects in the earlier stages of design, which will remove manufacturability problems that may occur after the design process. Both in the feature modeler and feature library, error tracking and handling mechanisms are involved, validating the geometrical constraints and checking feature interactions, to provide an effective design environment.

In order not to leave features just as an aid to create the model, not to loose the information attached on them once the rotational part is fully designed, a preprocessor has also been developed in the scope of this study. Most importantly, the preprocessor should facilitate the use of design data in CAM/CAPP systems, which led the creation of STEP-XML design output file.

STEP-XML file will be a highly appropriate input data for integration, which is self-describing and a programmer can parse it without the need for further understanding of EXPRESS or any feature recognition algorithms. CAPP system, STEP-NC and G-Code generators are being developed and will be developed in METUCIM laboratory, using the STEP XML design output file as their only input. This way the data exchange, thus the integration between Feature Modeler, CAM and CAPP systems for rotational parts, has been facilitated.

6.2) Recommendations on Future Work

Recommended future work for both the feature library and the STEP-based feature modeler for rotational parts are stated below:

Future work that can be done on the feature library

1. Features like threads, marking, knurl and compound feature may be added to the feature library.
2. Replicate features may be added to the feature library.
3. Some inner features that may result from boring operations may be added as alternative feature subtypes to the existing feature library.

Future work that can be done on the feature modeler

1. Features like slots and transition features, which exist in the feature library but could not be added to the feature modeler, may be added to the feature modeler design environment
2. Part properties data in the standard may be integrated into the feature library and then to the feature modeler,
3. A translator may be developed that will generate STEP-NC file from the STEP-XML file created.
4. A translator may be developed that will generate traditional G-Code file from STEP-XML file created to make the feature modeler compatible with the traditional NC and CNC machine tools
5. To make the feature modeler a parametric modeling tool a post-processor may be developed in order to provide the designer to continue some incomplete designs as feature based as it were during the initial design process. During the save action for an incomplete design, an automatically generated STEP-XML file may be post-processed in order to make it possible to continue the design action with the previously designed features and their attributes.

6. Saving, editing, modifying options for features may be added in order to make the feature modeler more flexible. The program in the feature modeler is so developed that the arrays storing the feature data is apt to include these functionalities by means of further processing.
7. The dependency of the feature modeler to AutoCAD may be removed by means of using a standalone 3D drawing ActiveX environment.

REFERENCES

[1] El Wakil, S. D., “Processes and Design for Manufacturing”, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1989

[2] Mantyla, M., Nau, D., Shah, J., “Challenges in feature-based manufacturing research”, Communications of the ACM, February 1996, Vol 39, No.2

[3] Groover, M.P., “Automation, Production Systems and Computer Integrated Manufacturing”, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987

[4] 4 Coutts, I.A., Weston, R.H., Murgatroyd, I.S. and Gascoigne, J.D., “Open applications within soft integrated manufacturing systems.”, Proceedings of the International Conference on Manufacturing Automation, Hong Kong, August 10-12, 1992. pp 800-805.

[5] K Case, J X Gao and N N Z Gindy, “The Implementation of a Feature-Based Component Representation for CAD/CAM Integration”, IMechE Proceedings, Part B: Journal of Engineering Manufacture, Vol 208, pp71-80, ISSN 0954-4054, 1994

[6] Voelcker, H.B., “New directions in solid modeling?”, Proceedings of the International Conference on Manufacturing Automation, Hong Kong, August 10-12, 1992. pp 157-168.

[7] ISO 10303-1:1994, “Industrial automation systems and integration Product data representation and exchange - Overview and Fundamental Principles, International Standard”, ISO TC184/SC4, 1994.

[8] Amaitik S. M., Kilic S. E., (2002), “STEP: a key to CAD/CAM systems integration”, Proc. MicroCAD-2002 Conf. Miskolc, Hungary, 2002, 1-6.

[9] Gu, P. and Norrie, D.H. “Intelligent Manufacturing Planning”, Chapman & Hall, NY 1995

[10] Shah, J. J., Rogers, M. T., Sreevalson, P.C., Hsiao, D.W., Mathew, A., Bhatnagar, A., Liou, B. B., 1990, “The ASU features testbed: an overview”, Computer in Engineering,1, 233-241

[11] Febransyah, A., “A feature based approach to automating high-level process planning”, PhD Thesis, NCSU

[12] Li, R. K., “A part-feature recognition system for rotational parts”, International Journal of Production Research, Vol.26, No.9, pp.1451-1475

[13] Han, J. H., “Survey of feature Research”, Technical Report IRIS-96-346, Institute for Robotics and Intelligent Systems, USC, USA, 1996

[14] Sheu, J. J., “A computer integrated manufacturing system for rotational parts”, International Journal of Computer Integrated Manufacturing, 1998, Vol.11, No.6, 534-547

[15] PDES (1998) Product Data Exchange Specification: first working draft, PB89-144794, US Department of Commerce, Gaithersburg

[16] Kim, J-Y, O’grady, P., Young, R. E., “Feature taxonomies for rotational parts: a review and proposed taxonomies”, International Journal of Integrated Manufacturing, 1991, Vol.4, No.6, 341-350

[17] ISO TC184/SC4/WG7 N262, (1992) ISO 10303–Part 1-Overview and Fundamentals Principles.

[18] ISO TC 184/SC4/WD, (1997), ISO 10303 – Part 11 – Descriptive Methods: The EXPRESS Language Reference Manual.

[19] ISO TC 184/SC4/WG3 N854, (2000), ISO 10303, Part 224–Mechanical Product Definition for Process Planning Using Machining Features.

[20] Puopolo, J. P., 1997, Writing OLE Controls, Prentice Hall, p. 3

[21] O'Grady, P., Seshadri, R., 1991, "X-Cell – intelligent cell control using object-oriented programming (Part I)", Computer Integrated Manufacturing Systems, Vol.4 No.3, pp. 157-163

[22] Quatrani Terry, UML Evangelist, "Introduction to the Unified Modeling Language", IBM Rational Software, UML Resource Center, "http://www3.software .ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/intro_rdn.pdf", July 2004 (last accessed date)

[23] Bell Donald, "UML Basics, Part III: The Class Diagram", IBM Rational Software, UML Resource Center, "http://www.ibm.com/developerworks/rational/ library/content/RationalEdge/nov03/t_modelinguml_db.pdf", August 2004 (last accessed date)

[24] Bergholz Andre, "Extending Your Markup: An XML Tutorial", IEEE Internet Computing, "http://www.computer.org/internet/xml/xml. tutorial.PDF", June 2004 (last accessed date)

[25] Latif, M. N., Hannam, R. G., Bergholz, "Feature based Design and The Object Oriented Approach", Journal of Engineering Design, 1996, Vol. 7 Issue 1, p 27

[26] Amaitik S. M., Kilic S. E., (2003), "A Review of STEP Features Technology", Proc. WESIC-2003 Conf. Miskolc, Hungary, 53-60.

[27] Amaitik S. M., Kilic S. E., (2002), “STEP-based feature modeler for CAPP”, Proc. ICRM-2002 Conf. Gaziantep, Turkey, 243-248.

[28] Martic Hardwick, “A Modular XML Implementation Method for STEP”, STEP Tools Inc., August 2004

APPENDIX A

SAMPLE CODE

A.1) Machining Features Class

The following code is taken as example, from one the most general class "*machining_features*" library. In the code, Parent machining features and member machining features are declared as public variables respectively, to be able to call them from anywhere in the feature modeler. This will also makes each parent feature a subclass of *machining_features* class. This code also presents how the inheritance structure is constructed all over the Dll file.

```
    'Parent machining Features
Public Outerdiameter As New outer_diameter
Public OuterDiametertoShoulder As New outer_diameter_to_shoulder
Public RevolvedFlat As New revolved_flat
Public RevolvedRound As New revolved_round
Public SphericalCap As New spherical_cap
    'Member machining Features
Public SquareUGroove As New square_u_groove
Public RoundUGroove As New round_u_groove
Public PartialCircularGroove As New partial_circular_groove
Public TeeGroove As New tee_groove
Public VeeGroove As New vee_groove
Public RoundHole As New round_hole
Public CounterboreHole As New counterbore_hole
Public CountersunkHole As New countersunk_hole
```

A.2) Preprocessor

A sample part of preprocessor code that creates the STEP XML tags and fills the necessary information in between the tags for the feature *outer_diameter* is shown below:

```
Public Sub EXPORT_XML(XMLFileName As String, TabPos As Integer)
' Outer Diameter Features STEP XML Format (Partial) Function
Print #1, String(TabPos, Chr(9)) & "<outer_diameter id=" & Chr(34) & Me.featureID & Chr(34) & ">" ' Puts the outer_diameter and places Feature ID info
' Following part of the code check feature related information and places related tags and information.
Print #1, String(TabPos + 1, Chr(9)) & "<reduced_size>"
If Me.reduced_size.taper_type = 2 Then ' 2 stands for angle_taper
Print #1, String(TabPos + 2, Chr(9)) & "<angle_taper angle=" & Chr(34) & Me.reduced_size.angle_taper.angle & Chr(34) & "/>"
End If
If Me.reduced_size.taper_type = 3 Then ' 3 stands for diameter_taper
Print #1, String(TabPos + 2, Chr(9)) & "<diameter_taper final_diameter=" & Chr(34) & Me.reduced_size.diameter_taper.final_diameter & Chr(34) & "/>"
End If
Print #1, String(TabPos + 1, Chr(9)) & "</reduced_size>"
…..
```

The code continues checking every information related to the feature, it is omitted here. Tab Positions indicate where the cursor left in the XML file, so that necessary information is placed in the right position in the XML file. The above function is called from the feature modeler as it is given below:

```
If MFeatures(i, 0).FeatureType = outer_diameter Then Call MFeatures(i, 0). Outerdiameter.EXPORT_XML(FileName, 2)
```

A.3) Outer Diameter Class

A sample part of the code out of the *outer_diameter* class, in the Features Dll, including 2D feature region creation and 3D feature solid creation using the geometry definitions presented in Chapter 4, is provided below:.

```
Public Function StraightOuterDiameter() As Acad3DSolid
    'This function is used to generate Straight Outer Diameter and
    'add it from the main part as the specified location and orientation
    On Error GoTo ErrorHandler
    'Declaration of point arrays
    Dim p1(0 To 2) As Double            ' insertion point
    Dim p2(0 To 2) As Double
    Dim p3(0 To 2) As Double
    Dim p4(0 To 2) As Double

'Definitions of points (l and d are defined in the code before)
    p1(0) = 0
    p1(1) = 0
    p1(2) = 0

    p2(0) = 0
    p2(1) = d / 2
    p2(2) = 0

    p3(0) = l
    p3(1) = d / 2
    p3(2) = 0

    p4(0) = l
    p4(1) = 0
    p4(2) = 0
```

```
' Define the lines constructing the 2D region
Dim curves(0 To 3) As AcadEntity
Set curves(0) = ThisDrawing.ModelSpace.AddLine(p1, p2)
Set curves(1) = ThisDrawing.ModelSpace.AddLine(p2, p3)
Set curves(2) = ThisDrawing.ModelSpace.AddLine(p3, p4)
Set curves(3) = ThisDrawing.ModelSpace.AddLine(p4, p1)

' Create the2D region
Dim regionObj As Variant
regionObj = ThisDrawing.ModelSpace.AddRegion(curves)

' Define the rotation axis
Dim axisPt(0 To 2) As Double
Dim axisDir(0 To 2) As Double
Dim angle As Double

'Temporary position of the feature
axisPt(0) = 0:      axisPt(1) = 0:      axisPt(2) = 0
' axis of revolution is x-axis  (0 1 0 for y-axis,  0 0 1 for z-axis)
axisDir(0) = 1:     axisDir(1) = 0:     axisDir(2) = 0
angle = 44 / 7      ' for 360 degree full revolution

' Create the feature solid(3D) checking if it is right or left feature
 If Me.Right_Feature Then Set StraightOuterDiameter =
ThisDrawing.ModelSpace.AddRevolvedSolid(regionObj(0), axisPt, axisDir, angle)
      If Me.Left_Feature Then
      Set StraightOuterDiameter =
ThisDrawing.ModelSpace.AddRevolvedSolid(regionObj(0), axisPt, axisDir, angle)
      StraightOuterDiameter.Rotate3D p1, p2, 180 * DegToRad  'Rotates the feature
180 degrees around the line between points p1 and p2 if it is a left feature
      End If
```

APPENDIX B

FEATURE GEOMETRIES LIBRARY

In Chapter 3, "Manufacturing Features Geometry Data" component of the "Feature Library" has been introduced and the manufacturing features for rotational parts are classified and defined. In Appendix B, 2D sketches, generation techniques and insertion points for each manufacturing feature that is a member of the feature library is presented, all which are referenced from the "Manufacturing Features Geometry Data" subsection of Chapter 3. In all of the figures presented in this Appendix, the points with an asterisk ("*") stands for the insertion point for that feature and the dimensions placed on the features stand for the geometrical attributes required to create the feature in the feature modeler. These dimensions also stands for the feature attributes that the designer have to input the feature modeler to create the feature with same letter conventions appearing in the interfaces.

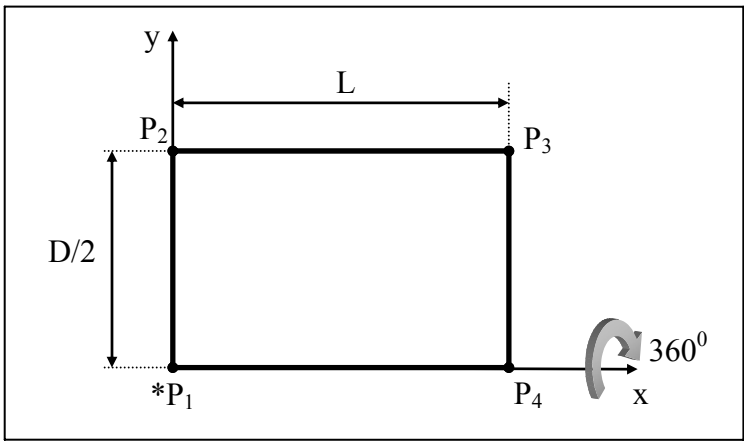

Figure B.1 Straight_outer_diameter

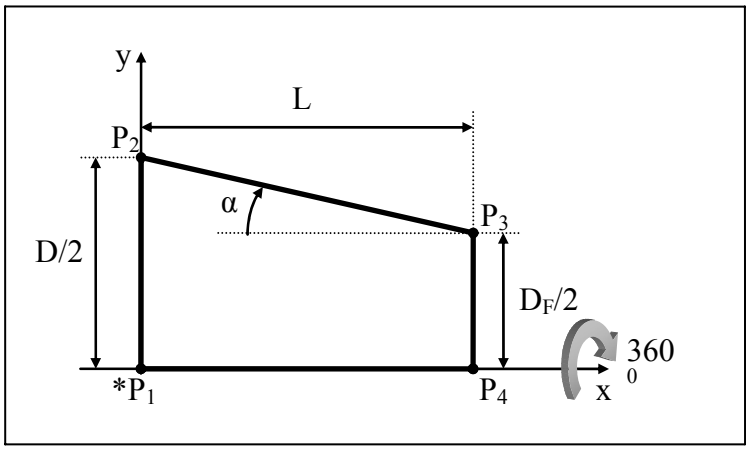

Figure B.2 Tapered_outer_diameter, decreasing diameter

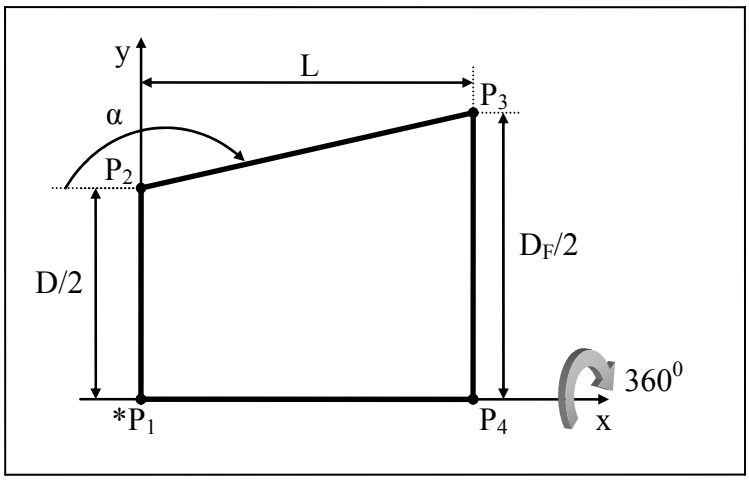

Figure B.3 Tapered_outer_diameter, increasing diameter

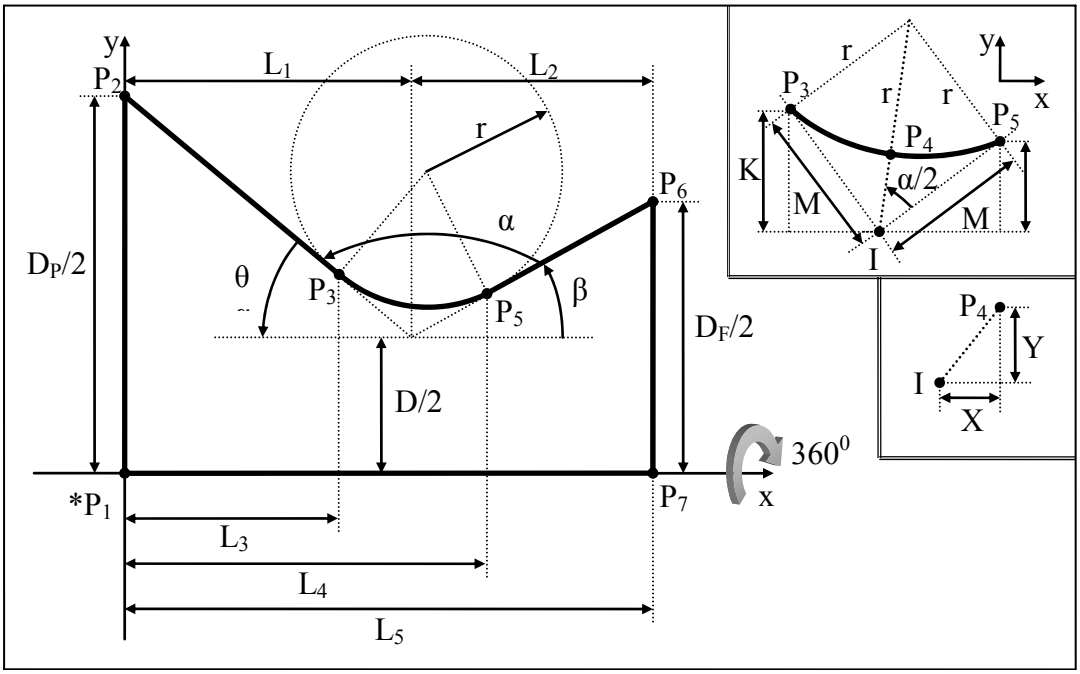

Figure B.4 Outer_diameter_to_shoulder

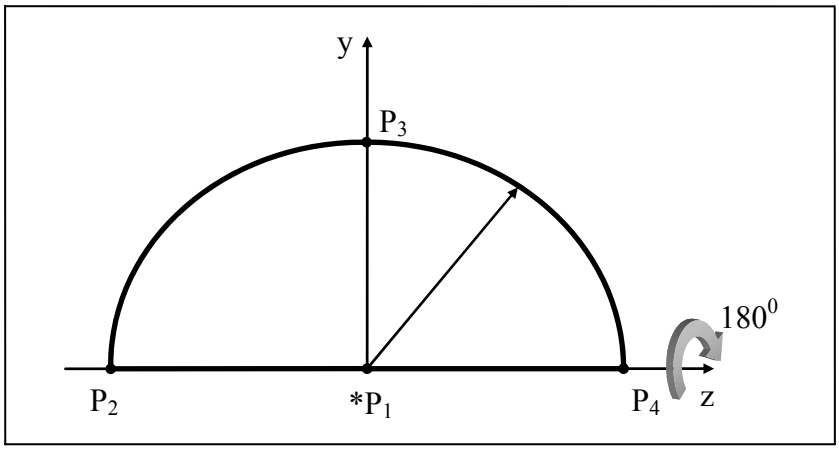

Figure B.5 Spherical_cap

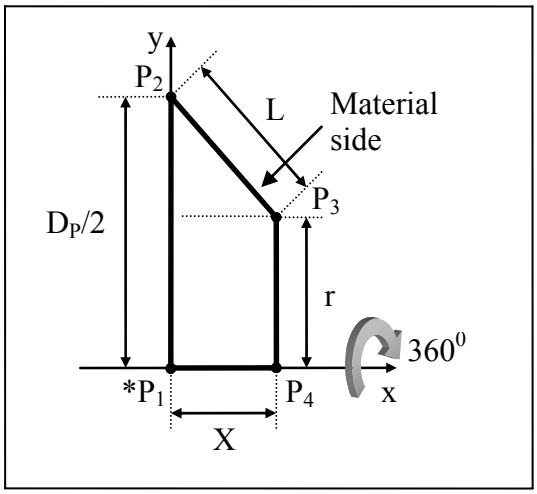

Figure B.6 Revolved_flat

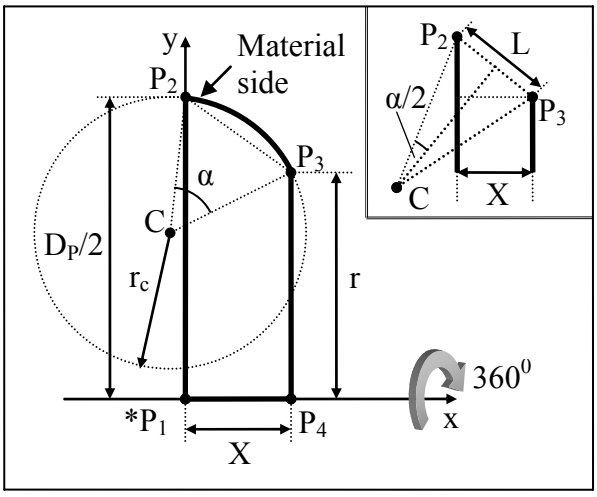

Figure B.7 Revolved_flat

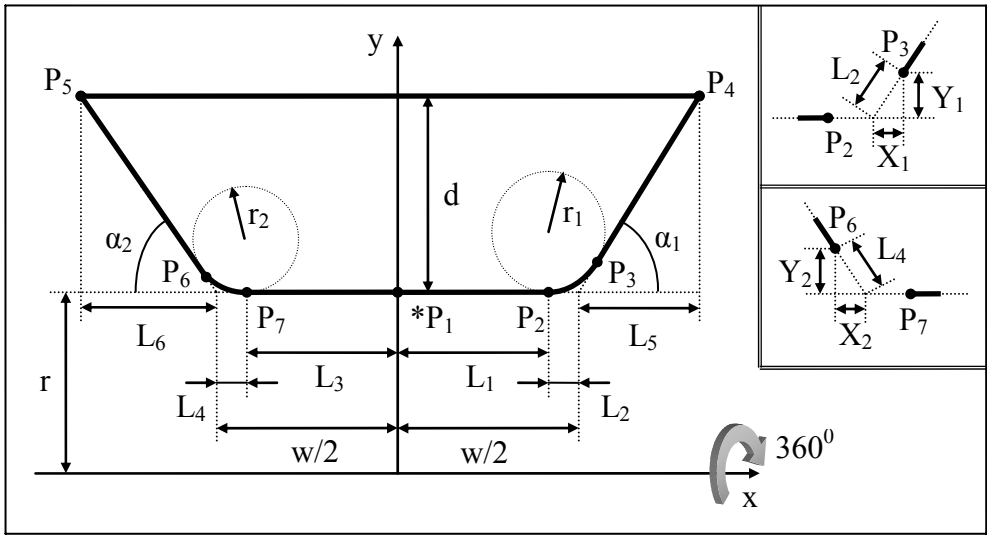

Figure B.8 Outer_square_u_groove

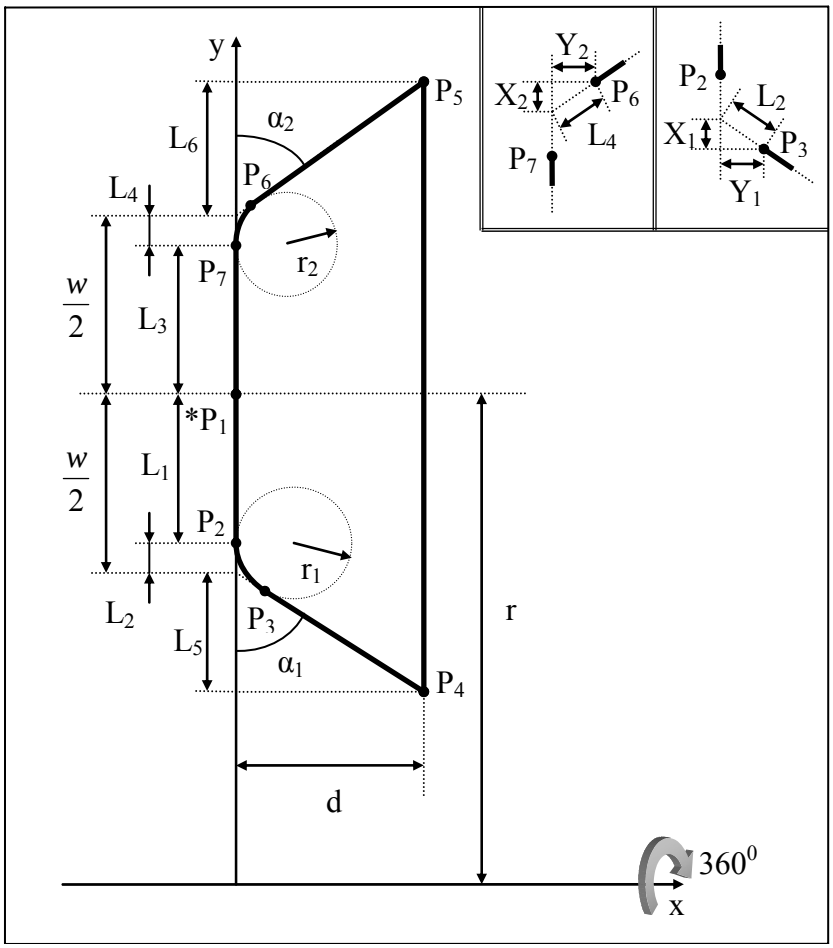

Figure B.9 Inner_square_u_groove

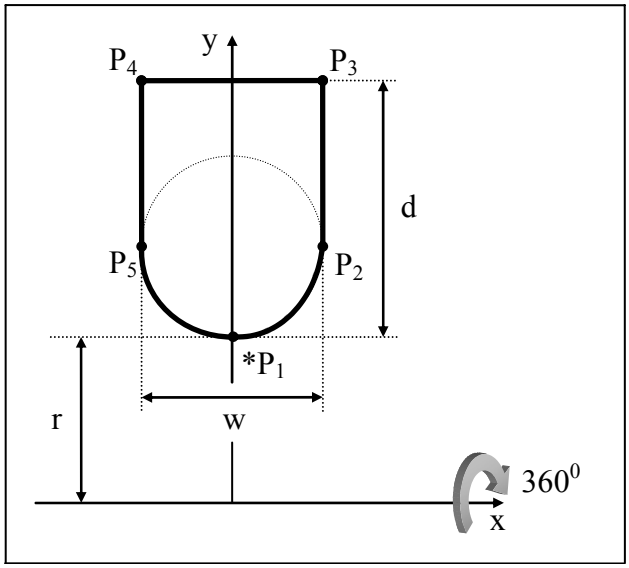

Figure B.10 Outer_rounded_u_groove

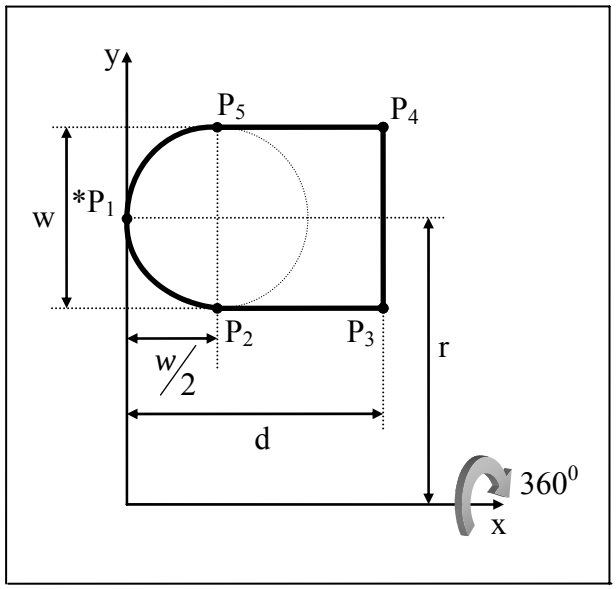

Figure B.11 Inner_rounded_u_groove

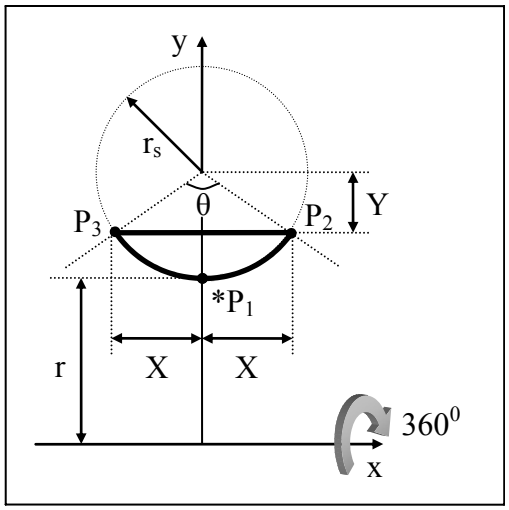

Figure B.12 Outer_partial_circular_groove

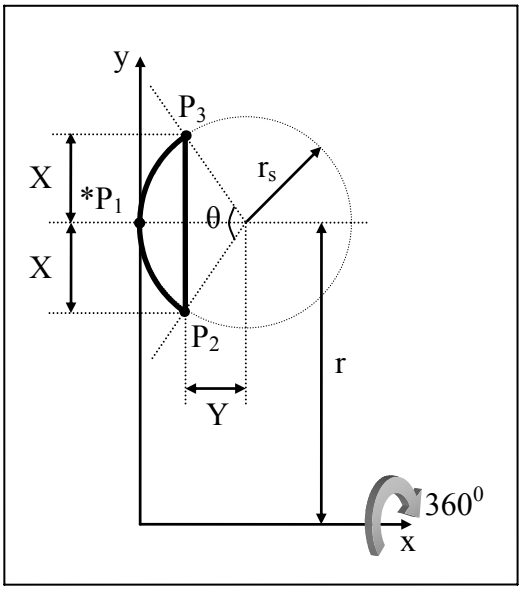

Figure B.13 Inner_partial_circular_groove

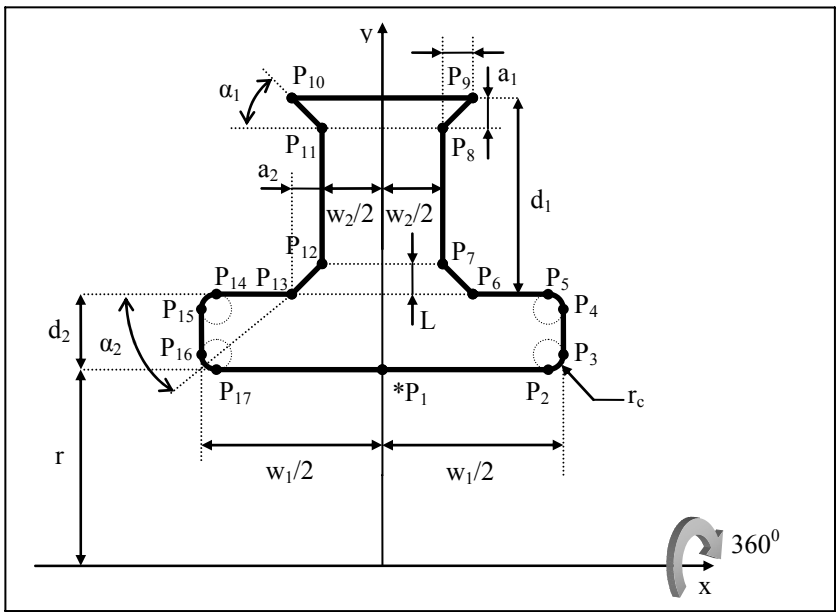

Figure B.14 Outer_tee_groove

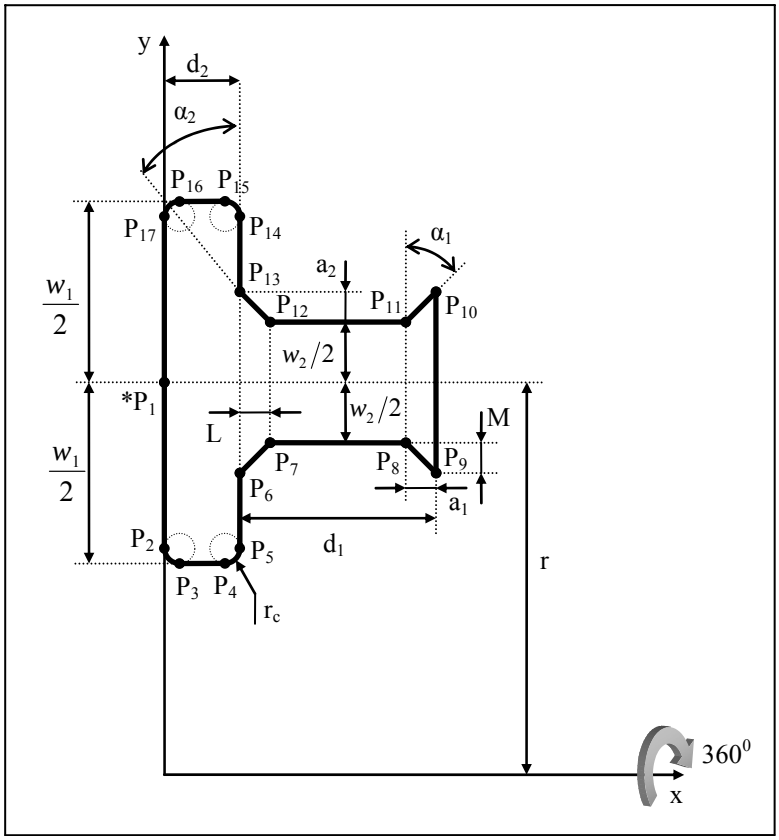

Figure B.15 Inner_tee_groove

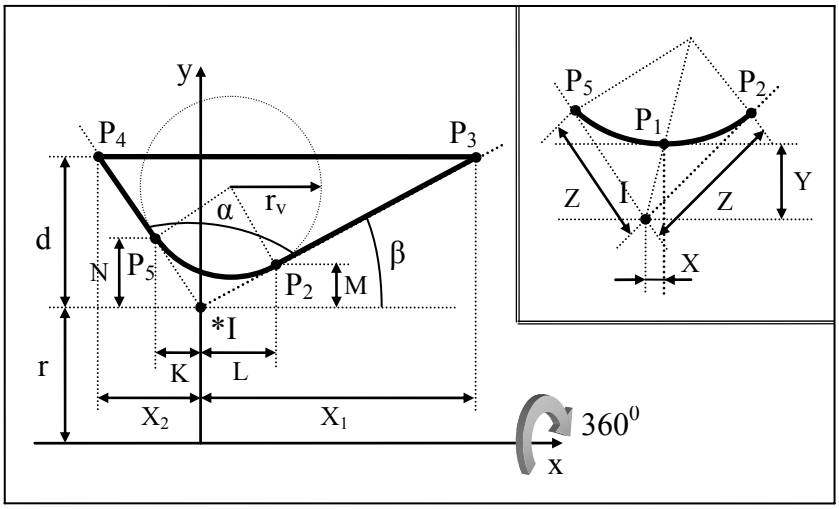

Figure B.16 Outer_vee_groove

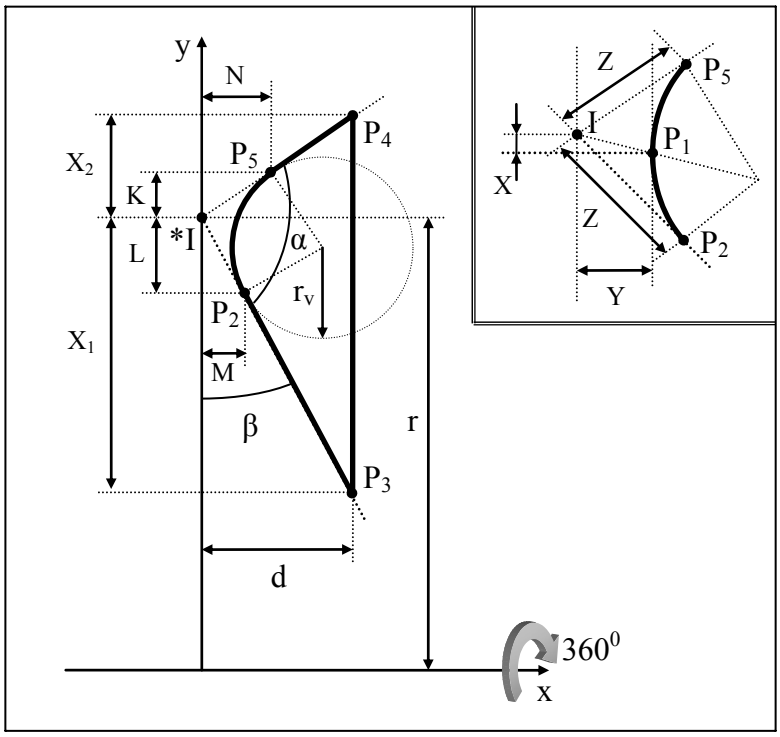

Figure B.17 Inner_vee_groove

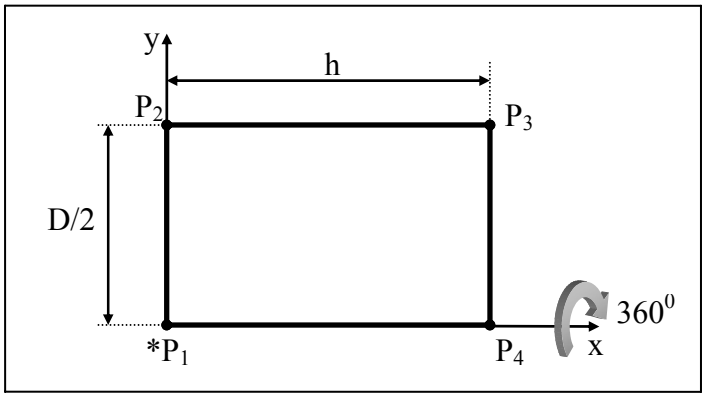

Figure B.18 Straight_through_round_hole

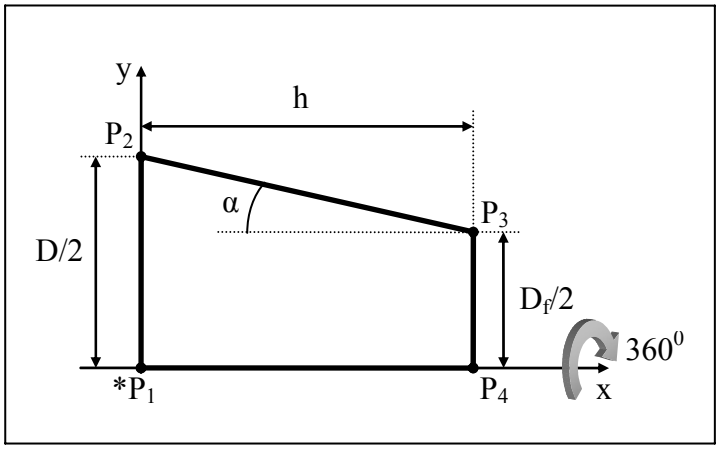

Figure B.19 Tapered_through_round_hole

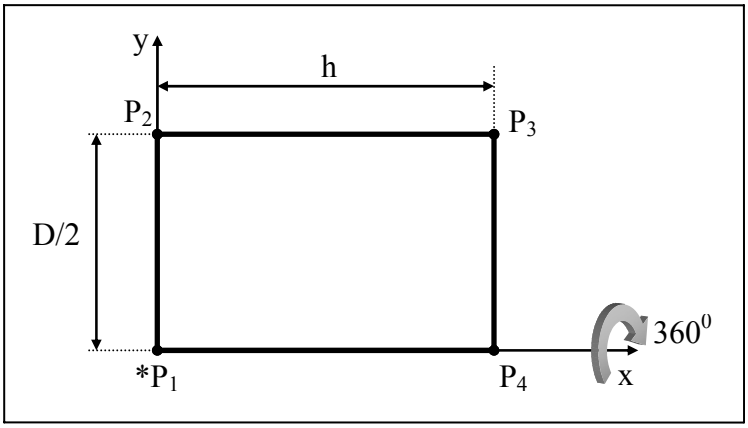

Figure B.20 Straight_blind_round_hole_with_flat_bottom

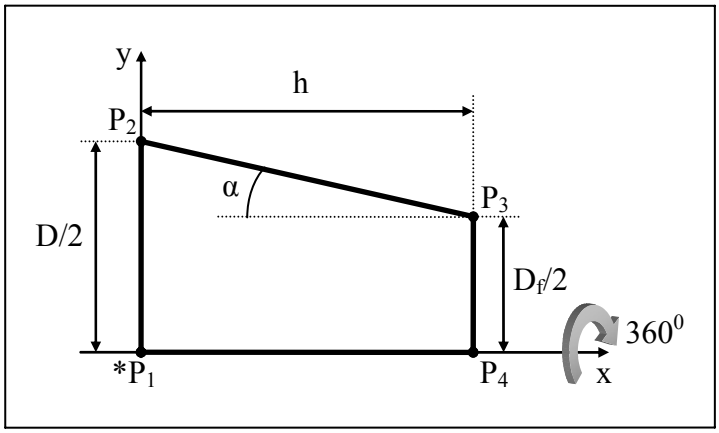

Figure B.21 Tapered_blind_round_hole_with_flat_bottom

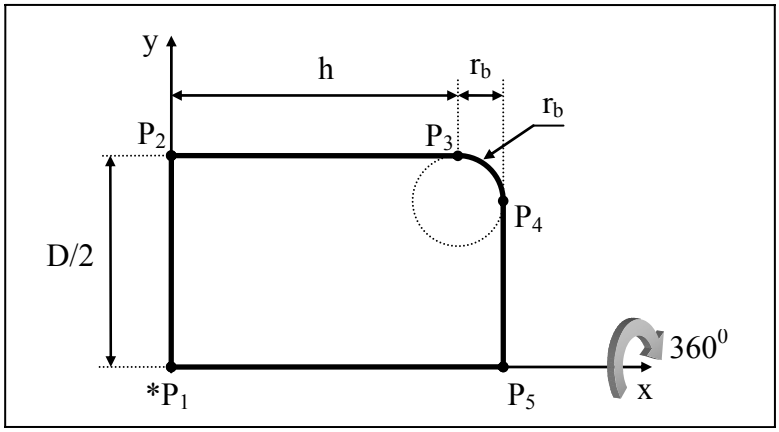

Figure B.22 Straight_blind_round_hole_with_radiused_bottom

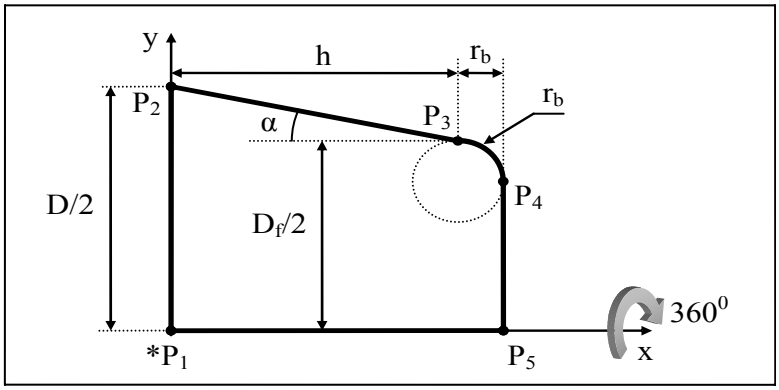

Figure B.23 Tapered_blind_round_hole_with_radiused_bottom

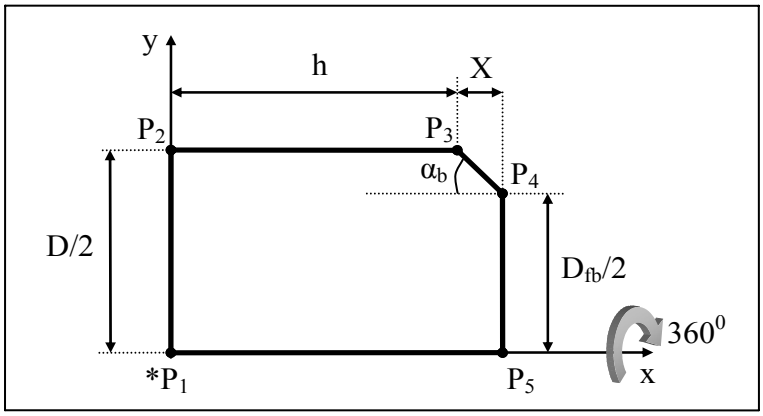

Figure B.24 Straight_blind_round_hole_with_tapered_bottom

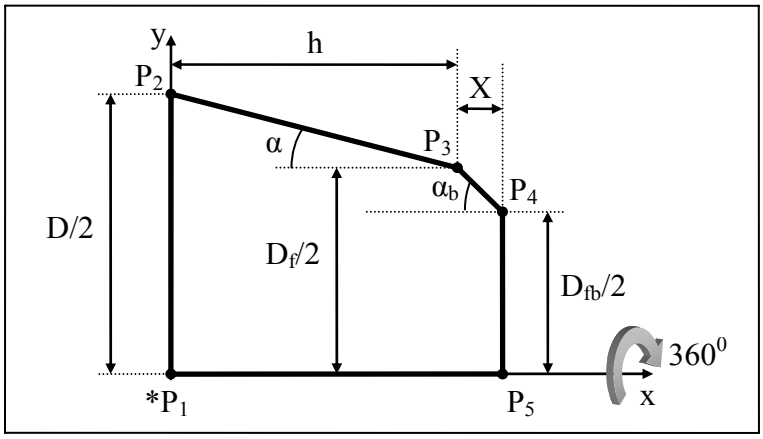

Figure B.25 Tapered_blind_round_hole_with_tapered_bottom

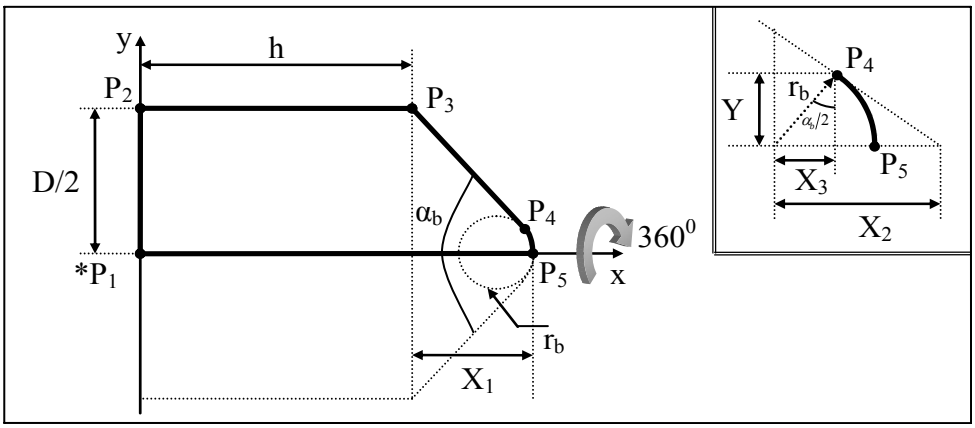

Figure B.26 Straight_blind_round_hole_with_conical_bottom

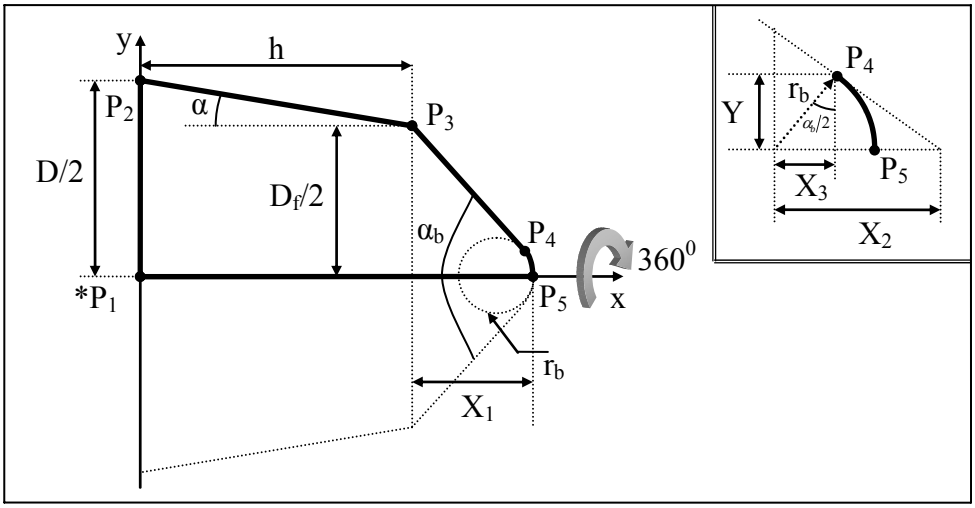

Figure B.27 Tapered_blind_round_hole_with_conical_bottom

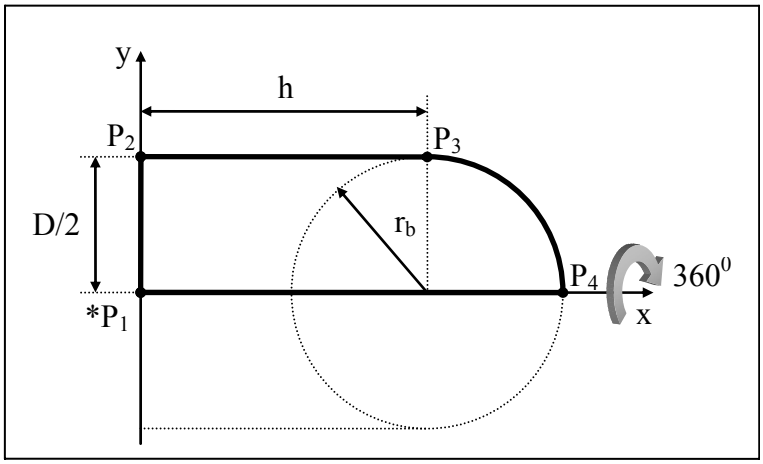

Figure B.28 Straight_blind_round_hole_with_spherical_bottom

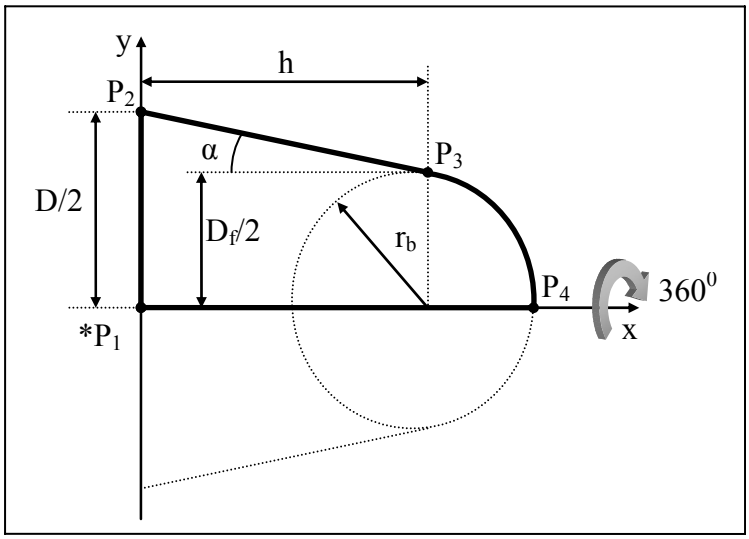

Figure B.29 Tapered_blind_round_hole_with_spherical_bottom

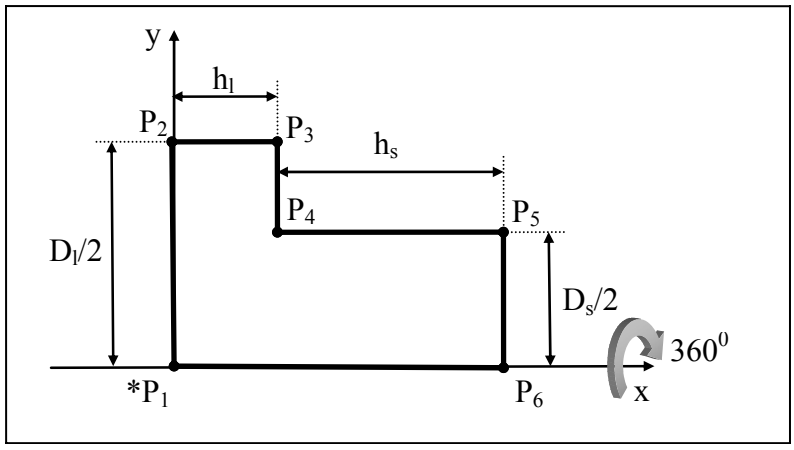

Figure B.30 Straight_through_counterbore_hole

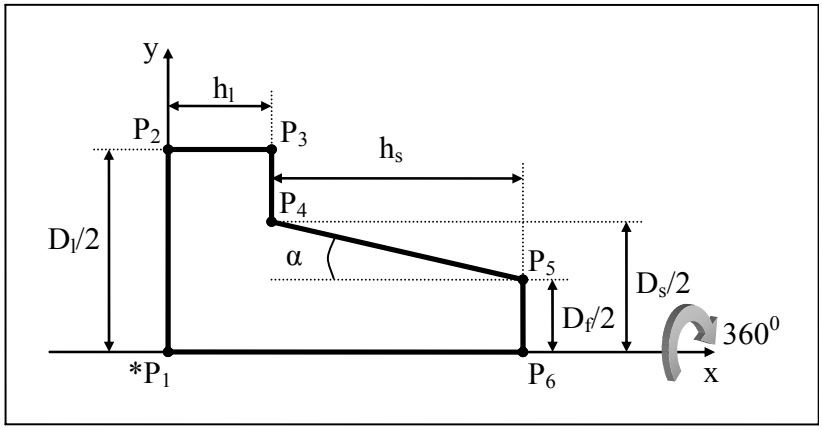

Figure B.31 Tapered_through_counterbore_hole

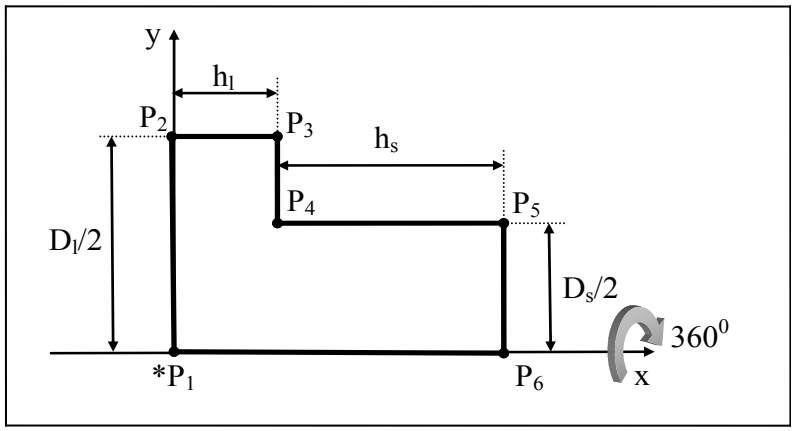

Figure B.32 Straight_blind_counterbore_hole_with_flat_bottom

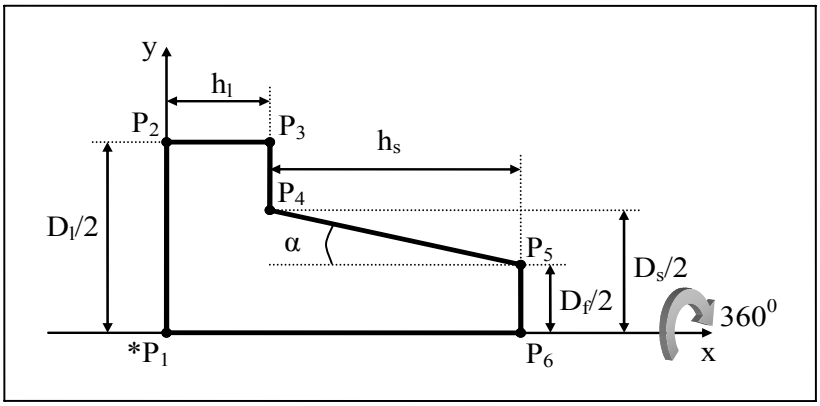

Figure B.33 Tapered_blind_counterbore_hole_with_flat_bottom

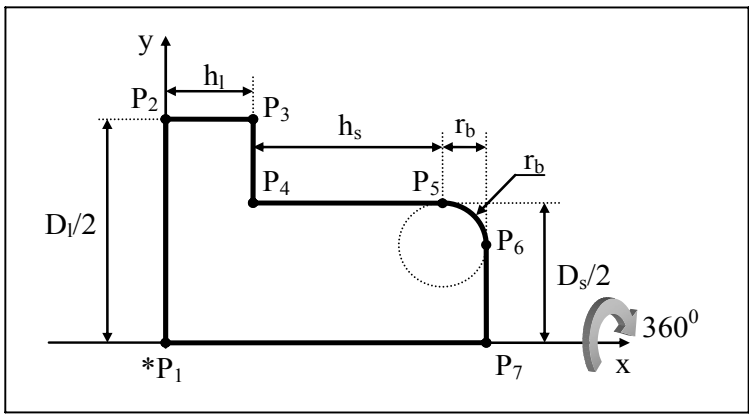

Figure B.34 Straight_blind_counterbore_hole_with_radiused_bottom

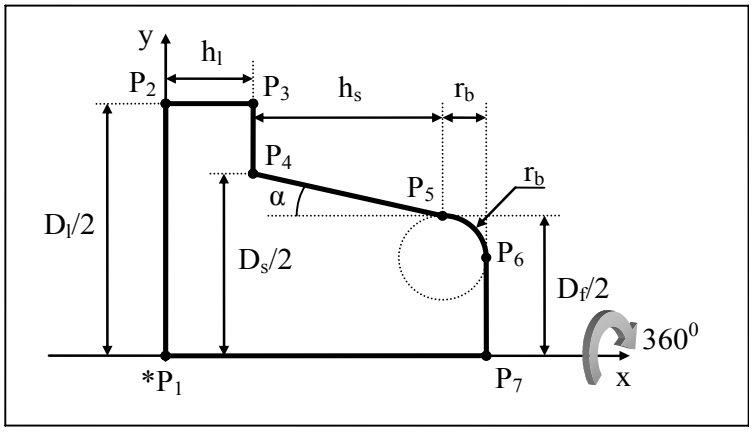

Figure B.35 Tapered_blind_counterbore_hole_with_radiused_bottom

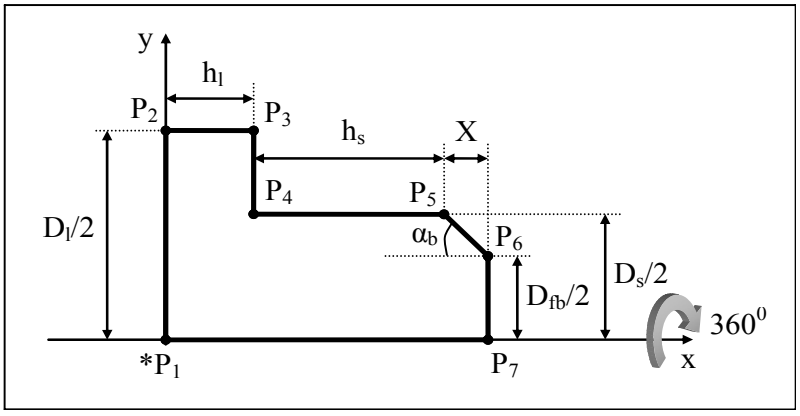

Figure B.36 Straight_blind_counterbore_hole_with_tapered_bottom

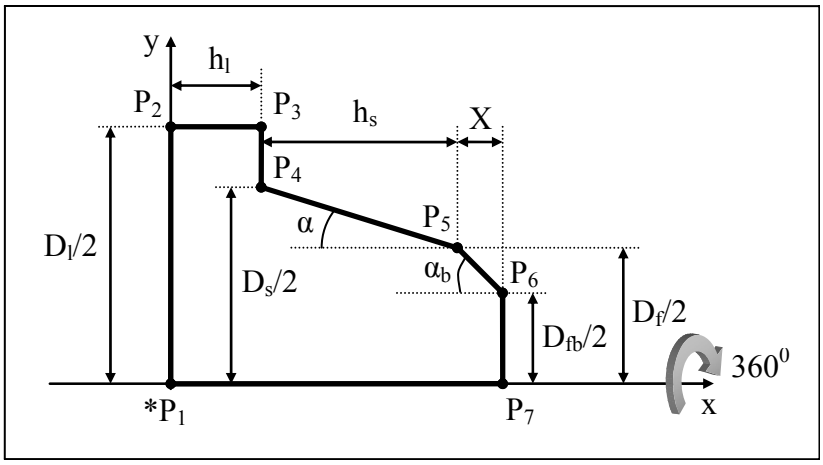

Figure B.37 Tapered_blind_counterbore_hole_with_tapered_bottom

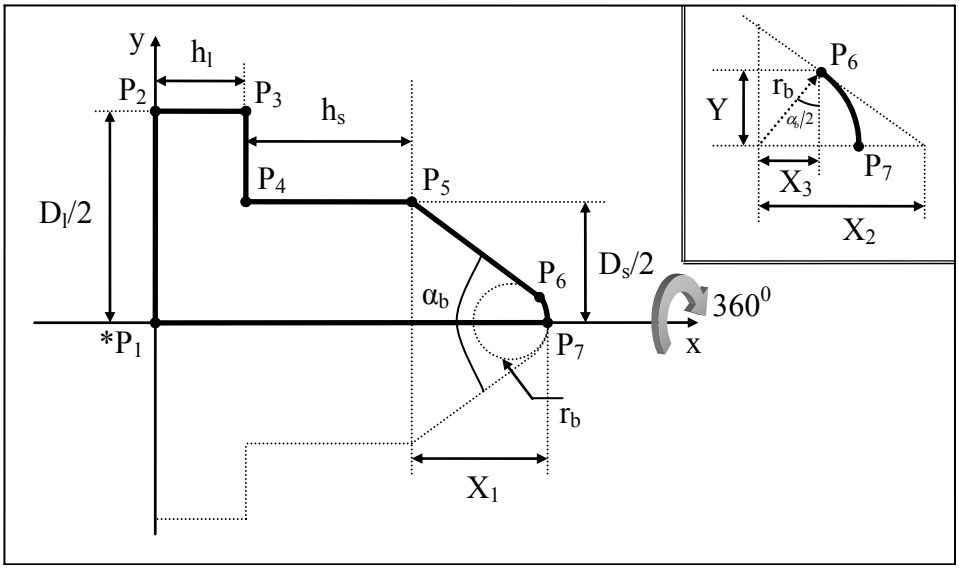

Figure B.38 Straight_blind_counterbore_hole_with_conical_bottom

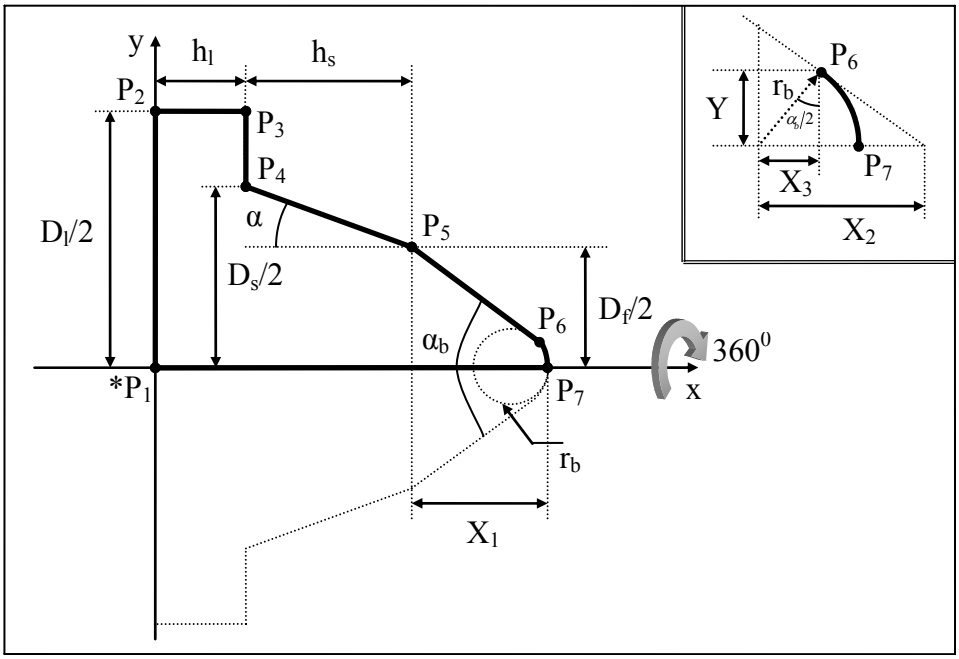

Figure B.39 Tapered_blind_counterbore_hole_with_conical_bottom

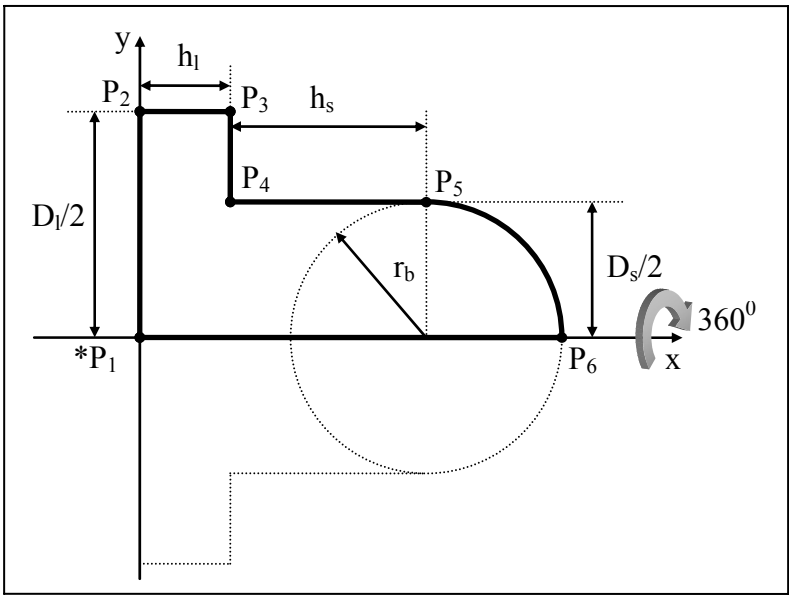

Figure B.40 Straight_blind_counterbore_hole_with_spherical_bottom

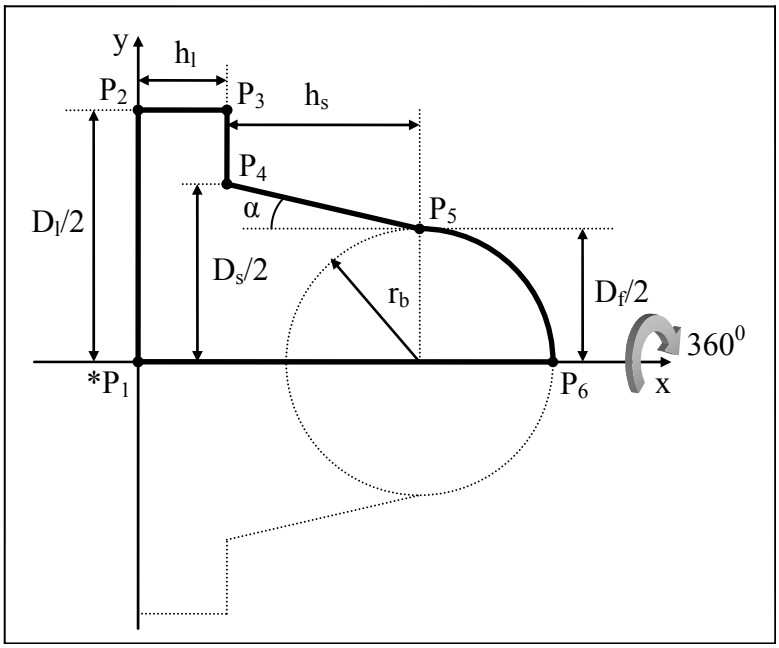

Figure B.41 Tapered_blind_counterbore_hole_with_spherical_bottom

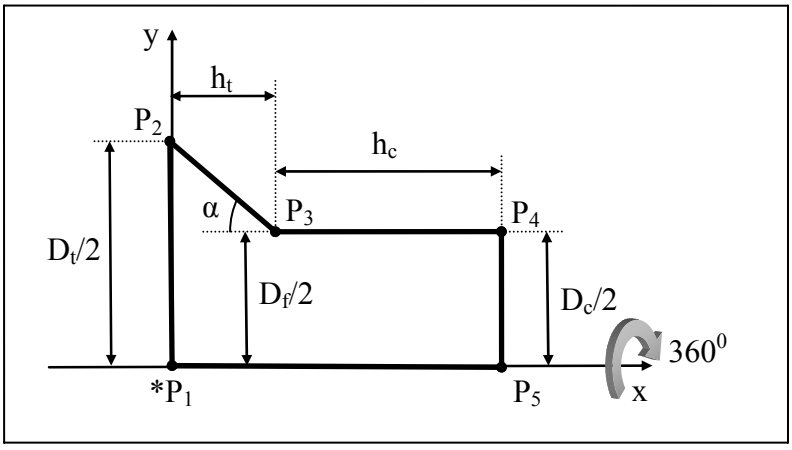

Figure B.42 Through_countersunk_hole

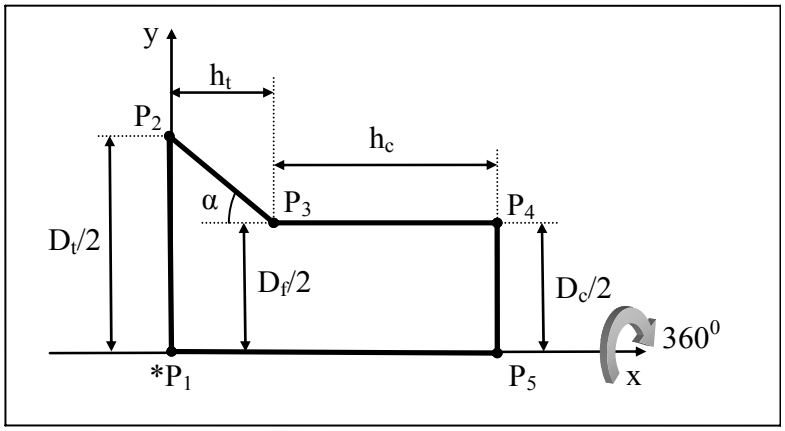

Figure B.43 Blind_countersunk_hole_with_flat_bottom

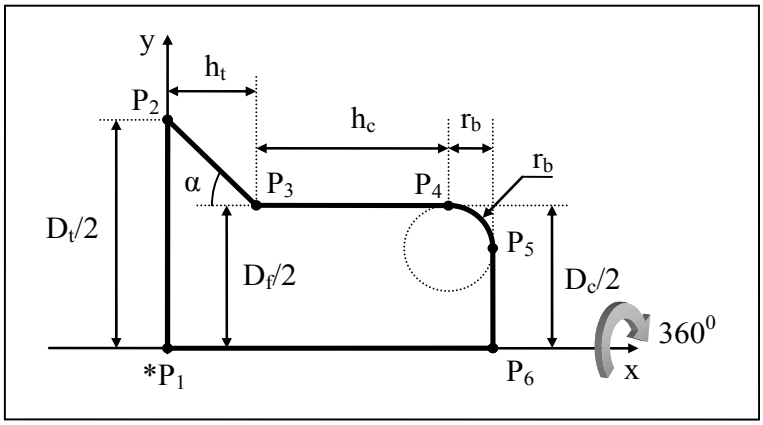

Figure B.44 Blind_countersunk_hole_with_radiused_bottom

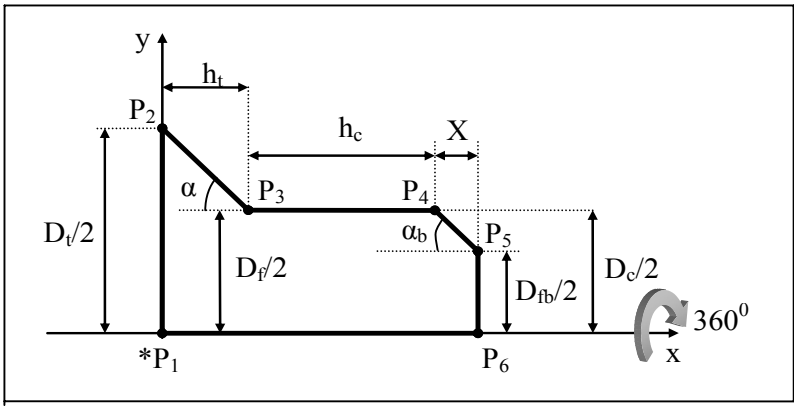

Figure B.45 Blind_countersunk_hole_with_tapered_bottom

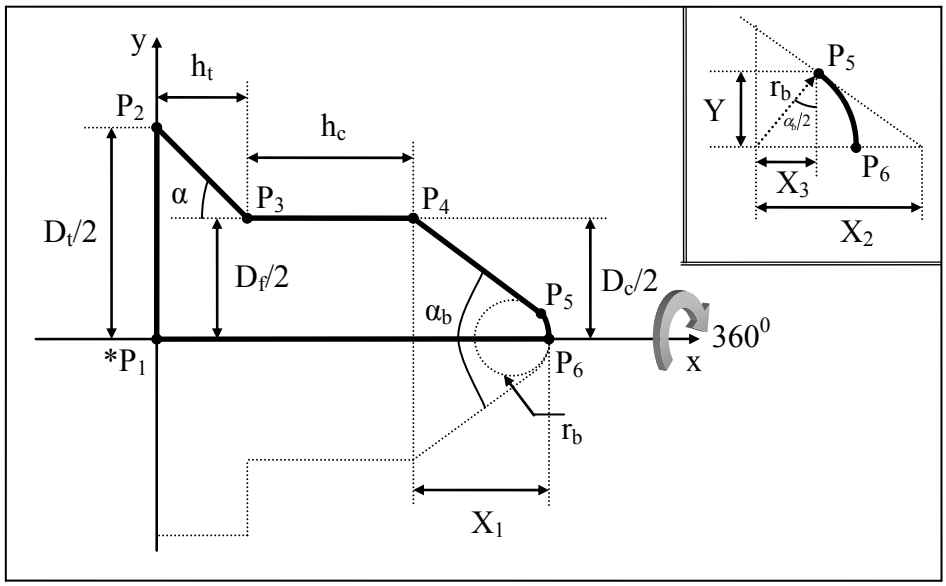

Figure B.46 Blind_countersunk_hole_with_conical_bottom

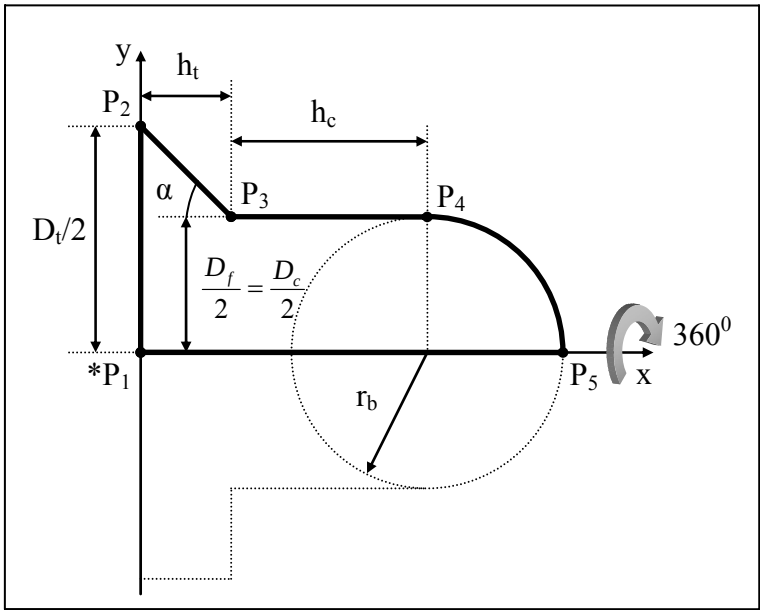

Figure B.47 Blind_countersunk_hole_with_spherical_bottom

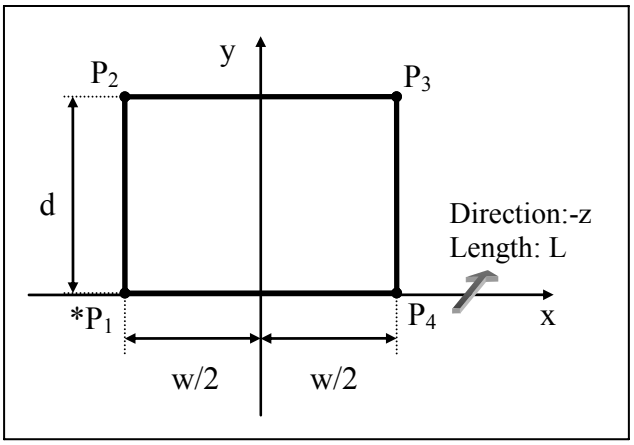

Figure B.48 Square_linear_slot_type1

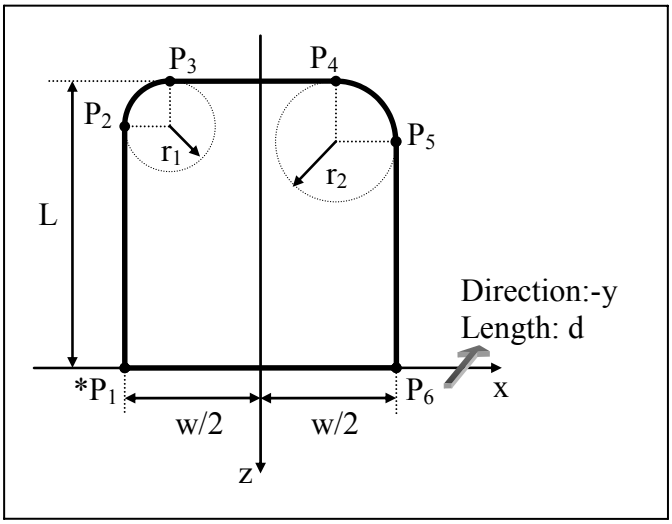

Figure B.49 Square_linear_slot_type2

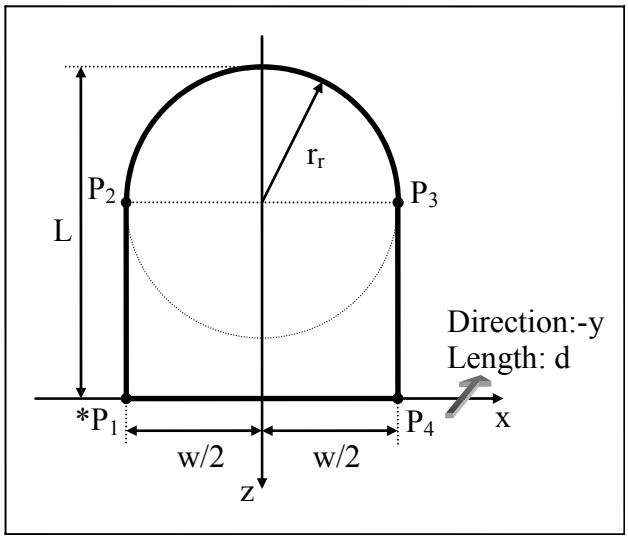

Figure B.50 Square_linear_slot_type3

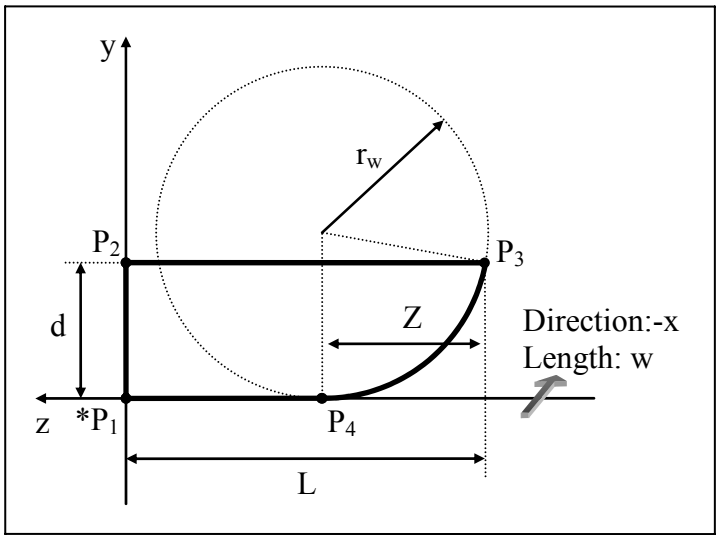

Figure B.51 Square_linear_slot_type4

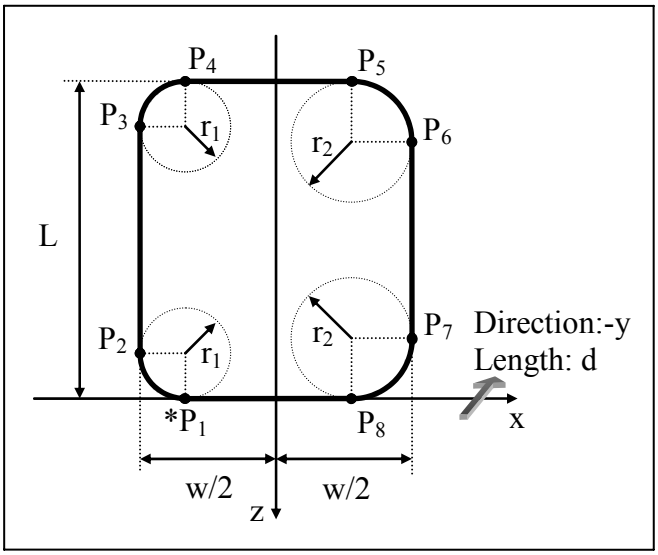

Figure B.52 Square_linear_slot_type6

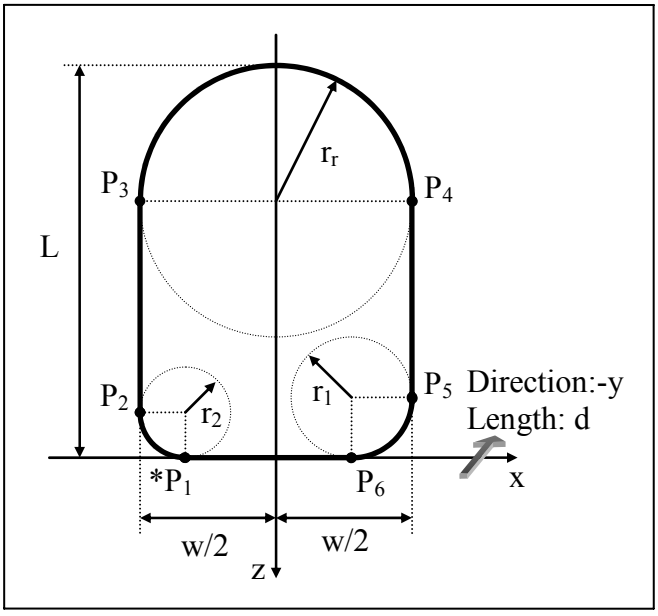

Figure B.53 Square_linear_slot_type7

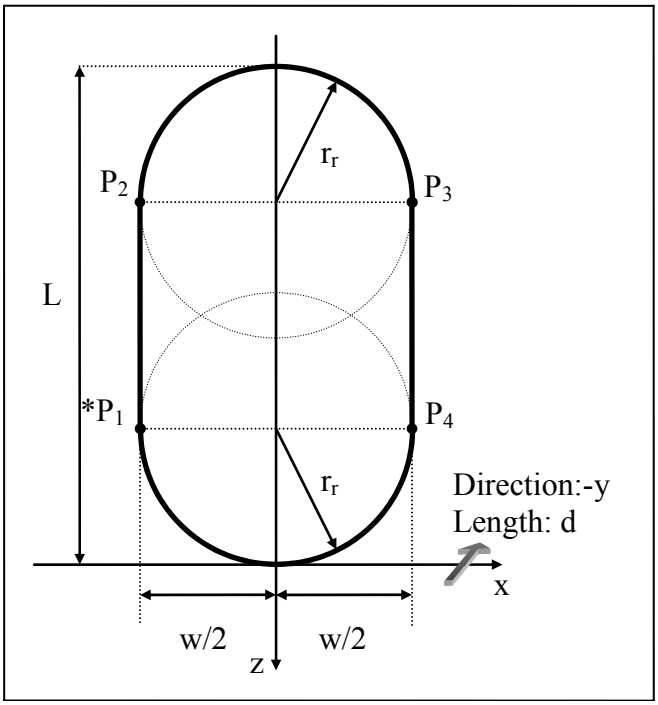

Figure B.54 Square_linear_slot_type10

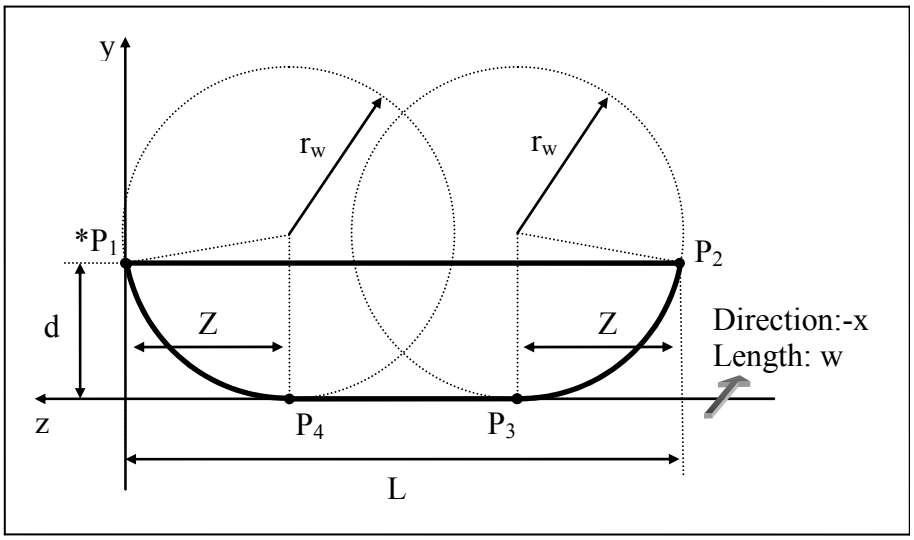

Figure B.55 Square_linear_slot_type12

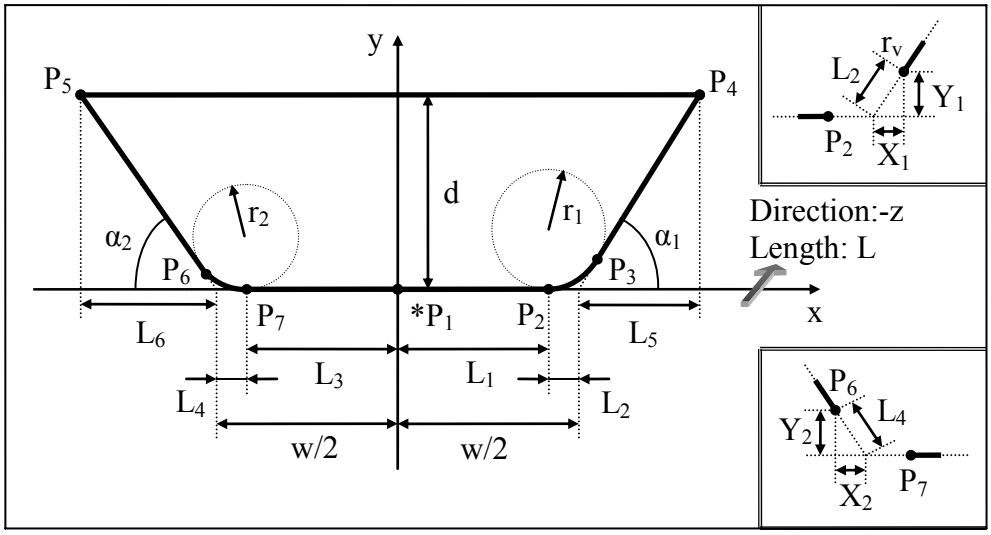

Figure B.56 Square_u_linear_slot

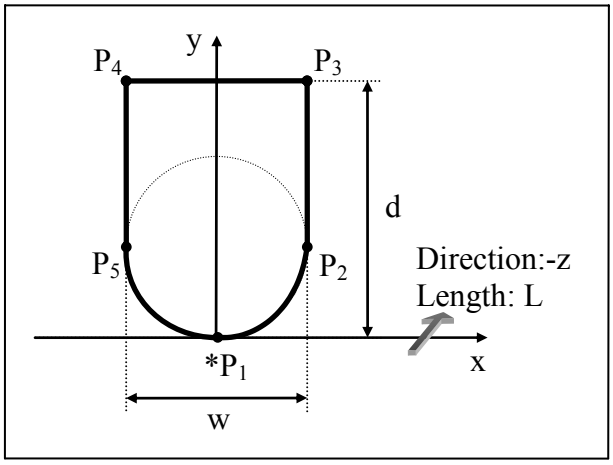

Figure B.57 Rounded_u_linear_slot

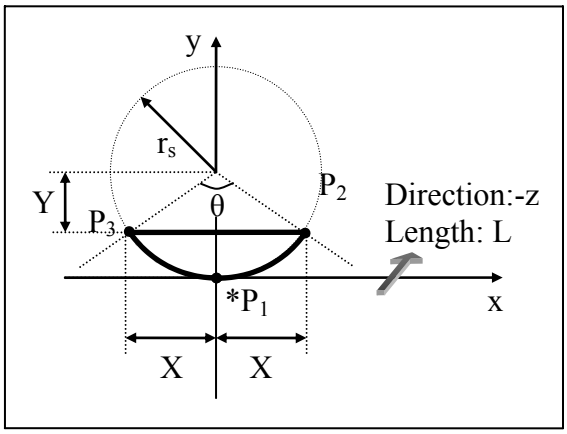

Figure B.58 Partial_circular_linear_slot

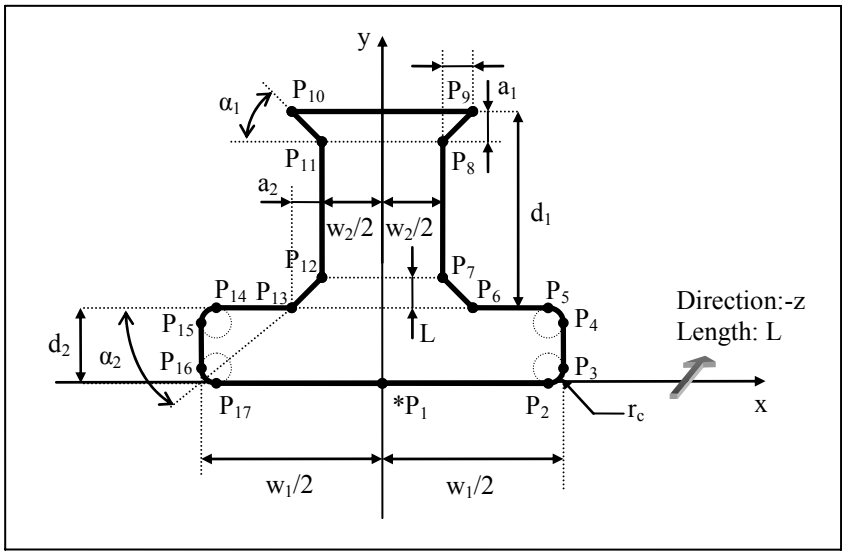

Figure B.59 Tee_linear_slot

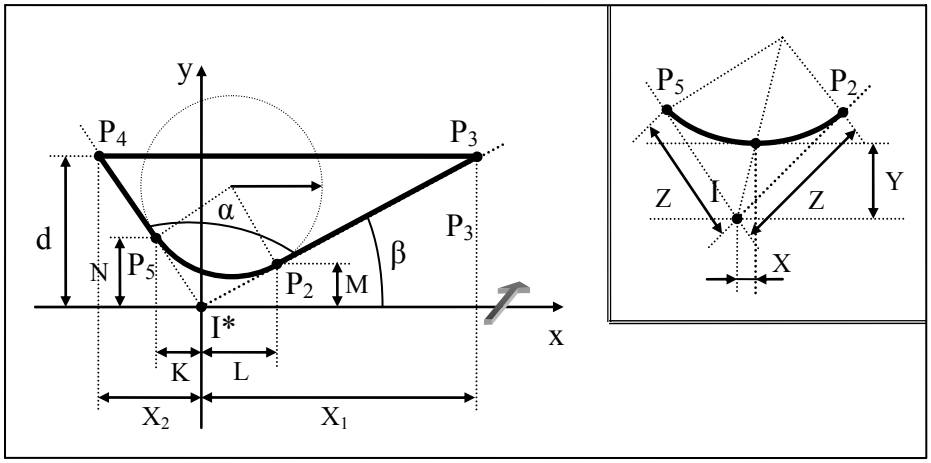

Figure B.60 Vee_linear_slot

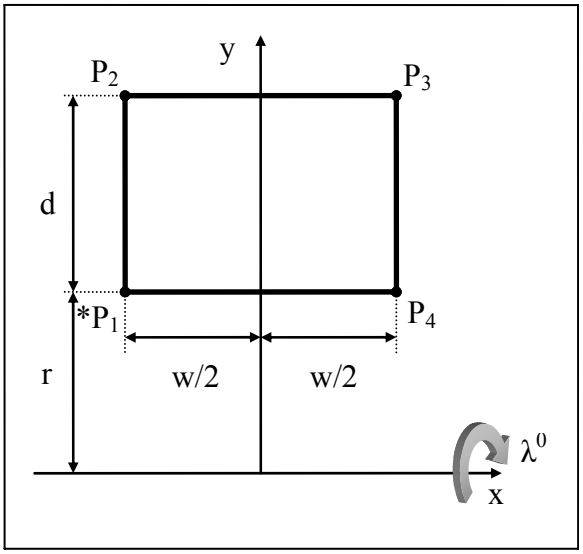

Figure B.61 Square_circular_slot_type1

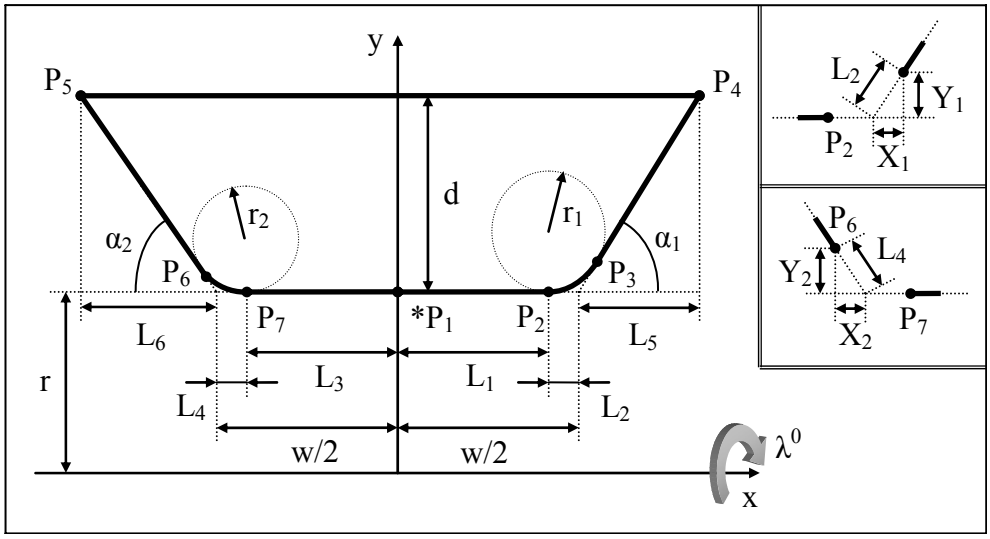

Figure B.62 Square_u_circular_slot

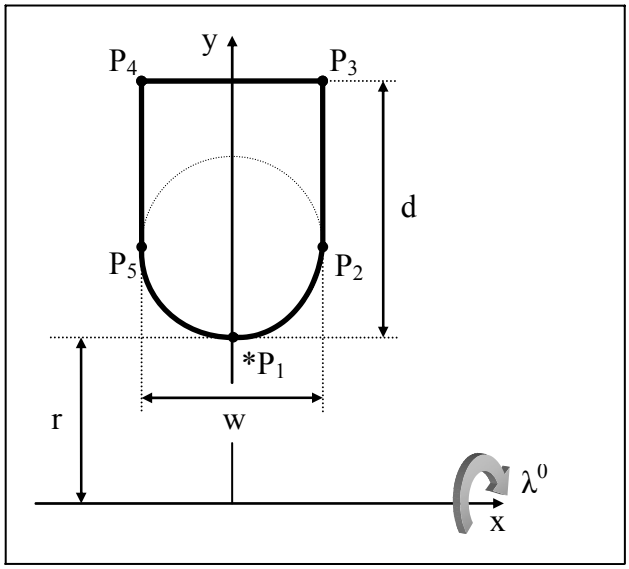

Figure B.63 Rounded_u_circular_slot

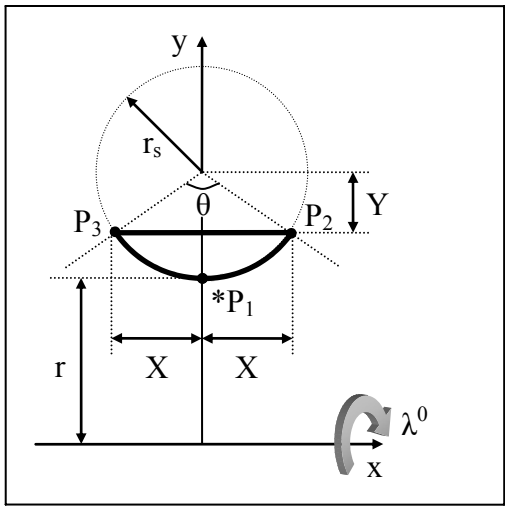

Figure B.64 Partial_circular_circular_slot

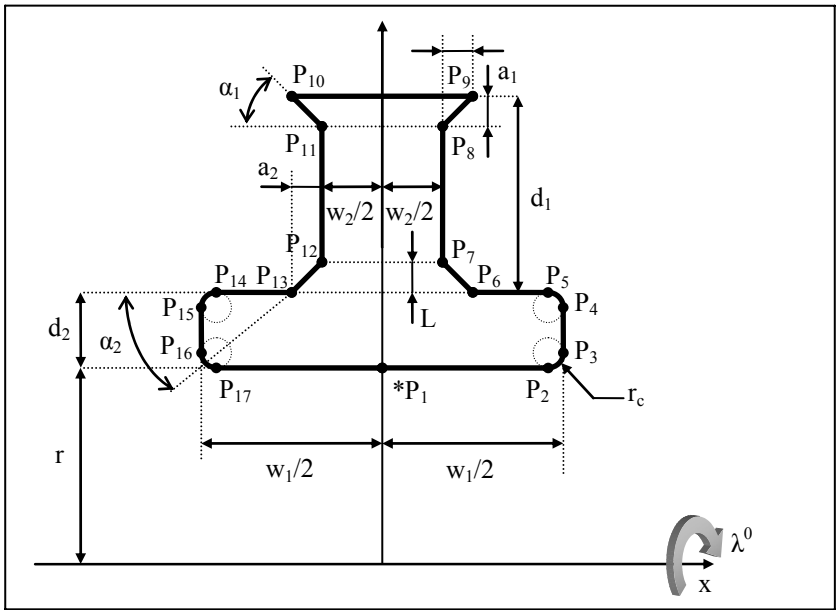

Figure B.65 Tee_circular_slot

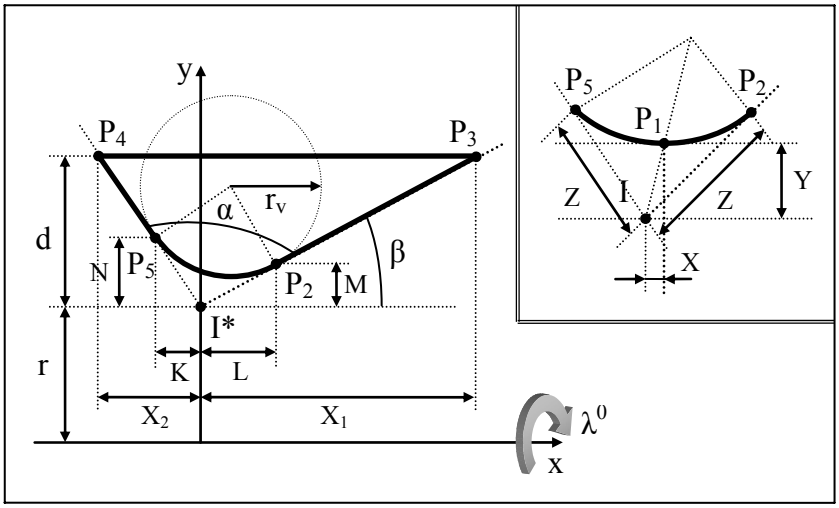

Figure B.66 Vee_circular_slot

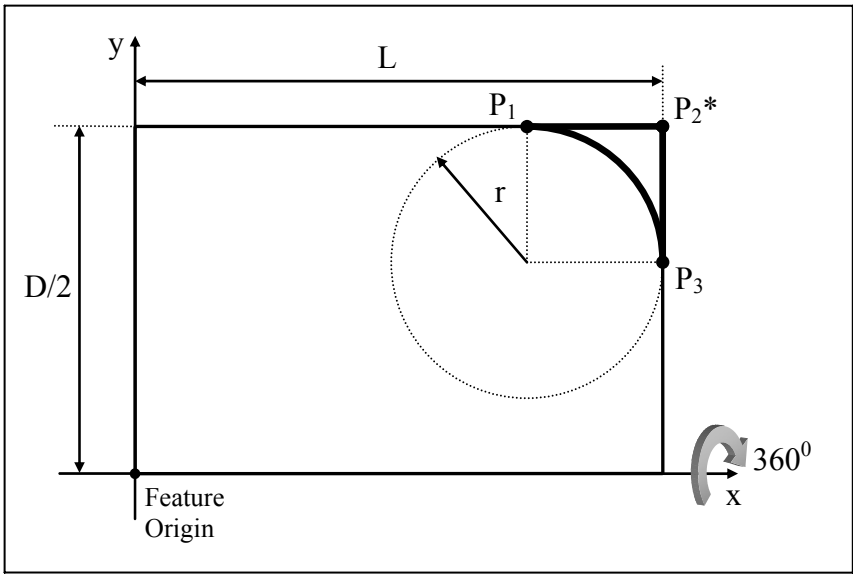

Figure B.67 Edge_round_type1

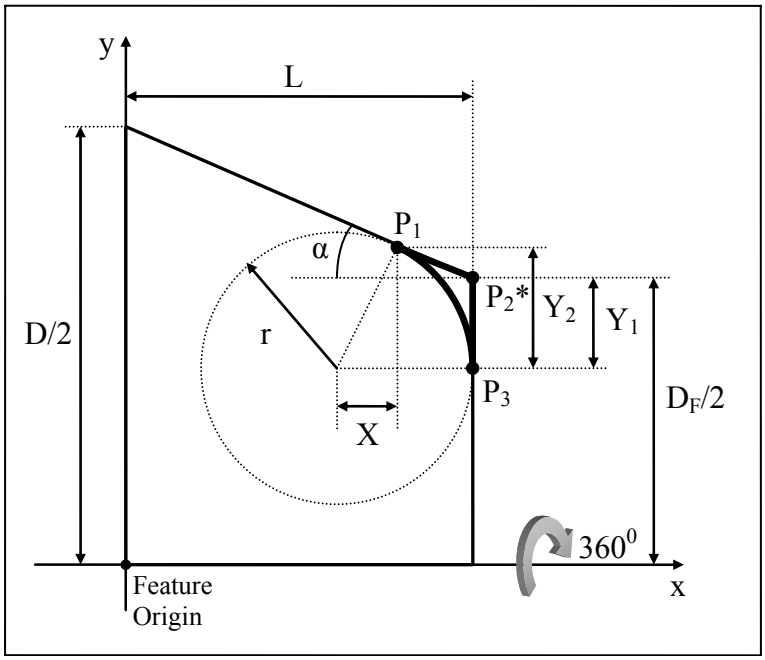

Figure B.68 Edge_round_type2

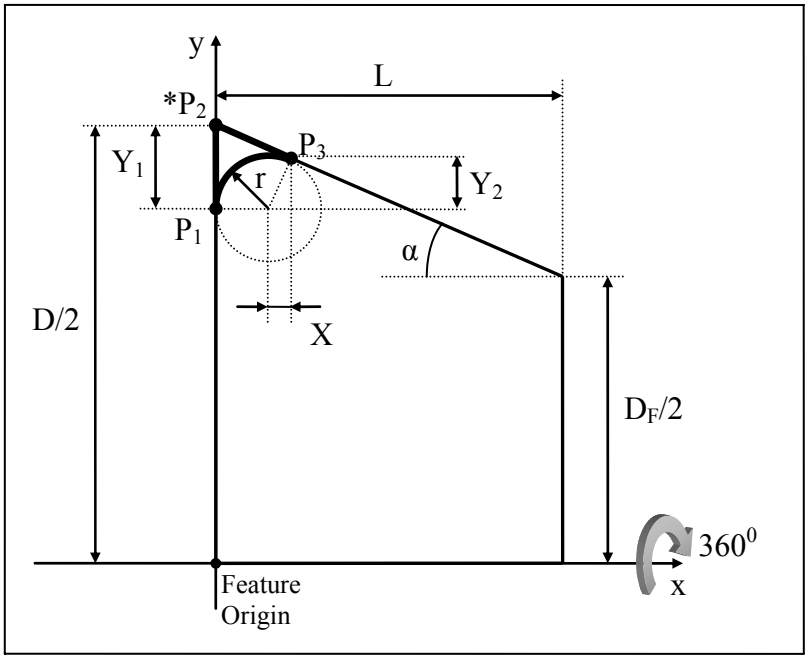

Figure B.69 Edge_round_type3

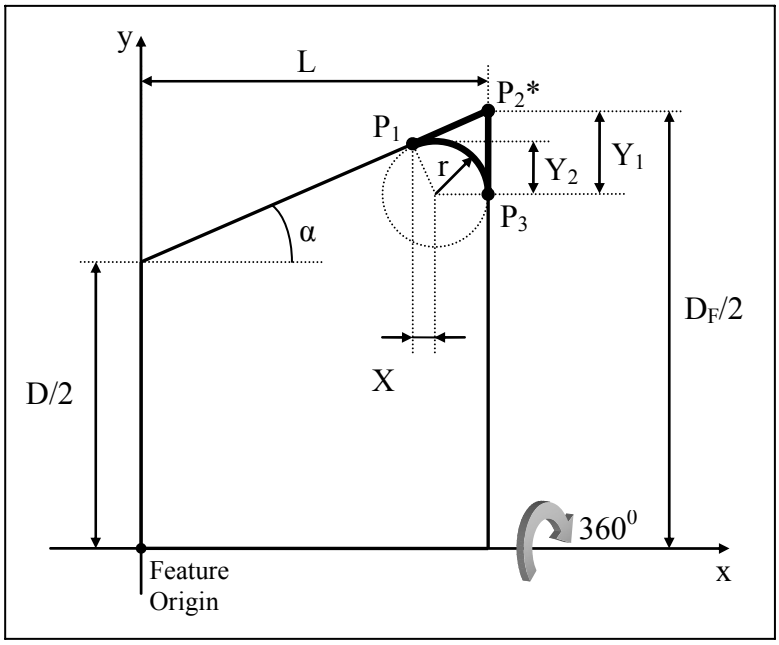

Figure B.70 Edge_round_type4

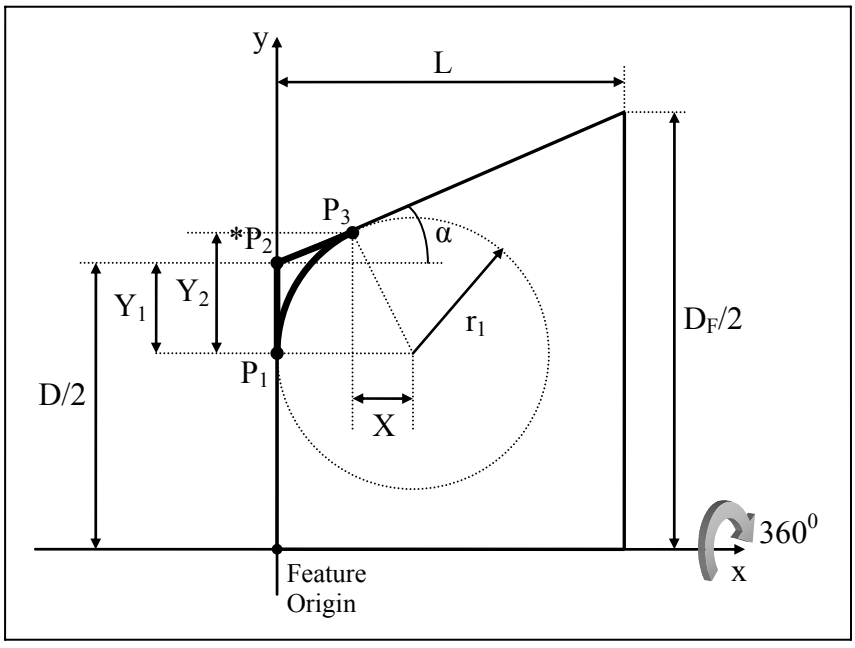

Figure B.71 Edge_round_type5

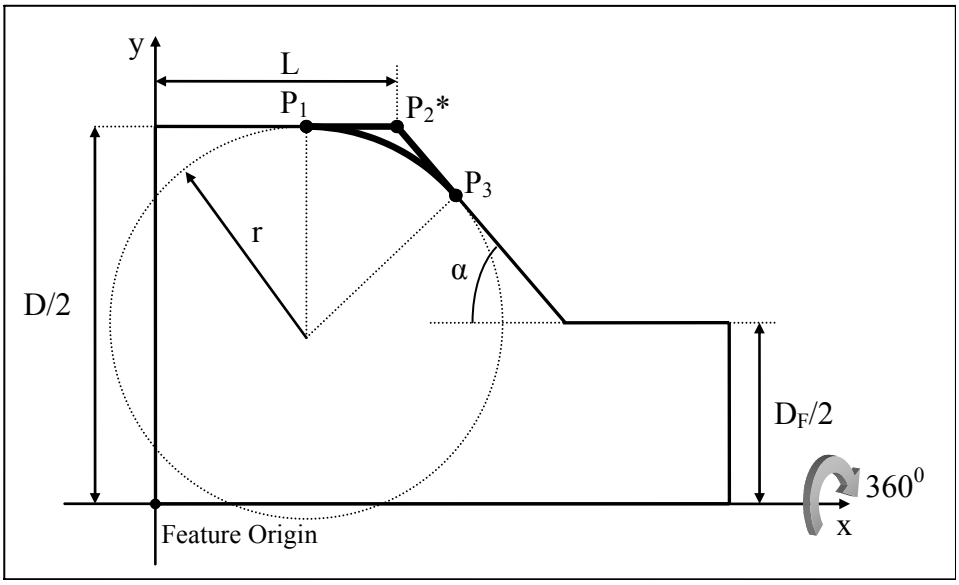

Figure B.72 M_fillet_type1

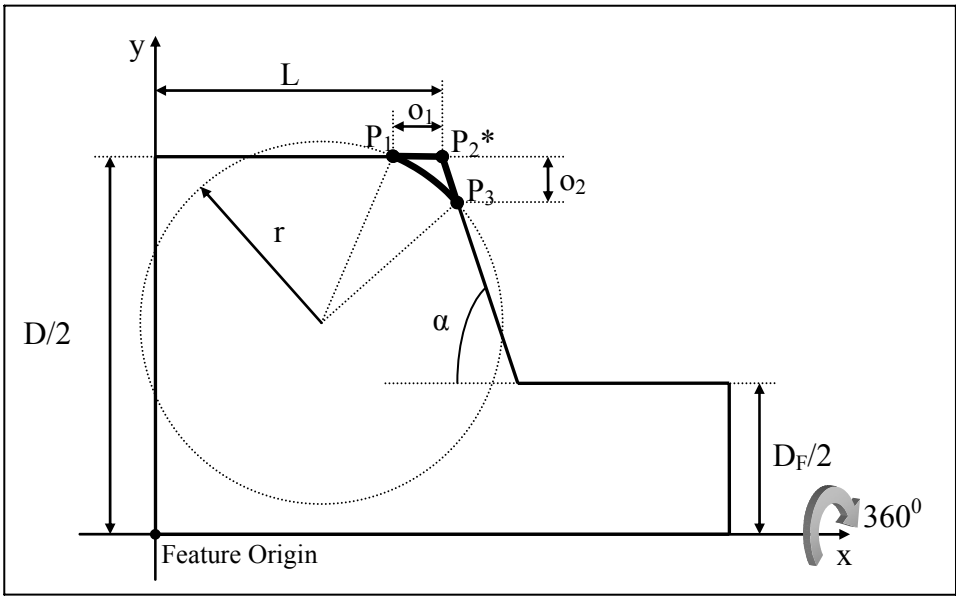

Figure B.73 M_fillet_type2

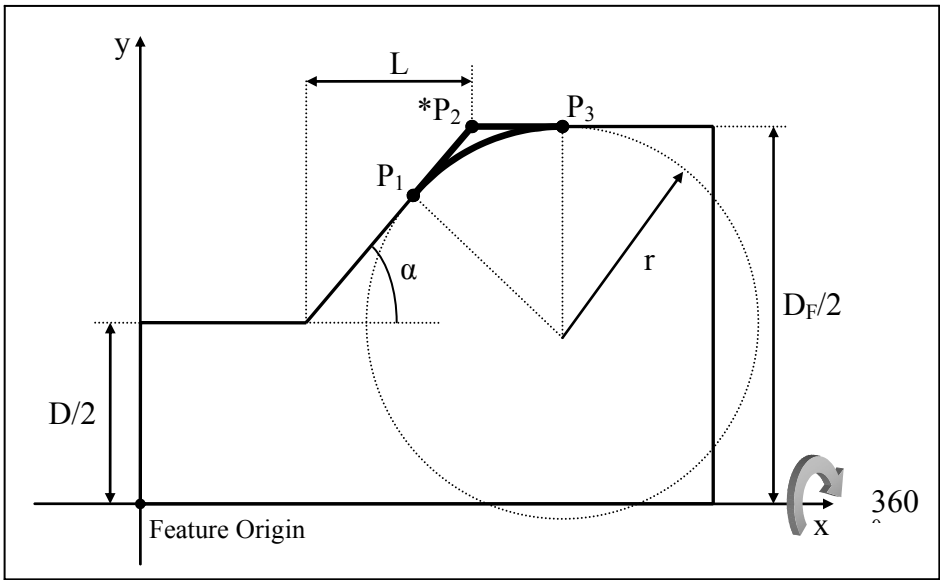

Figure B.74 M_fillet_type3

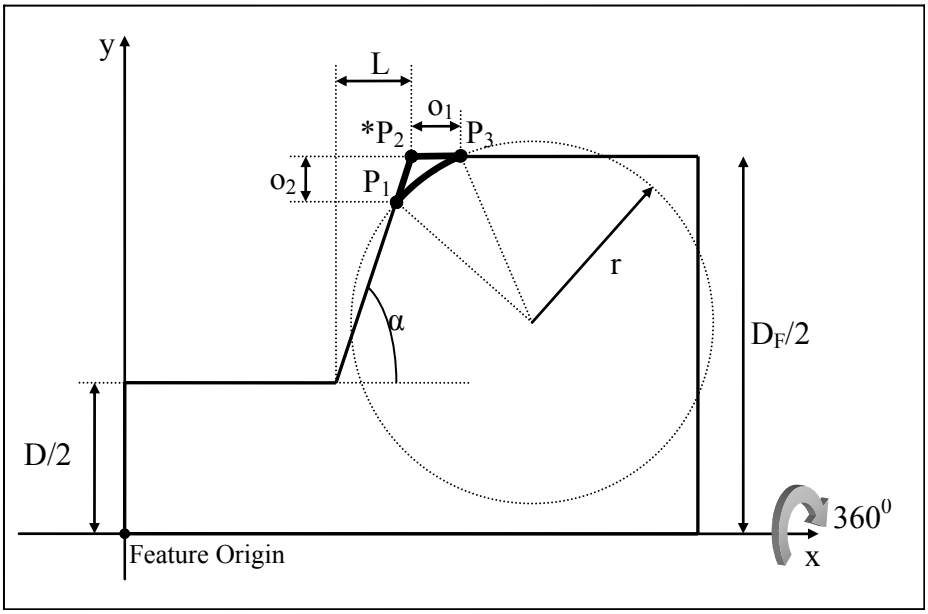

Figure B.75 M_fillet_type4

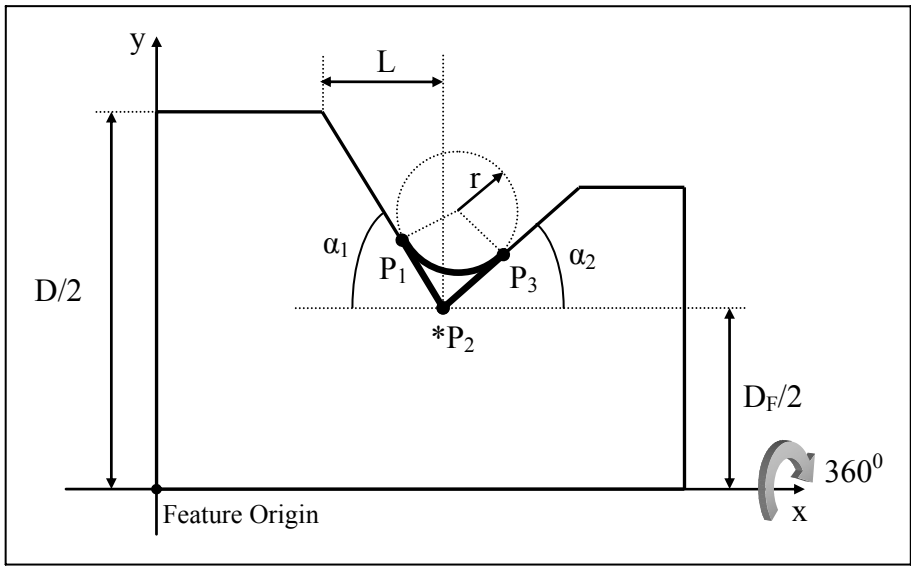

Figure B.76 M_fillet_type5

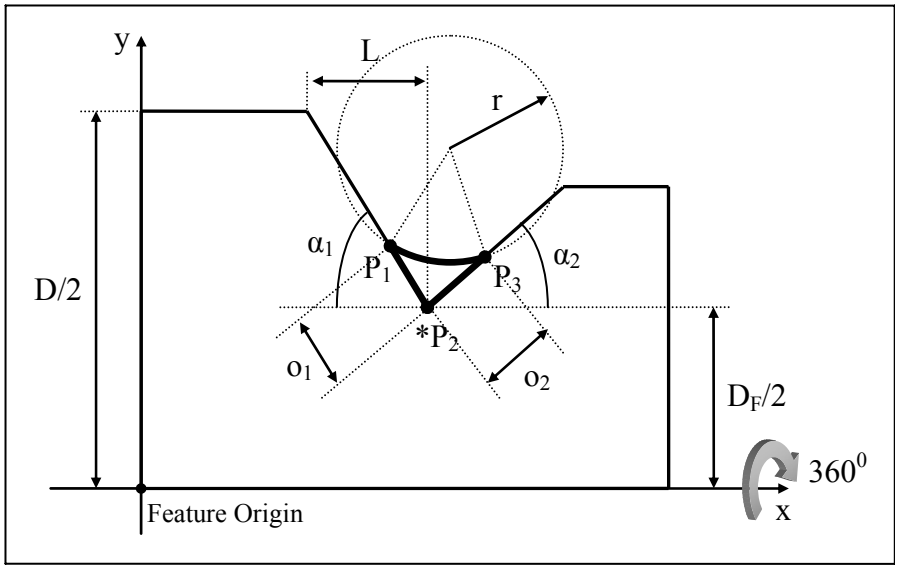

Figure B.77 M_fillet_type6

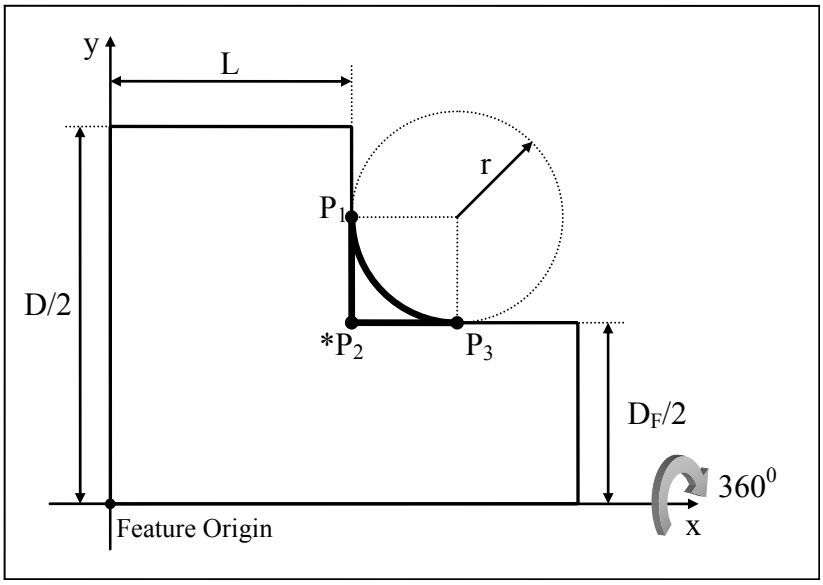

Figure B.78 G_fillet_type1

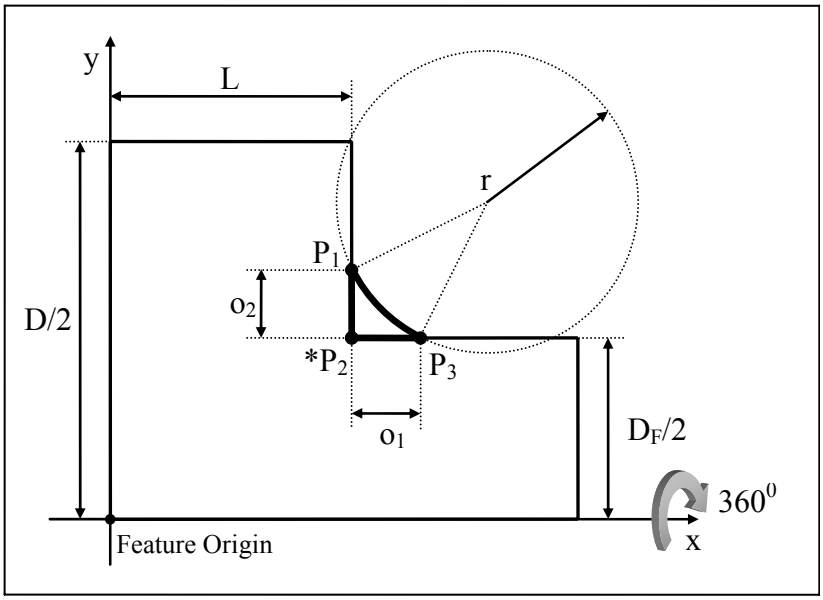

Figure B.79 G_fillet_type3

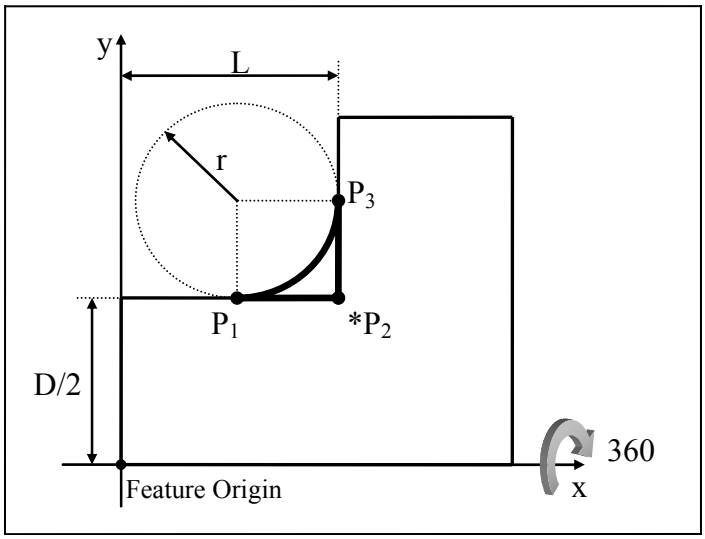

Figure B.80 Outer_tee_groove

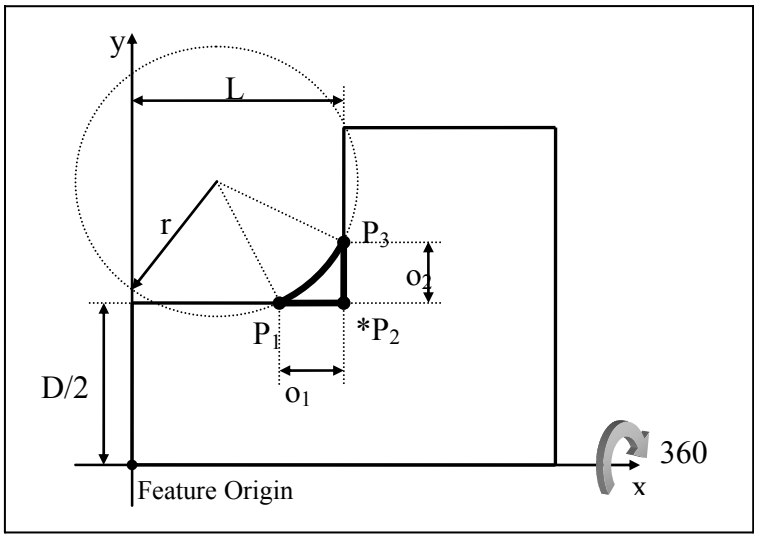

Figure B.81 G_fillet_type4

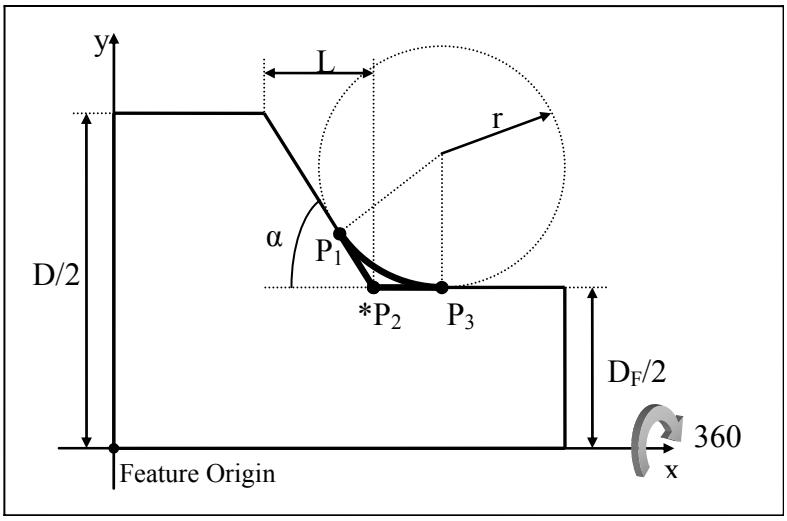

Figure B.82 G_fillet_type5

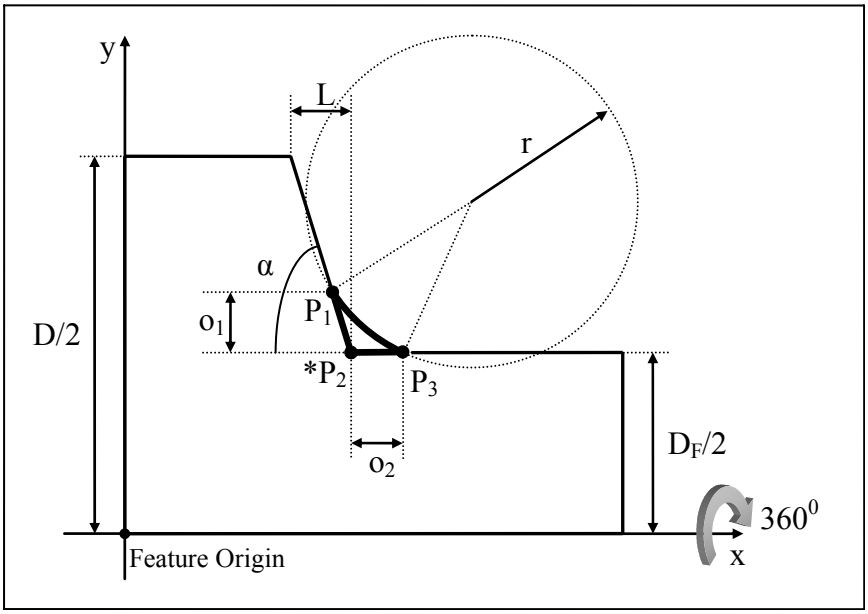

Figure B.83 G_fillet_type6

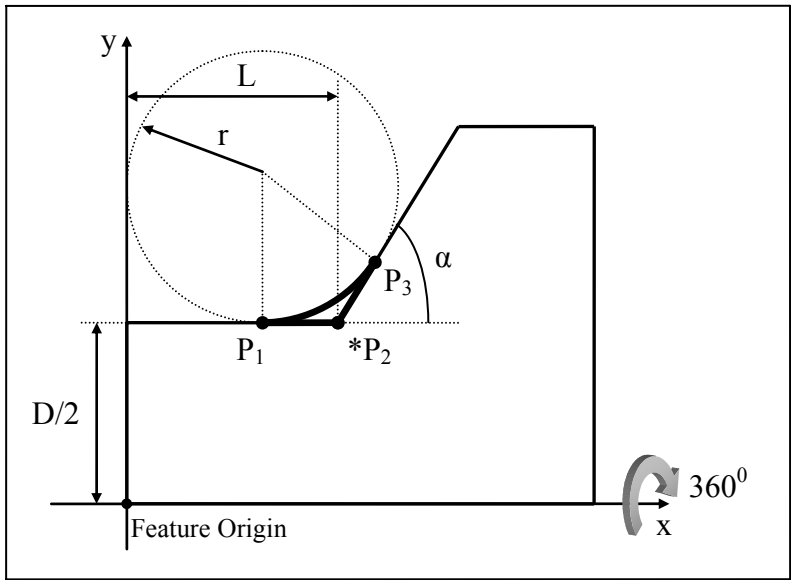

Figure B.84 G_fillet_type7

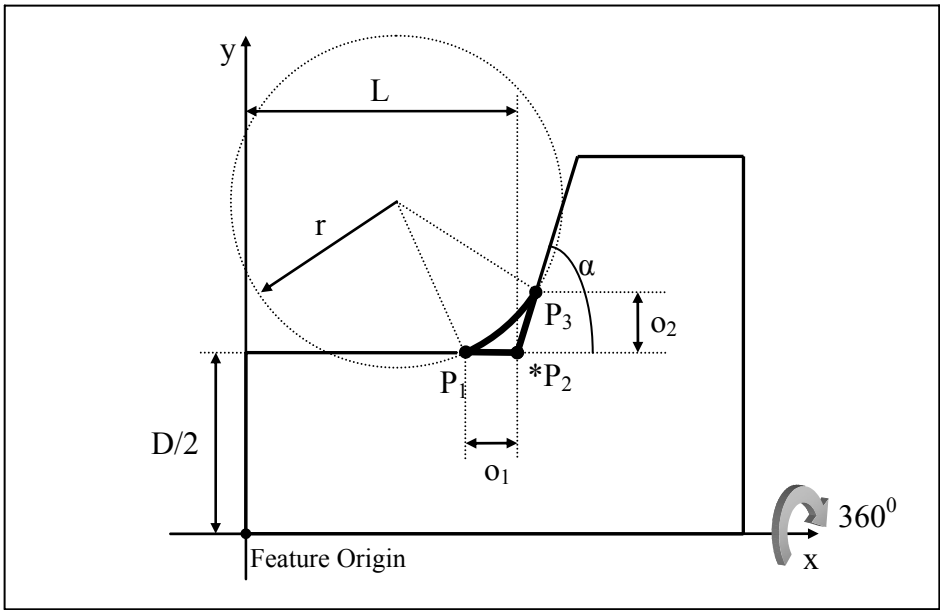

Figure B.85 G_fillet_type8

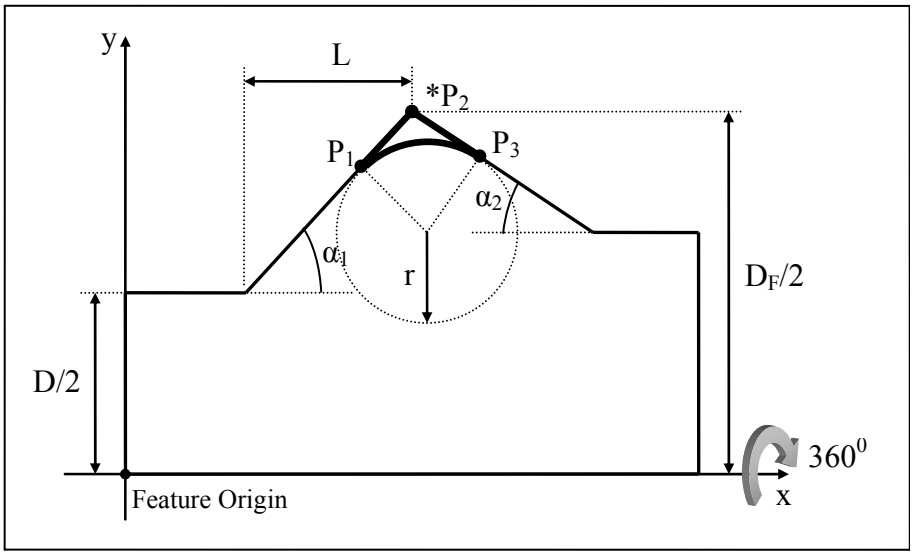

Figure B.86 G_fillet_type9

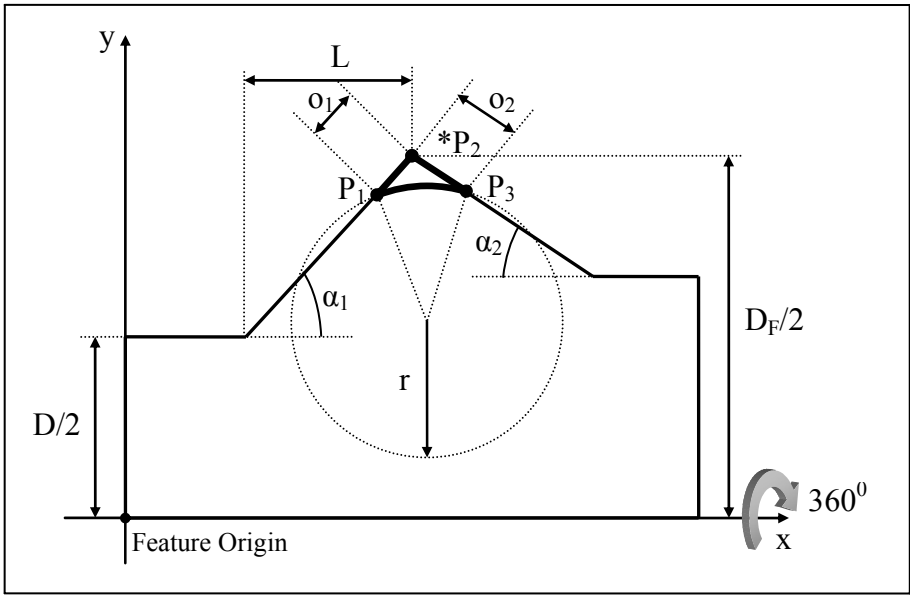

Figure B.87 G_fillet_type10