

Cimatron[®]



Cimatron
CAD/CAM Solutions for Manufacturing

Cimatron Development Kit - CimaDEK

Version 12



All rights reserved by Cimatron Ltd.

No part of this software or document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying or recording, for any purpose, without written permission from Cimatron Ltd.

Cimatron may make improvements and changes in the software described in this document at any time and without prior notice. These changes will be documented in future editions of this publication.

© Copyright Cimatron Ltd. 1984-2001

Cimatron, Cimatron^{it}, Simulator, Sketcher, CimaRender, CimaDEK, MoldBase, MoldBase 3D, Re-Enge, CPDM, MPDM, Cimagrafi and the C logo design are trademarks of Cimatron Ltd. QuickCompare, QuickElectrode, QuickConcept, QuickSplit, QuickMold and Quick Tooling are pending trademarks of Cimatron Ltd. Cimatron, Simulator, Sketcher, CimaRender, CimaRender Pro, CimaDEK, Cimagrafi, and the C logo design are registered in the U.S. Patent and Trademark Office. CimaRender and CimaRender Pro are copyrighted by Graffiti Software Industries Ltd. MoldBase 3D is based on a product copyrighted by R&B Ltd. Cimatron IMSpot is based on a product copyrighted by Intelligent Manufacturing Software, Inc. IMSpot is a trademark of Intelligent Manufacturing Software, Inc. MODView is copyrighted by Dataface, Co. Ltd.

All other company and product names are trademarks or registered trademarks of their respective owners.

All references to other trademarks and/or copyrighted material is for identification purposes and/or unintentional.

Cimatron Ltd. is not necessarily associated with any other product or vendors mentioned herein.

Disclaimer of Warranty and Liability.

No representations or warranties, expressed or implied, of any kind are made by or with respect to anything in this document.

In no event shall Cimatron Ltd., its employees or previous employees, be liable for any incidental, direct or indirect, special or consequential damages whatsoever (including but not limited to loss of profits) arising out of or related to this manual, and/or the product, or any use thereof.

Revised 2001.



Preface

Cimatron develops, markets and supports tools to automate the mechanical engineering process. Our systems support all phases of product development, with solutions for computer aided design (CAD) and manufacturing (CAM). **Cimatron's integrated technology** approach combines design tools with optimized command output to computer-controlled manufacturing equipment. Drafting-table-to-shopfloor integration lets Cimatron clients realize dramatic efficiencies in product development and manufacturing.

Cimatron^{it} - Cimatron's flagship product - covers the entire spectrum of design, engineering and manufacturing processes, including:

- A complete range of **wireframe, surface and parametric solid modeling tools** with rendering capabilities;
- Advanced **assembly, sub-assembly and part management**, and **associative drafting functionality**;
- Comprehensive, accurate **data exchange interface utilities** covering **DXF, DWG, IGES, JAMA-IS, VDA, PTC, STEP, SAT, CATIA** and **UNIGRAPHICS**;
- Powerful and intelligent **NC applications** for precise multi-axis machining.

The modular yet integrated structure of Cimatron^{it} grows to accommodate cutting edge tools and techniques. These now include the new **Quick Tooling** applications:

- **QuickSplit**

QuickSplit automates the search and separation of core, cavity and sliders to assist in determining the number of actions required to create a mold. After separating core, cavity and slides, QuickSplit identifies the parting lines and generates the parting surface.

Automatic and interactive tools allow the construction of parting surfaces for any complex geometry. Embedded Draft Analysis enables designers to identify potential problems with undercuts and confirm minimum draft per side.

QuickSplit is tolerant of surface models with gaps, mismatched boundaries or missing faces, therefore bypassing data corrections and saving precious time.

Component motion animation, dynamic cross-sectioning and clipping planes, reduce human error and verify parting design. QuickSplit enables several trial and error iterations in a very short time - resulting in optimal draw directions.

- **QuickElectrode**

QuickElectrode is an EDM electrode design solution used for shortening the electrode process. QuickElectrode is used for burn area selection, electrode design, management, documentation and manufacturing.

The QuickElectrode Navigator enables full control over the display and activation of electrodes, while allowing several users to collaborate on the same part.

QuickElectrode's report generation features includes set-up sheets, burn location

reports and a full electrode schedule, thereby alleviating the tedious task of documenting the process .

- **QuickConcept**

QuickConcept is a preliminary design and review package which allows tool designers and their suppliers to hold *virtual* review meetings over the Internet in real-time. Multiple users can connect to each other to section, label, dimension, and identify points of interest and problem areas of any given tool. All members of the review meeting will interactively view the same screen at the same time.

- **QuickCompare**

QuickCompare assists the tool designer in determining the scope and effect of Engineering Changes (ECOs) on the tooling process. QuickCompare mathematically compares the geometrical differences between two sets, graphically marks these differences and documents the changes in a CAD file. Here, the designer updates related components and tooling, while archiving ECOs. The typically long *CAD investigation process* is significantly shortened. QuickCompare ensures that all ECOs have been located, whether or not they were communicated from design.

- **MoldBase 3D**

MoldBase3D offers an innovative *wizard-based* approach to parametric mold base design. MoldBase3D automatically creates 3D solid (parametric & associative) moldbases, with all components and accessories, from industry-standard catalog suppliers such as HASCO, DME, PCS, FUTABA, DMS, PEDROTTI, RADOUREDIN, SIDECO, STRACK and MISUMI. Creation of the assembly and detailed drawings of each plate are automated, complete with 2D and 3D section views, ordinate dimensions, labels, balloons, and an itemized Bill of Materials. This module is fully associative to the mold design and changes are automatically reflected in all stages of the design process.

Cimatron's automated engineering expertise benefits many industries, as competition requires tighter development cycles and efficient fabrication.

Powerful modules within Cimatron^{it} expand your system's capabilities. These may be purchased from your Cimatron representative.

This publication provides a detailed description of the major features of the appropriate Cimatron^{it} application/topic. It is intended to help users in the daily operation of Cimatron^{it}.

A list of Cimatron^{it} documentation, for the current version, is shown on the next page.

Cimatron Documentation

Cimatron^{it} documentation comprises Reference Manuals, On-Line Help and Tutorials which together provide a comprehensive guide to Cimatron^{it}.

The list of Cimatron^{it} documentation, for the current version, is as follows:

Cimatron ^{it} Reference Manuals	Publication	Description	Display Options *
	Fundamentals & General Functions	Introduction to the fundamentals of Cimatron ^{it} and description of the General functions.	A H
	Modeling	Description of the wireframe and surface Modeling functions.	A H
	QuickSplit	QuickSplit automates the search and separation of core, cavity and sliders to assist in determining the number of actions required to create a mold.	A H
	QuickElectrode	QuickElectrode is an EDM electrode design solution used for shortening the electrode process.	A H
	QuickCompare	QuickCompare mathematically compares the geometrical differences between two models, graphically marks these differences and documents the changes in a CAD file.	A H
	Drafting	Description of the Drafting functions.	A H
	Solid Modeling	Solid Modeling functions including Sketcher.	A H
	MoldBase 3D	Description of the functions associated with the detailed design of mold plates and components. MoldBase3D offers an innovative <i>wizard-based</i> approach to parametric mold base design.	A W
	Numerical Control	Description of the NC functions.	A H
	Cimatron IMSPost	Cimatron IMSPost is a macro-based system for developing and customizing postprocessors.	A W
	General Post Processor	General Post Processor (GPP) functions.	A
	Finite Element Modeling	Description of Finite Element Modeling (FEM) functions.	A H
	Utilities	Various utilities that may be used with Cimatron ^{it} . These utilities are either Internal , run via the USER function, or External , run via the Main Menu.	A H
	Data Interface Utilities	Description of Cimatron's <i>comprehensive</i> data interface utilities; DXF, DWG, IGES, JAMA-IS, VDA, PTC, STEP, SAT, CATIA and UNIGRAPHICS.	A W
	CimaDEK	Cimatron's specialized Developer's kit, for programming customized functions.	A
	CimaRender Pro	A photo-realistic rendering package.	A W
	MPDM: Getting Started MPDM: Administrator	Description of how to use Manufacturing Product Data Management to track and organize all files and data associated with a project.	A W
	Re-Enge	Description of Reverse Engineering design functions.	A
Cimatron ^{it} Tutorials	Design - covers QuickCompare and QuickElectrode .		A H
	Drafting - covers the DMS function.		A H
	NC - covers the differences between versions 11 and 12.		A H

* Legend:

A	Acrobat PDF
H	HTML
W	Winhelp



Table of Contents

Introduction

About This Manual	Init-1
Typographical Conventions.	Init-3
How to Use this Manual	Init-4

Chapter 1 Concepts

General Concepts	1-1
The Entity ID	1-1
Classes and Types.	1-1
Coordinate Systems	1-1
Work Plane	1-1
Duplication of Entities	1-1
Select	1-2
Status	1-2
Mouse Response	1-3
Temporary and Permanent Entities	1-3
General Terms Used in This Manual	1-3
Function	1-3
Menu	1-4
User Program	1-4
USER	1-4
Directories	1-4

Chapter 2 Operations

Introduction	2-1
Creating the User Program	2-1
Umake mechanism.	2-1
Compiling and Linking the User Program	2-2
Compiling a file using the Umake mechanism	2-3
Linking using the Umake mechanism	2-3
Compiling and Linking using the Umake mechanism	2-3
Debugging	2-3
Changing compilation and link options using CimaDek System files	2-3
Running the User Program	2-4
Internal User.	2-4
External User	2-4
EDMSLink	2-4
User Program development under Windows NT/98.	2-5
Development environment.	2-5
Building a User Program with the umake mechanism	2-5
Building a User Program within Microsoft Visual C++ 5.x.	2-8
Project settings.	2-9
Example: Creating a Rectangle.	2-16

Section II User Package

Introduction	II-1
Commands	II-1
Development Authorization	II-1

Chapter 3 General Routines

ATTRIB	3-2
DISPLAY	3-15
ENTATT	3-18
INTERACTION	3-22
LEVELS	3-43
LINATT	3-48
MANAGEMENT	3-51
MATHEMATICS	3-60
PICK	3-92
PLOT	3-104
REPORTS	3-107
STATUS	3-108
STRING	3-118

Chapter 4 Modeling Routines

ANALYZE	4-2
CIRCLE	4-4
CONIC	4-10
CONTOUR	4-13
CORNER	4-18
EXPLODE	4-21
GROUP	4-22
LINE	4-28
MOVE	4-36
PLACE	4-43
PLANAR FACE	4-46
POINT	4-50
PROJECT	4-60
SPLINE	4-71
SURFACE	4-87
SWEEP	4-128
TRANSFORMATION	4-130
TRIM	4-138
UCS	4-143
VERIFY	4-152
WORKPL	4-159

Chapter 5 Drafting Routines

ANGULAR DIMENSION	5-2
CENTER LINES	5-11
CHAMFER DIMENSION	5-12
DIMENSION PARAMETERS	5-14
GEOMETRY	5-24
GEOTOL	5-29
HATCH	5-32
IDNUM	5-33
LABEL	5-34
LINEAR DIMENSION	5-40
NOTE	5-44
ORDINATE DIMENSION	5-53
RADIAL DIMENSION	5-57
STYLE	5-60
VIEWS	5-65

Chapter 6 NC Routines

START PROGRAM	6-3
OPEN TOOLPATH	6-4
GLOBAL PARAMS	6-5
TOOL POSITION	6-9
MACHINE PARAMS	6-10
CURRENT TOOL	6-17
TOOL MOTIONS	6-30
NEW MACHINE PARAMS	6-33
MARKERS	6-37
MESSAGE	6-39
TOOL	6-40
CLOSED TOOLPATH	6-45
TOOLPATH	6-46
MACSYS	6-50
PROCEDURE MANAGEMENT	6-51
TEMPLATES	6-54

Chapter 7 Solid Routines

SKETCHER	7-4
REFERENCE GEOMETRY	7-10
SOLID OPERATIONS	7-11
SOLID DATA BASE ACCESS	7-15
UTILITIES	7-23
SOLID PROCEDURE DATA ACCESS	7-28
SOLID PICK FUNCTIONS	7-31
ASSEMBLY	7-35
SOLID ATTRIBUTES	7-37
SOLID DISPLAY	7-42
SOLID EDIT	7-47
TRANSLATE WF -> SOLID	7-51

Chapter 8 External User Package

Introduction	8-1
Commands	8-1
Development Authorization	8-1
Routines for Accessing Part Files	8-2

Chapter 9 EDMSLink

Introduction	9-1
Commands	9-2
Development Authorization	9-3
Basic Level - USEREXEC Mechanism	9-3
Concept	9-3
Running the Basic Level	9-3
userexec.dat File Structure	9-3
Advanced Levels	9-4
Concept	9-4
Files Created	9-5
Service Requests	9-6
Running the Advanced Levels	9-7



Introduction

The **Cimatron Development Kit, CimaDEK** is intended for users of Cimatron^{it} and works in conjunction with it. **CimaDEK** is used to develop various applications such as User Package, NC User, Solid Routines, External User Package and EDMSLink applications.

User Package	Routines for developing interactive programs which run within Cimatron ^{it} .
NC User Package	Routines for developing interactive NC programs which run within the Cimatron ^{it} NC application.
Solid Routines	Routines for developing interactive programs which run within the Cimatron ^{it} Solid application.
External User Package	Routines for developing external user programs which run outside Cimatron ^{it} .
EDMSLink	Engineering Data Management Systems, which can be developed and run as either an internal or external user program.

CimaDEK includes a collection of hundreds of routines within the application areas mentioned above. These routines are used write application programs to be used in the Cimatron^{it} environment.

It is assumed that you have an overall knowledge of the concepts of the **C** Programming Language, although you need not necessarily be an experienced programmer.

This manual explains the steps to follow in order to build a “user procedure” which can be used in conjunction with Cimatron^{it}.

An in-depth explanation of **C** is not supplied in this manual. Therefore, it is recommended that you have a copy of a **C** Reference Guide, as well as the Cimatron^{it} **Utilities Manual**, available when working with **CimaDEK**.

About This Manual

This manual describes **CimaDEK** in depth and consists of nine chapters.

SECTION I, General Operating Instructions:

Chapter 1	Describes the general concepts of CimaDEK .
Chapter 2	Explains how to use CimaDEK .

SECTION II, User Package:

Chapter 3	General user routines associated with various activities within Cimatron ^{it} .
Chapter 4	Modeling (Wireframe and Surface) routines associated with the Modeling application of Cimatron ^{it} .

Chapter 5 Drafting routines associated with the Drafting application of Cimatron^{it}.

SECTION III, NC User Package:

Chapter 6 NC routines associated with the NC application of Cimatron^{it}.

SECTION IV, Solid Routines:

Chapter 7 Solid routines associated with the Solid application of Cimatron^{it}.

SECTION V, External User Package:

Chapter 8 Describes the External User Package (EUSYS).

SECTION VI, EDMSLink:

Chapter 9 Describes the method of linking Cimatron^{it} to the EDMS, using EDMSLINK.

Chapters 3 through 7 are further divided into subsections (in alphabetical order) logically corresponding to the interactive functions of the system. Each of these describes the routines appearing within it in alphabetical order.

At the beginning of each subsection a general description may be found. This consists of instructions to be followed when using these routines (where appropriate).

To use a subroutine within a C program, use the following command:

e.g. GREXPL (&idinst, &level, &maxnum, &numid, idbuf, &status);

Typographical Conventions

Throughout this manual, certain conventions have been used to present different types of information.

The description of each function includes a title line, the purpose of the function, a syntax line and function prototype. Notes have been added in some functions to clarify or stress important points.

For example:

EQPT2

Check whether two 2D points are equal.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQPT2 ( P1, P2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float P1[2];
float P2[2];
/* Two 2D points.
/*
/* Returns :
/* .TRUE.
/* Points are equal.
/* .FALSE.
/* Points are not equal.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

**End of
Chapter/
Subsection**

The end of a chapter or subsection is marked by a box character (□).

How to Use this Manual

Before attempting to create a **CimaDEK** application program from the routines found in this manual, you are advised to follow the examples in Chapter 9 carefully.

Once you have attempted the examples, and successfully run them without receiving error messages, you will be ready to use the manual to its full extent.

To find the appropriate routine for the task you wish to perform, first decide which of the **CimaDEK** applications it belongs to, and if, for example it is a User Package routine, decide whether it belongs in General, Modeling or Drafting. (Consider under which function a similar task would be performed in the interactive system.) Then decide which topic it falls under, and turn to the relevant page.

In the general description of the topic, you will find a brief explanation of each of the routines. Choose the appropriate routine(s) for your task, taking care to follow any set procedure described in the General Description before attempting to use the routine itself. Turn to the routine header for details of its arguments and build your routine accordingly. If you are unsure as to where you might find the appropriate routine, refer to the Table of Contents for the task you wish to perform. Routines are listed alphabetically within each task.

As you build your program, bear in mind all general C rules.

Note:

- While working with **CimaDEK** you may come across options that are illegal in the interactive system. It is your responsibility to check the safety of all actions performed by your program! □

Section I

General Operating Instructions



Chapter 1

Concepts

General Concepts

Many of the terms appearing in this manual may be unfamiliar to users of the interactive Cimatron^{it} system, simply because they have never had to deal with the internals of the system. However, the programmer using the routines in this package needs an overall knowledge of these concepts. This Chapter provides an explanation of the most important concepts mentioned in this manual.

The Entity ID

As in the interactive system this package works with the logical concept of an entity. Each entity has an internal identification number associated with it. This number is known as the ID number of the entity and uniquely identifies each entity within the database.

The variables containing the identification numbers of entities are of type **INTEGER**. Therefore, when a routine that creates an entity within **CimaDEK** is used, an integer value (the ID number) for that entity is created by the system.

The actual values of these variables are unimportant to the user of this package and are not retained from one session to another. They are simply there to be used as input and/or output parameters within the routines of the User Package.

Classes and Types

For information on Classes and Types, contact your Cimatron Provider.

Coordinate Systems

CimaDEK provides access to two coordinate systems. The Work Coordinate System and the Model Coordinate system. All coordinates and geometrical values appearing as input parameters refer to the current Work Coordinate system unless otherwise specified in the routines.

Work Plane

The work plane orientation is defined by the active Coordinate System. Its origin is the Model Coordinate system origin.

Duplication of Entities

Some of the **USER** programs, when called, may result in the modification or duplication of an entity.

Duplication will only occur if the original entity was created in a view that is different from the current view. In such a case, the original entity will remain

untouched and the new resulting entity will be built by duplicating the original and adding the changes to it.

For example, using the routine ENCHA1 (see LINATT, Chapter 3), we may change line attributes of an existing entity. If these changes are not performed in the view in which the entity was created, the original entity will not be changed. The resulting entity will be a copy of the original, but with the new attributes.

Select

SELECT is a string variable that appears as an input parameter in several USER routines. It is used to select the desired solution to a geometrical problem where there may be more than one valid solution.

For example, the intersection between a line and a circle may be one of two points. SELECT allows you to choose which point you require.

One of the following strings may be assigned to SELECT:

XLARGE	The solution for which X is largest.
XSMALL	The solution for which X is smallest.
YLARGE	The solution for which Y is largest.
YSMALL	The solution for which Y is smallest.
ZLARGE	The solution for which Z is largest.
ZSMALL	The solution for which Z is smallest.

Status

STATUS is an integer variable that appears as an output parameter in many USER routines. It returns an integer value describing the result of executing the routine. The normal case is STATUS = 0, signifying that the execution ended correctly with no errors. STATUS > 0 signifies warnings, and STATUS < 0 signifies serious errors.

Note: • Always check the STATUS value after running a routine.

The following table defines these codes in most cases:

CODE	EXPLANATION
2	Missing information, some default values were used.
1	There are no geometrical solutions.
0	Routine ended correctly.
-1	Fatal error. In most cases, database problems (cannot create an entity).
-2	Fatal error. In most cases, database problems (cannot open an old entity).
-n(11£n£29)	Fatal error. The argument is out of range, where n = 10 + argument number in argument list.
-n(31£n£59)	Fatal Error. The argument is an entity of the wrong database type, where n = 30 + argument number in argument list.

Note: • An argument number indicates the position of the argument in the subroutine. The fifth argument in the subroutine will be argument number 5. If it is out of range, **n** will be equal to 15. If it is an entity of the wrong type, **n** will be equal to 35.

Mouse Response

RESPON is an integer variable that appears as an output parameter in some USER routines.

The mouse may be used in four different ways to respond to a system prompt: SELECT from a menu or PICK an entity, EXIT, REJECT and SUBMENU. RESPON is assigned one of the following values according to the mouse response:

SELECT or PICK	= $n (n \neq 0)$	The significance of the value of n when it is greater than or equal to zero is explained in each subroutine in which it appears.
EXIT	= -1	
REJECT	= -2	
SUBMENU	= -3	

Temporary and Permanent Entities

Every complicated construction results in the creation of permanent and temporary entities. Permanent entities are stored in the data base, whereas temporary entities are only required during the creation of the model or during drafting.

CimaDEK provides two methods of defining permanent entities.

a. Call DBFLOF This routine switches off DBCLEA and makes all entities permanent. If DBFLOF is used, all unwanted entities have to be deleted, entity by entity, by using the routine DLDELE. DBCLEA is switched on again by using DBFLON.

b. Call DEFADD This routine inputs a list of entities and marks all these entities as permanent. Temporary entities are deleted by DBCLEA.

Call the routine TPLCLR after calling DBCLEA. TPLCLR deletes the contents of the temporary entity buffer.

General Terms Used in This Manual

You will find the following terms among the pages of this manual. Each of these general terms has a specific meaning within the context of this manual and care must be taken to differentiate between their meanings.

Function

The word “function” as it appears in this manual refers to a CimaDEK function.

When reference is made to a Cimatron^{it} function, this indicates a function within the interactive Cimatron^{it} system, e.g., the Cimatron^{it} function POINT.

All functions of type float or double should be declared, apart from Preprocessor Command Definitions (macros), or use prototype files (see **Prototypes Files** in Chapter 2).

Menu

The word “menu” refers to the list of Cimatron^{it} functions.

User Program

A program that is written with the functions described in this manual.

USER

The word “USER” (in uppercase) refers to the Cimatron^{it} USER function.

Directories

<root_cad> is the name of the Cimatron^{it} root directory.

<root_cad>\var\user is the name of the directory produced by the installation process and in which all the user-built programs are stored.





Chapter 2

Operations

Introduction

This chapter describes the general process to be followed for compiling, running and debugging a User Program on the system. An example can be found at the end of this chapter.

This chapter contains the following sections:

Creating the User Program.

Compiling and/or Linking the User Program.

Running the User Program.

Development Platforms :

Win32
Hewlett Packard
Sun
Silicon Graphics

Prototypes.

Include Files.

Using Fortran.

Example of a User Program

Creating the User Program

Create a C source User Program with the aid of any available editor. Your source file name must end with the appropriate suffix expected by your C compiler (e.g .C).

Umake mechanism.

The **umake** mechanism allows user programs to be built in the same manner, independently of the hardware used. **umake** commands are based on the appropriate compiler and linker and provide sequential building of the user program according to the input data.

The purposes of the **umake** mechanism are the following:

- compile source file, via the command line.
- create and compile the **main.c** file.
- add all relevant user files (command line or **.dat** file) to the link list.
- add relevant CimaDEK libraries to the link list.
- build a User Program and place it in the Cimatron[®] user directory.
- prepare a User Program for debugging (if required).

Compiling and Linking the User Program

To get a brief description of **umake**, run **umake** without arguments.

The output will be :

SYNOPSIS **umake** user_program [-cgl] [-ver number] [-ddate] [obj1 ... objN] [dat1 ... datN]

where user_program is the name of User Program (no extension).

OPTIONS

- c** - “compile only” option.
- g** - compile with debugging information.
- l** - “link only” option.
- ddate** - display the date and time of the build on the background window at run time.
- ver number** - display the version number on the background window at run time.

OBJECTS

- objs** - pathnames of additional objects / libraries to be included in the link.
- datas** - pathnames of *.dat list files to be included in the link.

Compiling and Linking the User Program

The command to compile and/or link the User Program depends upon the Development Kit you are working on. The table below shows the available options:

Development Kit	Commands		
	Compile & Link	Compile	Link
User Package	umake	umake -c or ucomp	umake -l or ulink
NC User Package	ncumake	ncumake -c or ncucomp	ncumake -l or nculink
External User	eumake	eumake -c or eucomp	eumake -l or eulink
EDMSLink Internal	lumake	lumake -c or lucomp	lumake -l or lulink
EDMSLink External	leumake	leumake -c or leucomp	leumake -l or leulink

Syntax examples below are based on the User Package, however, they apply to the whole Development Kit.

Note:

- The commands **umake**, **ucomp** and **ulink** run the executable file **makusr** with the appropriate option. **makusr** should not be accessed directly.

Compiling a file using the Umake mechanism

```
umake <file> -c
ucomp
```

Linking using the Umake mechanism

```
umake <file> -l [obj1 .. objn] [dat1 .. datn]
ulink <file> [obj1 .. objn] [dat1 .. datn]
```

Compiling and Linking using the Umake mechanism

```
umake <file> [obj1 .. objn] [dat1 .. datn]
```

where:

obj[1..n] Objects and/or libraries to be included in the link (full path names),

dat[1..n] Data files (full path names) containing names of objects and/or libraries to be included in the link.

If an object or data file is not specified, only system libraries from the **<root_cad>\bin** directory will be included in the link.

.obj and **.dat** are two optional parameters which enable specific objects and/or libraries to be included in the link in addition to system libraries. If many objects are to be included in the link, enter their names into a data file and use the **.dat** option.

Rules for writing data files

Lines beginning with a hash (#) character are comments.

When a line begins with **cimit** <file_name>, the mechanism will search for the file in the directory **<root_cad>\bin**.

Debugging

To prepare the program for debugging, it should be compiled and linked with the option -g (i.e. **umake <file> -g**).

Changing compilation and link options using CimaDek System files

The umake mechanism uses the following three files. These files can also be edited :

ccomp Contains a compilation script.

ulist.dat Contains a link list.

ulist0.dat Contains link editor options.

These files enable the user to define compilation and link options.

The search path for these files is :

Running the User Program

Current directory,
<root_cad>\var\profiles\<user>,
<root_cad>\var\profiles\<group>
<root_cad>\var\<usys, eusys>

To modify a system file, we recommend the following: copy the file from <root_cad>\var\usys, eusys to another location in the search path, and edit the copy.

Running the User Program

Internal User

To access a User Program, enter the interactive system in the normal way and select the **USER** function. The following will appear in the prompt and interaction areas at the top of the screen:

INVOKE USER PROGRAM (<CR> = <last used function name>)

There are a number of ways to run an internal User Program:

- Press <CR> to accept and run the last used User Program.
- Enter the name of a different User Program.
- Press <SUBMENU> to display the names of the available USER package User Programs, and select a User Program.

The name of the User Programs is displayed in place of the **USER** function button in the Current Path Area function bank. The User Program is loaded and begins running.

External User

The External User (non interactive) is run via the system prompt. Enter the User Program name at the system prompt.

EDMSLink

For the Internal EDMSLink, run the User Program as for the Internal User. For the External EDMSLink, run the User Program as for the External User.

User Program development under Windows NT/98

Development environment

Operating system : Windows NT 4.x or Windows 98.

Compiler : Microsoft Visual C++ 5.0.

Building a User Program with the umake mechanism

To build a User Program using the umake mechanism, perform the following steps:

How To:

1. Set environment variables.
2. Run Command Prompt.
3. Using the Command Prompt window, run umake with the appropriate arguments and options.

Environment variables

To set environment variables run Settings / Control Panel / System / Environment.

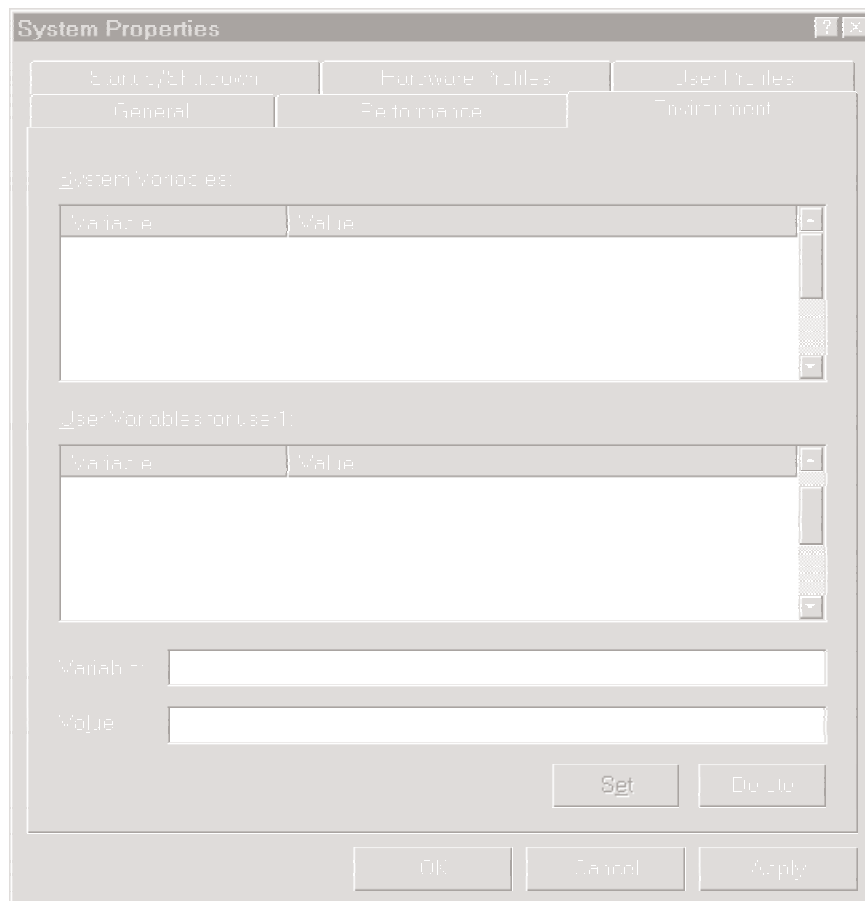


Figure 2-1: Environment Variables

Use the User Variables box to set following variables:

PATH	<p>The path variable should contain paths to:</p> <p><root_cad>\bin; <root_cad>\var\usys; <msdev>\vc\bin; <msdev>\SharedIDE\bin;</p> <p>Example:</p> <p>path =d:\winnt\system32;d:\msdev5\vc\bin; d:\msdev5\SharedIDE\bin; d:\cimit\bin; d:\cimit\var\usys;</p>
INCLUDE	<p>The include variable should contain paths to:</p> <p><root_cad>\inc; <msdev>\vc\include;</p> <p>Example:</p> <p>include=d:\msdev5\vc\include;d:\msdev5\vc\atl\include; d:\msdev5\vc\mfcl\include;d:\cimit\inc;</p>
LIB	<p>The lib variable points to the <msdev>\vc\lib;</p> <p>Example: lib=d:\msdev5\vc\lib;d:\msdev5\vc\mfcl\lib;</p>
MSDevDir	<p>The MSDevDir variable points to the Shared IDE location.</p> <p>Example: MSDevDir=d:\Msdev5\SharedIDE</p>
CPU	<p>The CPU variable should be set according to the machine type:</p> <p>IX86 MIPS ALPHA PPC MPPC</p> <p>Example: cpu=ix86</p>
MSDEV	<p>The MSDEV variable points to the Microsoft Visual C++ 5.x location.</p>

Umake arguments and options.

The Umake mechanism permits all of its options to be used under Win32 platforms. The structure of the .dat file is the following:

<file_name>.obj

...

<file_name>.lib

...

```

D:\users>umake rectan
Compiling USER program rectan.c . . .
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 11.00.7022 for 80x86
Copyright (C) Microsoft Corp 1984-1997. All rights reserved.

rectan.c
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 11.00.7022 for 80x86
Copyright (C) Microsoft Corp 1984-1997. All rights reserved.

main.c
Microsoft (R) 32-Bit Library Manager Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.

    Creating library tmprectan.lib and object tmprectan.exp
Linking USER program d:\cim90nt\var\user\rectan.dll . . .
Microsoft (R) 32-Bit Incremental Linker Version 5.00.7022
Copyright (C) Microsoft Corp 1992-1997. All rights reserved.

main.obj
rectan.obj
d:\cim90nt\bin\usys.lib
kernel32.lib
user32.lib
gdi32.lib
comdlg32.lib
msuport.lib
oldnames.lib
-force:multiple
-base:0x1C000000
-dll
-entry:LibMain@12
-NODEFAULTLIB
-ignore:4006
-out:d:\cim90nt\var\user\rectan.dll tmprectan.exp

D:\users>

```

Figure 2-2: Example of the creation of rectan using umake.

Building a User Program within Microsoft Visual C++ 5.x

To compile, link and debug User Program within Microsoft Visual C++ 5.x, the following stages are required:

How To:

1. Create a new Microsoft project
2. Customize appropriate settings.
3. Create a .def file.
4. Add all relevant files to the project

Note: • A complete example can be found on the Cimatron Internet site (see page 2-28).

Create project.

From the File menu select New (Ctrl+N). The following dialog box appears:

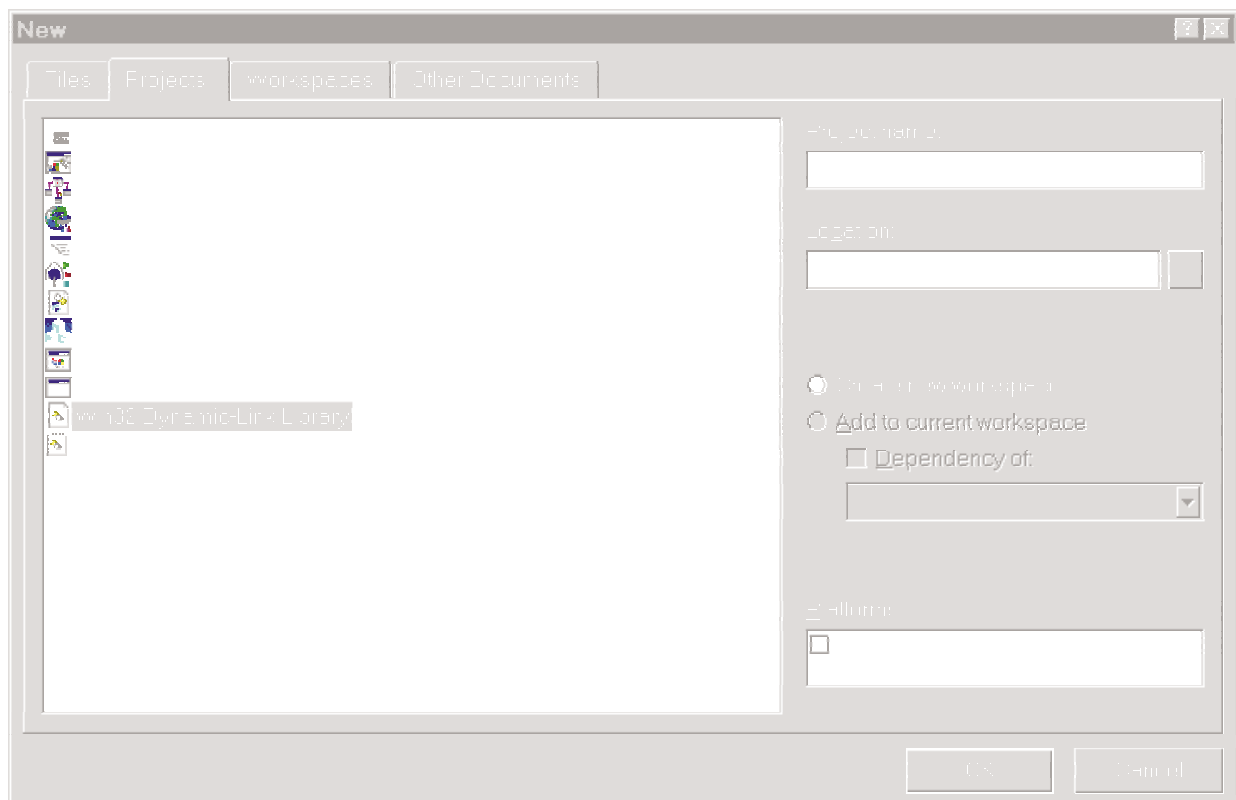


Figure 2-3: Initial Project Creation Dialog

Select the Projects tab.

Select Win32 Dynamic-Link Library from the list.

Set name of the project and its location.

Press OK.

Project settings.

From the Project menu select Settings... (Alt+F7). The following dialog box appears:

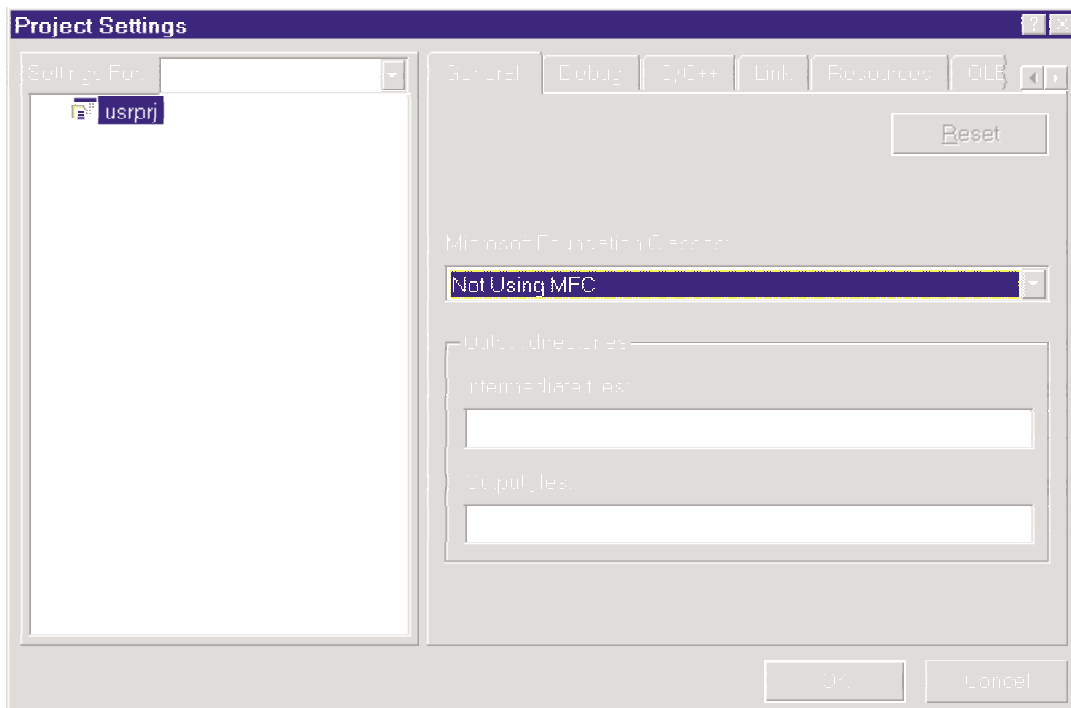


Figure 2-4: Projects Setting Dialog

Select the Debug tab.

In the field Executable for debug session, enter the full path to the Cimit.exe.

Fill in the fields Working directory and Program arguments (if needed).

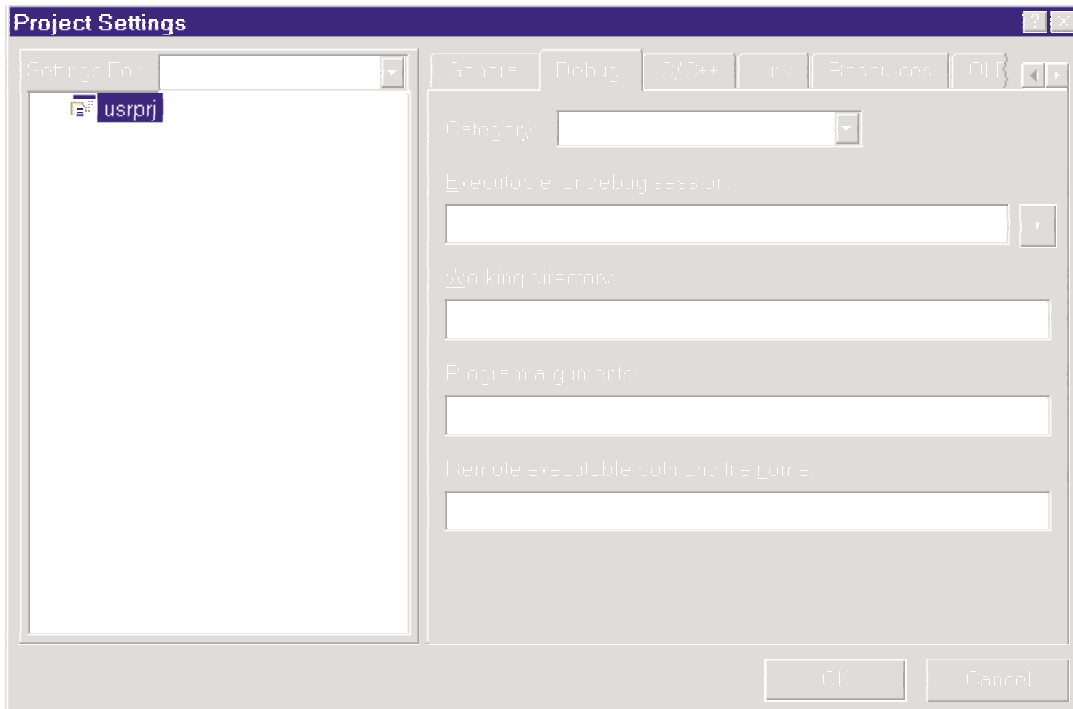


Figure 2-5: Debug Tab

Select the C/C++ tab.

Select the category Code Generation and change the Use run-time library to Multithreaded DLL.

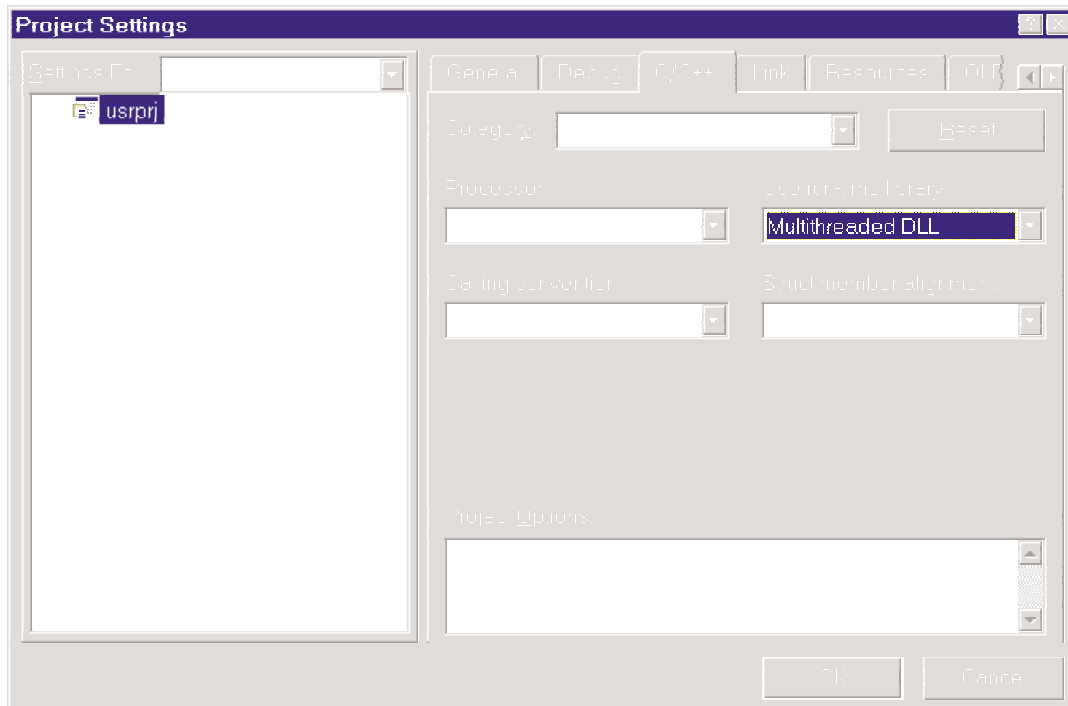


Figure 2-6: C/C++ Tab

Select the category Preprocessor.

Add macros and constants to the Preprocessor definitions field according to the **..var\usys\ccomp.bat** file:

CRTAPI1=_cdecl, CRTAPI2=_cdecl, _X86_=1, WIN32,_WIN32,_MT,_DLL, NT=11;

Enter the directory <root_cad>\inc in the field Additional include directories.

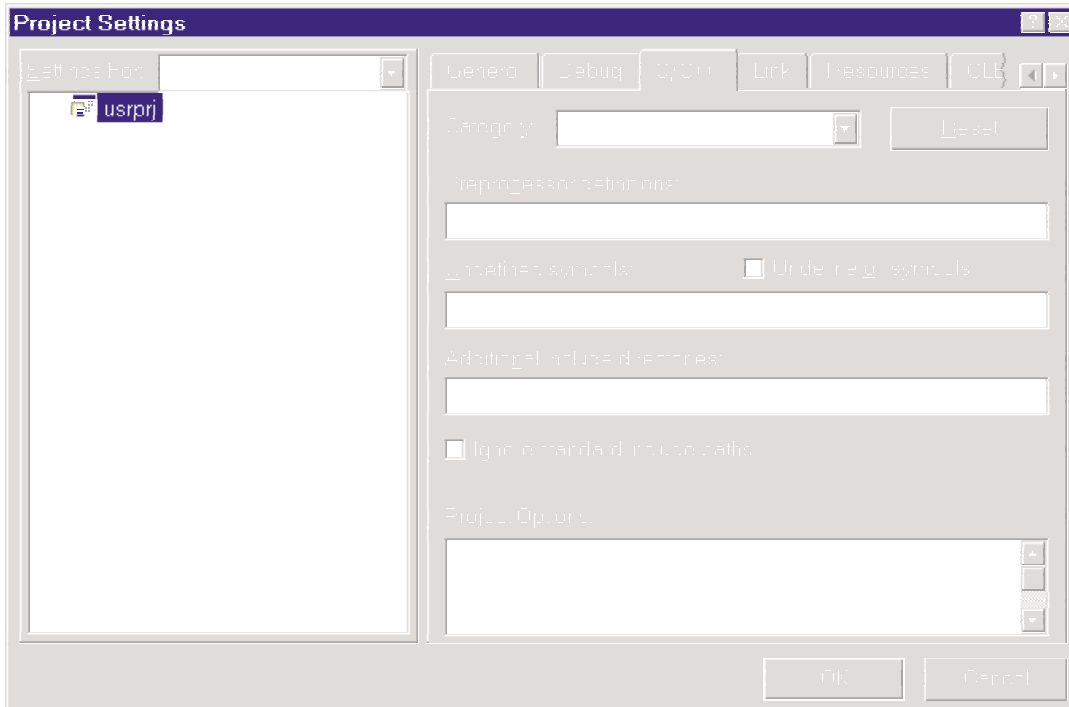


Figure 2-7: C/C++ Tab - Preprocessor Category

Select the Link tab.

Select the category General. In the field Output file name, change the file name according to the Cimatron user directory (**..\var\user\<user_prog_name>.dll**).

In the field Object/Library modules, set the libraries according to the file

..\var\usys\ulist0.dat:

kernel32.lib user32.lib gdi32.lib comdlg32.lib winspool.lib (does not appear in the file ulist0.dat) msvcrt.lib oldnames.lib

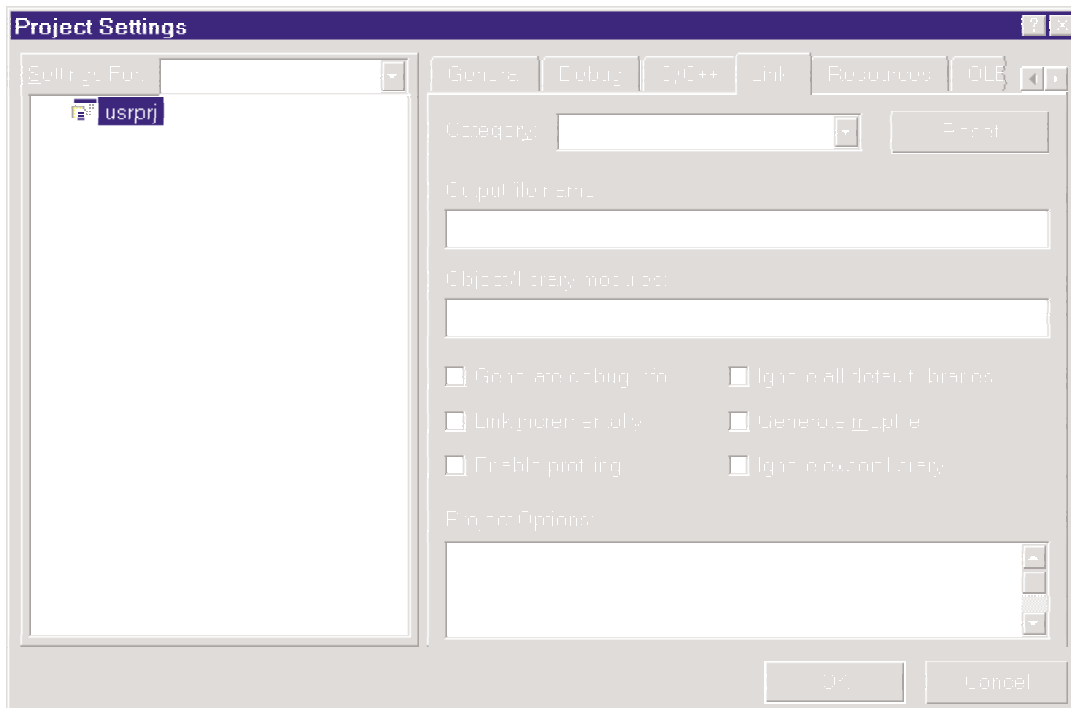


Figure 2-8: Link Tab

Select the category Customize and check the Force file output check box.

Select the category Input and enter libc.lib in the Ignore libraries field.

Select the category Output. Enter 0x1C000000 in the Base address field and enter LibMain@12 in the Entry-point symbol field.

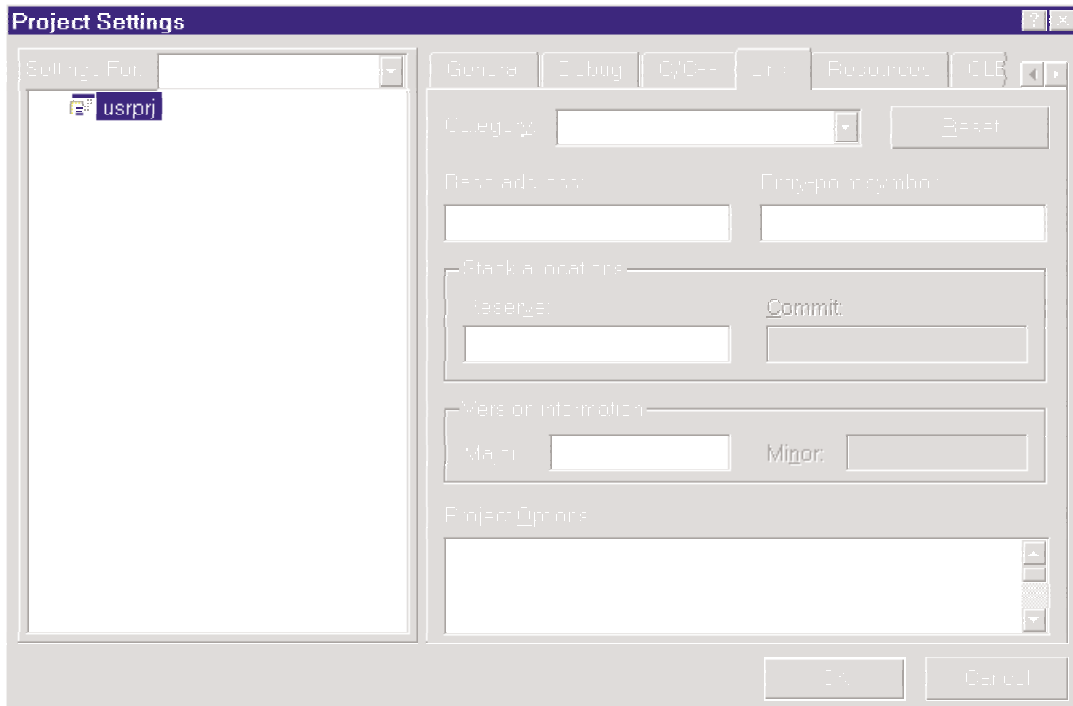


Figure 2-9: Link Tab - Output Category

Press OK to save the settings.

Definition file

The project definition file should as follows:

LIBRARY <user_prog_name>

EXPORTS <user_prog_name>

Adding files

From the Project menu select Add to Project / Files... .

Add the following files to the project:

<Root_cad>\bin\usys.lib

main.obj

.def file

all relevant user files.

The file **main.obj**, can be found in the Cimatron Interanet site, (See page 2-28).

After these settings have been completed, sources may be compiled, linked and debugged using Microsoft Visual C++ tools.

Debugging a user program built with umake

Drag and drop the user DLL file into the Microsoft Visual C++ 5.x. A new project will be opened automatically.

From the Project menu select Settings... (Alt+F7).

Select the Debug tab.

Enter the full path name to the **cimit.exe** in the Executable for debug session field.

Fill in the fields Working directory and Program arguments (if needed).

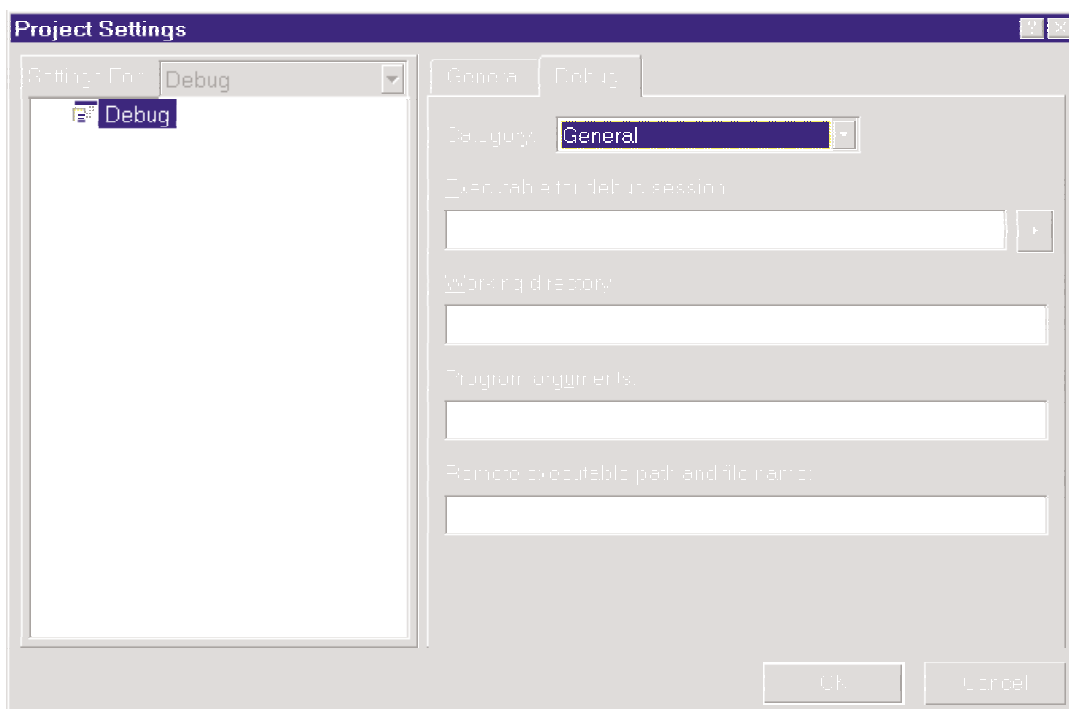


Figure 2-10: Debug Tab

Example: Creating a Rectangle

An example User Program is shown below. Additional examples may be found in the file **cimadek_examples.zip** on Cimatron's Internet site: <http://www.cimatron.com/ftp/manuals/> .

The User Program below creates a rectangle of width A and height B on the work plane. Create a rectangle defined by the following:

the corner where X is smallest and Y is smallest defined by coordinates X, Y and Z

one side parallel to the X axis having length A

one side parallel to the Y axis having length B

lying on a plane parallel to the WORK plane defined by Z

```
#include "for_user.h"
#include "ugeneral.pro"
#include "umodeling.pro"
#define MINLEN 0.001          /* The minimum value of the Height or Width */
#define MAXLEN 1000000.0      /* The maximum value of the Height or Width */
void rectan()
{
    int    IdPnt[5];
    int    IdLin[4];
    int    Status = 0;
    int    Respon = 0;
    int    Option = 1;
    int    NumPnt = 5;
    int    com = 1;
    int    Choice = 1;
    float  Height = 100.0;
    float  Width = 100.0;
    float  MinLen = (float)MINLEN;
    float  MaxLen = (float)MAXLEN;
    float  Xcur = 0.0;
    float  Ycur = 0.0;
    float  Zcur = 0.0;
    float  BasePoint[3];

    /* loop until REJECT or EXIT are pressed. */
    while(Respon >= 0)
    {
        MODINZ(); /* cleaning the Prompt Area. */
        PROMPT("INDICATE CORNER");
        /* get input parameters. */
        MODR2("HEIGHT", &Height, &MinLen, &MaxLen, &com);
        MODR2("WIDTH", &Width, &MinLen, &MaxLen, &com);

        /* get point coordinates. */
        GETPT(BasePoint, &Option, &Respon);

        /* check which mouse button was pressed. */
        if(Respon >= 0)
        {
            DBFLON();

            Xcur = BasePoint[0];
            Ycur = BasePoint[1];
            Zcur = BasePoint[2];

            IdPnt[0] = PTCOWO(&Xcur, &Ycur, &Zcur);

            Xcur += Width;
            IdPnt[1] = PTCOWO(&Xcur, &Ycur, &Zcur);

            Ycur += Height;
            IdPnt[2] = PTCOWO(&Xcur, &Ycur, &Zcur);
```

Example: Creating a Rectangle

```
Xcur -= Width;
IdPnt[3] = PTCOWO(&Xcur,&Ycur,&Zcur);

IdPnt[4] = IdPnt[0];

DBFLOF();

LNPOEN(&NumPnt,IdPnt,IdLin,&Status);
if(Status != 0) USRMES("LNPOEN failure");

/* Remove all temporary entities from the screen. */
DBCLEA();
    }
}
```



Section II

User Package



Section II

User Package

Introduction

The User Package is intended for creating interactive applications and works in conjunction with Cimatron^{it}.

This section includes the following chapters:

General	consisting of general user routines associated with various activities within Cimatron ^{it} .
Modeling (Wireframe & Surfaces)	consisting of user routines associated with the Modeling application of Cimatron ^{it} .
Drafting	consisting of user routines associated with the Drafting application of Cimatron ^{it} .

Commands

The following commands are used:

- umake** - to compile *and* link User Package programs.
- umake -c** or **ucomp** - to compile User Package programs.
- umake -l** or **ulink** - to link User Package programs.

For additional information on commands, see Operating Instructions in Chapter 2.

Development Authorization

The following codes are required:

- user_dev** - to develop *and* run User Package programs.
- user_run** - to run User Package programs.



Chapter 3

General Routines

Introduction

This chapter contains general user routines associated with various activities within Cimatron^{it}. Each routine appears under an appropriate subsection that logically corresponds to the interactive functions of the system.

The subsections contained in this chapter are as follows:

ATTRIB	Create, update and delete attribute records.
DISPLAY	Display entities on the screen.
ENTATT	Retrieve the attributes of an entity.
INTERACTION	Interaction between the operator and the USER procedures.
LEVELS	Level handling.
LINATT	Line attribute routines.
MANAGEMENT	Routines to “manage” your programs.
MATHEMATICS	Mathematical routines and functions. Subdivided into: <ul style="list-style-type: none">- Conversion Routines.- Elementary Arithmetic.- Geometrical Routines.
PICK	Unmask and pick entities.
PLOT	Plot entities.
REPORTS	Handle Reports.
STATUS	Retrieve data on the current part file.
STRING	String activities.

ATTRIB

General Description

With the following routines you can create, update and delete attribute records and perform other functions to attach or detach them to/from geometrical entities.

A call to the **ATTINI** routine must always be issued before a call to any of the other routines listed below.

When working with the interactive Cimatron^{it} system, a **ddinput.att** file must be prepared before using any of the **ATTRIB** routines. (See Appendix D of the Cimatron^{it} **Fundamentals and General Functions Manual**.)

The **ddinput** file defines the record types to be used. In the following example from Appendix D of the Cimatron^{it} **Fundamentals and General Functions Manual**, three record types are defined: **MODEL**, **SUPPLIER** and **NUT**.

RECORD MODEL

```
FIELD (NAME=SERIE,TYPE=I)
FIELD (NAME=COLOR,TYPE=S,MAX=7,DEF=BLUE,DOMAIN=(BLUE, RED,
    YELLOW, BLACK),CONV=U)
FIELD (NAME=PRICE,TYPE=R,DEF=4000.)
FIELD (NAME=WIDTH,TYPE=I,MIN=-123,MAX=123,DEF=0)
FIELD (NAME=MACHINE,TYPE=S,DEF=MS12,CONV=U)
FIELD (NAME=COMPOSE,COMP=(COLOR,MACHINE))
```

RECORD SUPPLIER

```
FIELD (NAME=SERIAL_NR,SERIAL=SSS)
FIELD (NAME=ADDRESS,TYPE=S)
FIELD (NAME=PRICE,TYPE=I,MIN=0,MAX=5000)
```

RECORD NUT

```
FIELD (NAME=MATERIAL,TYPE=S,DEF=BRASS,DOMAIN=(BRASS,
    STEEL,AL))
FIELD (NAME=DIAMETER,TYPE=S,DEF=3/8,DOMAIN=(1/8,3/16,3/8))
FIELD (NAME=MIL-SPEC,TYPE=S)
$END
```

Each record type acquires an ID number when it is created. Records belonging to the same record type are entered sequentially into a table. The record sequence number in the table is referred to as the record number. The result is that every record is fully defined by a pair of numbers - the ID of its record type plus its record number.

Several subroutines require as input the string representing the contents of a record. Fields are separated by the **@** symbol. For example, in the record type **SUPPLIER** in the example, if the following field values are assigned:

NAME = Super_Soft

ADDRESS = Silicon_33

PRICE = 3.28

the string representing this record will be:

Super_Soft@Silicon_33@3.28

This string can be prepared by the programmer using the subroutine PREREC.

Notes:

- The length of each string field must be ≤ 20 characters.
- The length of each field name must be ≤ 8 characters.
- The length of each record type must be ≤ 12 characters.
- The number of fields in each record type must be ≤ 100 .

ATDEID

Attach/detach a record, whose ID is specified, to/from a selected geometric entry.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void ATDEID ( Mode, Idtabl, Recnum, Idgent, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Mode;      /* 1 = Attach
/* 2 = Detach
int    *Idtabl;    /* ID number of the record type table of records.
int    *Recnum;    /* The record number.
int    *Idgent;    /* ID number of the geometric entity.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

ATGECO

Given the ID of a record type table, a field position number, and the contents of the field; retrieve the total number of geometric entities attached to all records in this table, or to records which have a given value attached to a specified field position number.

Syntax :

```

/*- - - - - */
void ATGECO ( Idtabl, Fldnum, Intval, Reaval, Strval, Strlen, Geotot, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idtabl;    /* The ID of the record type table.
int    *Fldnum;    /* Position number of the field in the record. ( If 0,
/* count all geometric entities attached to records in
/* this table. )
int    *Intval;    /* Integer value in the field to be compared, if it is
/* an integer.
float  *Reaval;    /* Real value in the field to be compared, if it is a
/* real.
char   *Strval;    /* String value in the field to be compared, if it is a
/* string.
int    *Strlen;    /* Length of Strval. Maximum 20 characters.
/*
/* Output :
/*
int    *Geotot;    /* Total number of geometric entities which are
/* attached to all records in this table, or to records
/* which have a given value attached to a specified
/* field position number.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

ATSUMU

Given the ID of a record type table and a field position number; find the sum of all the values in this field for all records in this table which are attached to geometric entities.

Syntax :

```

/*- - - - - */
void ATSUMU ( Idtabl, Fldnum, Intsum, Reasum, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idtabl;    /* The ID of the table of records.
int    *Fldnum;    /* Position number of the field in the record.
/*
/* Output :
/*
int    *Intsum;    /* The sum, if the field contains integers.
float  *Reasum;    /* The sum, if the field contains real numbers.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

ATTDET

Attach/detach a record of a record type table whose name and length are specified, to/from a selected geometric entity.

Syntax :

```

/*- - - - - */
void ATTDET ( Mode, Typnam, Typlen, Recnum, Idgent, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Mode;      /* 1 = Attach
/* 2 = Detach
char    *Typnam;    /* The record type name assigned in ddinput.
int    *Typlen;     /* The length of Typnam.
int    *Recnum;     /* The record number.
int    *Idgent;     /* ID number of the geometric entity.
/*
/* Output :
/*
int    *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

ATTINI

Prepare the database for use with ATTRIB subroutines.

Syntax :

```

/*- - - - - */
void ATTINI ( Status )
/*- - - - - */
/*
/* Output :
/*
int    *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

GTABID

Given the name of a record type table, retrieve its ID number.

Syntax :

```

/*- - - - - */
void GTABID ( Typnam, Typlen, Idtabl, Status )
/*- - - - - */
/*
/* Input :
/*
char    *Typnam;    /* The record type table name assigned in ddinput.
int    *Typlen;     /* The length of Typnam in characters.
/*
/* Output :
/*
short   *Idtabl;     /* The ID number of the record type table.
int    *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

GTABNA

Given the ID number of a record type table, retrieve the name of the record type.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GTABNA ( Idtabl, Typnam, Lentyp, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Idtabl;    /* ID number of the record type table.
/*
/* Output :
/*
char    *Typnam;    /* The record type name assigned in ddinput.
int    *Lentyp;    /* Length of Typnam in characters.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GATTR

Given an ID number of a geometric entity, retrieve ID numbers of record type tables attached to the entity and pair them with the appropriate record number of the records in each.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GATTR ( Idgent, From, Max, Tablst, Reclst, Numret, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Idgent;    /* ID number of the geometric entity.
int    *From;      /* List position number of the record from which to
/*                  /* begin the retrieval.
int    *Max;       /* Maximum number of pairs to be retrieved.
/*
/* Output :
/*
short  Tablst[Max]; /* List of ID numbers of record type tables.
short  Reclst[Max]; /* List containing record numbers for each record type
/*                  /* table.
int    *Numret;    /* Number of pairs actually retrieved.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - As records are attached to an entity, they are assigned a sequential number, even if they have different record types. (This is in addition to the record type ID and the sequential record number within a record type table.) This number indicates a position within a list of all records attached to the entity. This is the number required as input for the variable FROM in this routine.

GTFLNA

For a given record type table named TYPNAM, output the names of its fields and indicate the type for each.

Syntax :

```

/*- - - - - */
void GTFLNA ( Typnam, Typlen, Numfld, Fldnam, FlnaleFldtyp, Status )
/*- - - - - */
/*
/* Input :
/*
char *Typnam; /* Record type name assigned in ddinput.
int *Typlen; /* Length of TYPNAM in characters.
/*
/* Output :
/*
int *Numfld; /* Number of fields.
char *Fldnam; /* Array containing the name of the fields.
int *Flnale; /* Array containing the length of each field name.
int *Fldtyp; /* Type of field:
/* 1 = Integer
/* 2 = Float
/* 3 = String
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

GTFLV1

Given the ID of a record type table, a sequential number in the table and a field position number within the record; retrieve the value in the field.

Syntax :

```

/*- - - - - */
void GTFLV1 ( Idtabl, Recnum, Fldnum, Fldtyp, Integr, Rel, Str, Strlen,
/*- - - - - */
/*
/* Input :
/*
int *Idtabl; /* ID of record type table.
int *Recnum; /* Sequential number of record in the record type table.
int *Fldnum; /* Position number of the field within the record.
/*
/* Output :
/*
int *Fldtyp; /* Type of field:
/* 1 = Integer
/* 2 = Float
/* 3 = String
int *Integr; /*
float *Rel; /*
char *Str; /* The contents of the field. One of these variables
/* will be output, depending on the type code of Fldtyp.
int *Strlen; /* The length of the field contents ( number of
/* characters ) if Fldtyp = 3, string.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

GTFLVL

Retrieve the value (contents) of a given field in a given record.

Syntax :

```

/*- - - - - */
void GTFLVL ( Typnam, Typlen, Recnum, Fldnum, Fldtyp, Integr, Rel, Str, Strlen,
              Status )
/*- - - - - */
/*
/* Input :
/*
char *Typnam; /* Record type table name assigned in ddinput.
int *Typlen; /* Length of TYPNAM in characters.
int *Recnum; /* Sequential record number.
int *Fldnum; /* Position number of the field within the record.
/*
/* Output :
/*
int *Fldtyp; /* Type of field:
/*      1 = Integer
/*      2 = Float
/*      3 = String
/*
int *Integr; /*
float *Rel; /*
char *Str; /* The contents of the field. One of these variables
/* will be output, depending on the type code of Fldtyp.
int *Strlen; /* The length of the field contents ( number of
/* characters ) if Fldtyp = 3, string.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

GTGEOM

Given the ID of a record type table and the sequential number of a record in the table; retrieve the ID numbers of the geometric entities attached to it.

Syntax :

```

/*- - - - - */
void GTGEOM ( Idtabl, Recnum, From, Max, Idgent, Numret, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idtabl; /* ID of the record type table.
int *Recnum; /* Record number in the table.
int *From; /* List position number of the record from which to
/* begin the retrieval. ( See note in GTATTR. )
int *Max; /* Maximum number of geometric entities to be
/* retrieved. ( Dimension if Idgent. )
/*
/* Output :
/*
short Idgent[Max]; /* Array containing the IDs of the geometric entities.
int *Numret; /* Number of geometric entities actually retrieved.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

GTRCNU

Given the contents of a record, retrieve its table and the number which indicates its position in the table.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int GTRCNU ( Typnam, Typlen, String, Strlen, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Typnam; /* Record type table name assigned in ddinput.
int *Typlen; /* Length of Typnam in characters.
char *String; /* String containing the contents of the record with a
/* @ as a separator between fields.
int *Strlen; /* Length of the string. ( The number of significant
/* characters plus the separators. )
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* Number which represents the position of the
/* specified record in the table.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GTTYE

Given the ID of a record type table and a field position number, retrieve the type of the field.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void GTTYE ( Idtabl, Fldnum, Fldtyp, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Idtabl; /* The ID of the record type table.
int *Fldnum; /* Position number of the field in the record.
/*
/* Output :
/*
int *Fldtyp; /* Type of field:
/* 1 = Integer
/* 2 = Float
/* 3 = String
int *Status; /* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

IGTABID

Given the name of a record type table, retrieve its ID number.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void IGTABID ( Typnam, Typlen, Idtabl, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Typnam; /* The record type table name assigned in ddinput.
int *Typlen; /* The length of TYPNAM in characters.
/*
/* Output :
/*
int *Idtabl; /* The ID number of the record type table.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

IGTATTR

Given an ID number of a geometric entity, retrieve ID numbers of record type tables attached to the entity and pair them with the appropriate record number of the records in each.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void IGTATTR ( Idgent, From, Max, Tablst, Reclst, Numret, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Idgent; /* ID number of the geometric entity.
int *From; /* List position number of the record from which to
/* begin the retrieval.
int *Max; /* Maximum number of pairs to be retrieved.
/*
/* Output :
/*
short Tablst[Max]; /* List of ID numbers of record type tables.
short Reclst[Max]; /* List containing record numbers for each record type
/* table.
int *Numret; /* Number of pairs actually retrieved.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note: As records are attached to an entity, they are assigned a sequential number, even if they have different record types.
 (This is in addition to the record type ID and the sequential record number within a record type table.)
 This number indicates a position within a list of all records attached to the entity.
 This is the number required as input for the variable FROM in this routine.

IGTGEOM

Given the ID of a record type table and the sequential number of a record in the table; retrieve the ID numbers of the geometric entities attached to it.

Syntax :

```

/*- - - - - */
void IGTGEOM ( Idtabl, Recnum, From, Max, Idgent, Numret, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idtabl;    /* ID of the record type table.
int    *Recnum;    /* Record number in the table.
int    *From;      /* List position number of the record from which to
/* begin the retrieval. ( See note in GTATTR. )
int    *Max;       /* Maximum number of geometric entities to be
/*
/* Input/Output :
/*
int    *Idgent;    /* Array containing the IDs of the geometric entities.
/*
/* Output :
/*
int    *Numret;    /* Number of geometric entities actually retrieved.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

NGCPAT

Attach to an entity all NGD records attached to another entity.

Syntax :

```

/*- - - - - */
void NGCPAT ( Idto, Idfrom, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idto;      /* ID of the entity to which the NGD records are to be
/* attached.
int    *Idfrom;    /* ID of the "master" entity.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

PREREC

Construct a string (OUTSTR), representing the contents of a record by appending field values and inserting @ between them.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void PREREC ( Fldtyp, Integr, Rel, Str, Strlen, Maxlen, Pos, Outstr, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Fldtyp;    /* Type of field:
/*      1 = Integer
/*      2 = Float
/*      3 = String
/*
int    *Integr;    /*
float  *Rel;       /*
char   *Str;       /* Contents of the field. One of these variables will
/* be used depending on the type code of Fldtyp.
int    *Strlen;    /* Length of the field contents (if Fldtyp = 3, String).
int    *Maxlen;    /* Maximum length of OUTSTR in characters, including
/* separators.
/*
/* Input/Output :
/*
int    *Pos;       /* First available character position to which field
/* contents can be appended.
/* The next available character position to which field
/* contents can be appended.
char   *Outstr;    /* Contents of the record to be constructed before the
/* current field is appended.
/* The contents of the constructed record after the
/* current field is appended.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

RECCRE

Create a new record in a specified record type table.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void RECCRE ( Typnam, Typlen, Strnew, Lennew, Recnum, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char   *Typnam;    /* Record type table name assigned in dinput.
int    *Typlen;    /* Length of TYPNAM in characters.
char   *Strnew;    /* The string containing the contents of the record to
/* be created with @ as a separator between fields.
int    *Lennew;    /* The length of the new record, Strnew in characters
/* including separators.
/*
/* Output :
/*
int    *Recnum;    /* The sequential record number.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

RECDEL

Delete a record from a specified record type table.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void RECDEL ( Typnam, Typlen, Strold, Lenold, Recnum )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Typnam; /* The record type table name assigned in dinput.
int *Typlen; /* Length of Typnam in characters including separators.
char *Strold; /* The string containing the significant contents of
/* the record to be deleted with @ as a separator
/* between fields.
int *Lenold; /* The length of the old record, Strold.
/*
/* Output :
/*
int *Recnum; /* The position number in the table of records of the
/* deleted record.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

RECUPD

Replace an existing record in a specified record type table with a new one.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void RECUPD ( Typnam, Typlen, Strold, Lenold, Strnew, Lennew, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Typnam; /* The record type table name assigned in dinput.
int *Typlen; /* The length of Typnam.
char *Strold; /* The string containing the significant contents of
/* the record to be replaced with @ as a separator
/* between fields.
int *Lenold; /* The length of the old record, STROLD, in characters
/* including separators.
char *Strnew; /* The string containing the significant contents of
/* the record to be created with @ as a separator
/* between fields.
int *Lennew; /* The length of the new record, Strnew, in characters
/* including separators.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

RETYIN

Given the ID of a record type table, retrieve the total number of records in the table.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - */
void RETYIN ( Idtabl, Rectot, Status )
/*- - - - - - - - - - - - - - - - - - - - */
                                /*          */
                                /* Input :   */
                                /*          */
int    *Idtabl;                /* The ID of the table of records. */
                                /*          */
                                /* Output : */
                                /*          */
int    *Rectot;                /* Total number of records in the table. */
int    *Status;                /* See General Concepts-Status on page 1-2. */
/*- - - - - - - - - - - - - - - - - - - - */

```



DISPLAY

General Description

Display entities on the screen.

CDK_RUBBER_CIRCLE Draws a rubber circle.
Syntax :

```

/*- - - - - */
void CDK_RUBBER_CIRCLE ( center, rad, mode )
/*- - - - - */
/*
/* Input :
/*
double center[3]; /* The circle's center (WORK).
double rad;       /* The circle's radius.
int mode;         /* Mode to create :
/*              = 0 : Erase, = 1 : Create, = 2 : Compliment.
/*- - - - - */

```

CDK_RUBBER_LINE Draws a rubber line.
Syntax :

```

/*- - - - - */
void CDK_RUBBER_LINE ( start, end, mode )
/*- - - - - */
/*
/* Input :
/*
double start[3]; /* The Line's Start Point (WORK).
double end[3];   /* The Line's End Point (WORK).
int mode;        /* Mode to create :
/*              = 0 : Erase, = 1 : Create, = 2 : Compliment.
/*- - - - - */

```

CDK_WINDOW_ACTIVE Activate an existing window.
Syntax :

```

/*- - - - - */
int CDK_WINDOW_ACTIVE ( void )
/*- - - - - */
/*
/* Input :
/*
int vpid; /* View port id number.
/* Returns :
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_WINDOW_DELETE Delete an existing window.
Syntax :

```

/*- - - - - */
int CDK_WINDOW_DELETE ( vpid )
/*- - - - - */
/*
/* Input :
/*
int    vpid;    /* View port id number.
/* Returns :
/* See General Concepts-Status on page 1-2.
/* = -3 : attempt to delete active window.
/* = -4 : attempt to delete last window.
/*- - - - - */

```

CDK_WINDOW_MULTY Recall existing multi-window layout.
Syntax :

```

/*- - - - - */
int CDK_WINDOW_MULTY ( )
/*- - - - - */
/* Returns :
/* = -1 : multi-window layout is already displayed.
/* = -2 : multi-window layout does not exist.
/*- - - - - */

```

CDK_WINDOW_NEW_DIVISION

Define new windows by a division point.

Syntax :

```

/*- - - - - */
int CDK_WINDOW_NEW_DIVISION ( XDiv, YDiv, vps )
/*- - - - - */
/*
/* Input :
/*
int    XDiv;    /* X Division point.
int    YDiv;    /* Y Division point.
/*
/* Output :
/*
int    vps[];   /* The new created view port ids 2-4.
/* Returns :
/* See General Concepts-Status on page 1-2.
/* = -3 : Too many windows.
/* = -4 : Window too small.
/*- - - - - */

```

CDK_WINDOW_NEW_NUMERIC

Define new windows by numeric divide.

Syntax :

```
/*- - - - - */
int CDK_WINDOW_NEW_NUMERIC ( HorDiv, VerDiv )
/*- - - - - */
/*
/* Input :
/*
int    HorDiv;    /* Number of horizontal divisions.
int    VerDiv;    /* Number of vertical divisions.
/* ( HorDiv * VerDiv <= 10 )
/* Returns :
/* See General Concepts-Status on page 1-2.
/* = -3 : Too many windows.
/* = -4 : Window too small.
/*- - - - - */
```

CDK_WINDOW_SINGLE

Change to a single window.

Syntax :

```
/*- - - - - */
int CDK_WINDOW_SINGLE ( vpid )
/*- - - - - */
/*
/* Input :
/*
int    vpid;    /* view port id number.
/* Returns :
/* See General Concepts-Status on page 1-2.
/* = -2 : Single window is already displayed.
/*- - - - - */
```

CDK_WINDOW_SUB_DIVIDE

Subdivide a given window by a given division point.

Syntax :

```
/*- - - - - */
int CDK_WINDOW_SUB_DIVIDE ( vpid, XDiv, YDiv, vps )
/*- - - - - */
/*
/* Input :
/*
int    vpid;    /* View port id number.
int    XDiv;    /* X Division point.
int    YDiv;    /* Y Division point.
/*
/* Output :
/*
int    vps[];    /* The new created view port ids 2-4.
/* Returns :
/* See General Concepts-Status on page 1-2.
/* = -3 : Too many windows.
/* = -4 : Window too small.
/*- - - - - */
```

ENTATT

General Description

The routines in this section are used for retrieving the attributes of an entity.

EGATTR

Get the line attributes of an entity.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void EGATTR ( Id, Font, Color, Pen, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Id;
/* Entity identifier.
/*
/* Output :
/*
int *Font;
/* Entity LINEFONT ( Sequential number ).
int *Color;
/* Entity COLOR ( Sequential number ).
int *Pen;
/* Entity PENNUM ( Sequential number ).
int *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

EGBLNK

Check if a given entity is blanked.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void EGBLNK ( Id, Blanked, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Id;
/* Entity identifier.
/*
/* Output :
/*
int *Blanked;
/* 0 = Not blanked.
/* 1 = Blanked.
int *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

EGLEVL

Get the level of an entity.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void EGLEVL ( Id, Level, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Entity identifier.
/*
/* Output :
/*
/* Entity level.
/*
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

EGNGD

Check if NGD is attached to this entity.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void EGNGD ( Id, Ngd, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Entity identifier.
/*
/* Output :
/*
/* 0 = Not attached.
/*
/* 1 = Attached.
/*
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

EGTRAN

Get the matrix transformation of an entity.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void EGTRAN ( Id, Mat, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Entity identifier.
/*
/* Output :
/*
/* Entity matrix.
/*
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

EVISM

Get/set visibility mask from/to the entity ID.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void EVISM( Id, Mode, Vis )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Entity ID.
/* = 1 : Get.
/* = 2 : Set.
/*
/* Input / Output :
/*
/* The visibility mask ( mask convention ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int *Id;
int *Mode;
shrt Vis[];

```

OBATTR

Obtain entity attribute. The only valid attributes are:

'SL' - Slave
 'TY' - Type
 'CL' - Class
 'LE' - Level
 'DE' - Delete status
 'BL' - Blank status
 'VN' - Parent view number
 'PR' - Application number
 'LF' - Line font
 'CO' - Color
 'PE' - Pen
 'RP' - Pointer to the transformation matrix
 'MV' - Model-view association pointer
 'NG' - NGD status
 'OW' - Owner
 'ME' - Member

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBATTR ( Id, Name, Value )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Entity identifier.
/* The attribute name.
/*
/* Output :
/*
/* > 0 Value of entity attributes.
/* < 0 See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int *Id;
char *Name;
int *Value;

```

UEVISM

Get visibility mask of entity ID

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UEVISM ( Id, Vism )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*                                     */
int      *Id;                   /* Entity identifier.
                                /*                                     */
                                /* Output :
                                /*                                     */
short    Vism[4];               /* The visibility mask.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



INTERACTION

General Description

The following routines handle interaction between the operator and the **USER** procedures. More routines for handling interaction are described in the **PICK** group in this chapter.

In the interactive system the top line on the screen is divided into the prompt area and the interaction area. The operator controls the prompt area (columns 1-20 of the top line) by using the **PROMPT** subroutine. The interaction area, (columns 21 to the end of the line) may be manipulated by using the other routines in this section.

The subroutine **MODINZ** must be called before attempting to use **MODI**, **MODFX**, **MODM** and **MODR**. This clears the interaction area of any previous contents and initializes it. To just clear the interaction area, use the subroutine **DBLNK**. Use **DTEXT** to display text in the interaction area.

To read values from the keyboard, use **MODI** (for integers), **MODFX** (for float numbers independent of the **USER** units) or **MODR** (for float numbers in the current **USER** units). **MODM** may be used to select one item, from a list of items provided, via the input of the routine.

To set default values for variables which will be displayed in one of the **MOD** routines (**MODFX**, **MODR**), assign the default value after the **MODINZ** call and directly before the **MOD** routine call. Only one value is stored for each variable. If the value of the variable is changed interactively within a routine, the new value is stored as the value of the variable. Defaults set before **MODIFY** was called will not be used.

When a routine which requires a response is called, the program must provide the operator with the ability to input that response. This is done with **MODIFY**, when modal values are to be input. **UPICK**, to pick a single entity or, **GETPT**, to indicate a point from the **PICK** group of routines, also give access to modal fields by calling the **MODIFY** routine, automatically.

Routines **DMATT**, **DMDEF**, **DMSHOW**, **DMINZ**, **DMINZ5**, **DMINP** and **DMINPT** may be used for work with dynamic menus (such as appear when defining the Selection Mask or the active level).

The subroutine **DMINZ** or **DMINZ5** must be called before attempting to use any other routine of this group. These routines initialize the dynamic menu buffer, **DMINZ** for one-line menus and **DMINZ5** for multi-line menus. To define the next field use the subroutine **DMDEF**. If the menu has more than one layout use the subroutine **DMSHOW** to define which 'page' to show. To highlight the appropriate field use the subroutine **DMATT**. Subroutines **DMINP** and **DMINPT** are intended for getting the users' response.

The subroutine **RTEXT** reads a string of characters from the keyboard. The maximum number of characters is defined by the user as input to the routine.

USRERR sounds a bell and displays the specified text message. This routine is useful for displaying error or warning messages.

The routine **REPLY** accepts a **YES/NO** answer to a question to be displayed in the prompt field area, allowing the program to continue accordingly.

It is important to remember that the interaction area begins at column 21. Thus, whenever specifying a column number in the interaction area in a routine, care must be taken to add 20 to the absolute column number required. The columns available for the interaction are 21 to the end of the line. The maximum length will depend on the operating system.

CDK_FILE_BROWSE

Interactive file browser.

Syntax:

```

/*- - - - - */
int CDK_FILE_BROWSE ( FileBrowser, initDir, fExtention, fName )
/*- - - - - */
int      FileBrowser; /* I : File Browser Option :
/*                      = 0 - Default File Browser.
/*                      = 1 - Windows File Browser ( Win32 platforms ).
char      initDir[]; /* future parameter.
char      fExtention[]; /* I : file extension (use as a file type filter).
char      fName[]; /* O : full path name of selected file.
/* R : Status
/* > 0 - O.K. : Number of files in the given
/*                      directory.
/* = 0 - No files found.
/* < 0 - User response : Exit/Reject.
/*- - - - - */

```

CDK_HOT_VIEW

Change the view picture (front, side, iso, top) according to the hot-key pressed.

Syntax:

```

/*- - - - - */
int cdk_hot_view ( key )
/*- - - - - */
/*
/* Input :
/*
int      key; /* Hot-Key number: front=1, side=2, iso=3, top=4
/*
/* Return value :
/* 0 = OK
/* 1 = ERROR
/*- - - - - */

```

CDK_LEVEL_ACTIVATE

Activate a predefined level.

Syntax:

```

/*- - - - - */
int CDK_LEVEL_ACTIVATE(level)
/*- - - - - */
/* Purpose: Activate a predefined level.
/*
/* Input :
/*
int      level; /* Level Number.
/* Return Status :
/*- - - - - */

```

CDK_LEVELS_DSP_MASK Obtain displayed level mask of a view port.
Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_LEVELS_DSP_MASK ( vpid ,dsp_mask )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int    vpid;           /* I   : View port ID.                */
int    dsp_mask[32];   /* O   : Levels entities mask.         */
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* R   : Status.                */
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_LEVELS_ENT_MASK Obtain level mask of all entities.
Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void CDK_LEVELS_ENT_MASK ( ent_mask )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int    ent_mask[32];   /* O   : Level entities mask.         */
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* R   : Status.                */
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_MENU_POPUP_DEF Invoke a dynamic popup menu with options of line and column definition.
Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_MENU_POPUP_DEF ( Text, Ds, tDim, Choice )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*                                     */
/* Input :                             */
/*                                     */
char    *Text;           /* String containing the contents of the record with a */
/*                                     */
/* @ as a separator between fields.    */
int     Ds[2];           /* Device coordinates - X, Y of top-left corner.      */
int     tDim[2];         /* Text dimensions - columns/rows.                    */
/*                                     */
/* Input :                             */
/*                                     */
int     *Choice;         /* > 0 : Sequential number of the selected item.      */
/*                                     */
/* < 0 : ERROR;            */
/* Returns : user response */
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK MODALS SET COLORS

Syntax :

```

*-----*/
int CDK_MODAL_Sets_COLORS( Opt, NumMods, Colors )
/*-----*/
/*
/* Input:
/*
int      Opt;
/* = 1 : Use the current colors setting.
/* = -1 : The colors of the modals can be set by
/*        pressing ctrl + b.
int      NumMods;
/* Number of modals in the table.
int      Colors[];
/* A table of colors options, the size of the table is
/* the number of the modals to be set.
/* When using Opt == 1 the options (1 - 5) are used as
/* defined in Cimatron using ctrl + b.
/* When using Opt == -1 then :
/*   = 1 Set to default color.
/*   = 2 Set to default complement color.
/*   = 3 Set to not as default color.
/*   = 4 Set to not as default complement color.
/* Returns :
/* See General Concepts-Status on page 1-2.
/*-----*/

```

CDK_MODAL_SET_SIZE	Set number of modals and size for the next row of modals.
---------------------------	---

Syntax :

```

/*- - - - - */
int CDK_MODALSET_SIZE( NumMods, ModLen )
/*- - - - - */
/*
/* Input:
/*
int NumMods; /* Number of modals in the next row.
int ModLen[]; /* Array of column lengths ( Array size is NumMods ).
/* Returns :
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Notes :

- Until next call to MODCLS or to MODINT, call this function only if you need a different setting for the modals in the next row.
- The modal size cannot exceed 30 characters.

DBLNK Display a number (`NUMBER`) of blanks in the interaction area.

Syntax :

```

/*      - - - - - */
void DBLNK ( Column, Number )
/*      - - - - - */
/*          /*
/*          /* Input :
/*          /*
int      *Column;    /* Column number at which display will begin. ( COLUMN
/*          /* may not be less than 21.)
int      *Number;    /* Number of blanks to display.
/*      - - - - - */

```

Note : - COLUMN + NUMBER may not exceed the maximum line length.

DISPOF Stop display of only those entities created after this subroutine is activated.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DISPOF ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

Note : - Entities created after DISPOF has been activated are not displayed. The entities will only be displayed if Redraw (from the Immediate Access Pop-Up Functions) is selected.

DISPON Enable display of entities created after this routine is activated. Disable the previously activated DISPOF subroutine.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DISPON ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

DMATT Change selection mark on button of dynamic menu.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DMATT ( Item )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int *Item; /* Number of menu items.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

DMDEF Define a dynamic menu field and display it immediately.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DMDEF ( Name )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
char *Name; /* Field name ( English ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

DMDEFN

Define a dynamic menu field and display it immediately. When working with the option '-lang other', the field name is translated.

Syntax :

```
/*- - - - - */
void DMDEFN ( Name )
/*- - - - - */
/*
/* Input :
/*
char *Name; /* Field name ( English ).
/*- - - - - */
```

DMINP

Select dynamic menu item, Immediate-Access functions allowed.

Syntax :

```
/*- - - - - */
void DMINP ( Res )
/*- - - - - */
/*
/* Output :
/*
int *Res; /* User response:
/* = 0: Selected item number.
/* = -1: EXIT
/* = -2: REJECT
/*- - - - - */
```

DMINPT

Select dynamic menu item.

Syntax :

```
/*- - - - - */
void DMINPT ( Res, Name, Immf )
/*- - - - - */
/*
/* Output :
/*
int *Res; /* The user response:
/* > 0: Selected item number.
/* = -1: EXIT
/* = -2: REJECT
char *Name; /* Name of the selected item.
/*
/* Input :
/*
int *Immf; /* = TRUE:
/* Access to immediate functions.
/* = FALSE:
/* No access to immediate functions.
/*- - - - - */
```

DMINZ Initialize the dynamic menu buffer for one-line menus.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DMINZ ( Msize )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int *Msize; /* Field size ( number of characters ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DMINZ5 Initialize the dynamic menu buffer for multi-line menus.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DMINZ5 ( Msize )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int *Msize; /* Field size ( number of characters ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DMSHOW Show a dynamic menu and set displaying a one or more line group containing a given item number.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DMSHOW ( Item )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int *Item; /* The item number in the dynamic menu buffer.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DPIXEL To display a pixel.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DPIXEL ( Pixel, Dmode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int Pixel[2]; /* The display coordinates of the pixel.
int *Dmode; /* Display mode:
/* 0 = Erase,
/* 1 = Display,
/* 2 = Turn to complement.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DTEXT

Display text in the interaction area.

Syntax :

```

/*- - - - - */
void DTEXT ( Text, Column )
/*- - - - - */
/*
/* Input :
/*
/* Text to be displayed.
/* Column number at which text will begin. ( Column may
/* not be less than 21. )
/*
/*- - - - - */
char *Text;
int *Column;

```

GSCLR

GET/SET no clear frame status ON or OFF.

Syntax :

```

/*- - - - - */
void GSCLR ( Mode, Clear )
/*- - - - - */
/*
/* Input :
/*
/* GET/SET Mode:
/* 1 = Get,
/* 2 = Set.
/*
/* Input/Output:
/*
/* Clear status:
/* 0 = Clear,
/* 1 = No Clear.
/*
/*- - - - - */
int *Mode;
int *Clear;

```

Note : - Use this routine with MODINT and MODISP for changing MODALS without refreshing the MODALS TABLE (MODE = 2, CLEAR = 1).

GUPDAT

Read user input during segment execution.

Syntax :

```

/*- - - - - */
void GUPDAT( UsInp )
/*- - - - - */
/*
/* Output:
/*
/* UsInp > 0 = a key that was pressed.
/*
/*- - - - - */
int *UsInp;

```

MENU

Display a menu in the interaction area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MENU ( Items, Respon )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char    *Items;                /* String of characters with a "@" character separating
                                /* each item of the menu. Each item has a maximum
                                /* length of 30 characters.
                                /*
                                /* Output :
                                /*
int     *Respon;              /* User's response. See General Concepts in Chapter 2.
                                /* If one of the items was picked, RESPON will be equal
                                /* to the sequential number of the item selected from
                                /* the menu ( i.e., 1 for first, 2 for second etc. ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODCOM

Flip mode of complement of modal and redisplay the modal.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODCOM ( Key )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int     *Key;                 /* The modal number.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODDIM

Get dimensions of modal table with maximum item size.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODDIM ( Numcol, Numrow, Itmlen )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Output :
                                /*
int     *Numcol;              /* Maximum number of columns.
int     *Numrow;              /* Maximum number of rows.
int     *Itmlen;             /* Maximum item length.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODFX

Define a new numeric (float) modal parameter in the interaction area to be displayed/changed, independent of the unit of measure of the part.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODFX ( Parnam, Reavar, Minval, Maxval )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Parnam; /* Parameter name with a maximum length of 20
/* characters.
/*
/* Input/Output :
/*
float *Reavar; /* Float variable to be displayed.
/* Changed float variable.
/*
/* Input :
/*
float *Minval; /* Minimum value that may be assigned to REAVAR.
float *Maxval; /* Maximum value that may be assigned to REAVAR.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - If PARNAM has a length of over 20 characters, it is truncated and an error message is displayed.

MODFX2

Define a new numeric modal parameter to be displayed/changed in interaction area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODFX2 ( Parnam, Reavar, Minval, Maxval, Commode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Parnam; /* Parameter name with maximum length of 20 characters.
/*
/* Input/Output :
/*
float *Reavar; /* Float variable value to be displayed.
/* Changed float variable.
/*
/* Input :
/*
float *Minval; /* Minimum value that may be assigned to REAVAR.
float *Maxval; /* Maximum value that may be assigned to REAVAR.
int *Commode; /* Complement mode:
/* 0 = no complement,
/* 1 = complement.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODHIX

Return length value of the MODALS AREA.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int MODHIX ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Returns :
/* The length of the MODALS AREA.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODI

Define a new numeric (integer) modal parameter to be displayed/changed in the interaction area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODI ( Parnam, Intvar, Minval, Maxval )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Parameter name with a maximum length of 20
/* characters.
/*
/* Input/Output :
/*
/* Integer variable value to be displayed.
/* Changed integer variable.
/*
/* Input :
/*
/* Minimum value that may be assigned to INTVAR.
/*
/* Maximum value that may be assigned to INTVAR.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - If PARNAM has a length of over 20 characters, it is truncated and an error message is displayed.

MODI2

Define a new numeric modal parameter to be displayed/changed in interaction area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODI2 ( Parnam, Intvar, Minval, Maxval, Commode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Parameter name with maximum length of 20 characters.
/*
/* Input/Output :
/*
/* Integer variable value to be displayed.
/* Changed integer variable.
/*
/* Input :
/*
/* Minimum value that may be assigned to INTVAR.
/*
/* Maximum value that may be assigned to INTVAR.
/*
/* Complement mode:
/* 0 = no complement
/* 1 = complement
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODIF1 Unlimited modal modification interaction with immediate return.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODIF1 ( Respon )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int    *Respon;    /* See Chapter 1, General Concepts, Mouse Response.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODIFY Wait for the operator to change modal values in the interaction area, terminated by an extra <CR>.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODIFY ( Respon )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int    *Respon;    /* User's response. See Chapter 1, General Concepts.
/* The second <CR> will cause Respon to be equal to
/* zero.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODINT Initialize modal parameters interaction buffer.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODINT ( Inloc, Itleng, Ncols, Nrows, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    Inloc[2];    /* Location of UpperLeft corner ( screen coordinates ):
/* ( 0, 0 ) UpperLeft corner right to prompt.
/* ( -1, 0 ) UpperLeft corner bellow prompt
/*
int    *Itleng;    /* Fixed field size.
int    *Ncols;    /* The number of columns.
int    *Nrows;    /* The number of rows.
/*
/* Output :
/*
/*
int    *Status;    /* Error flag:
/* 0 = O.K.
/* -1 = Error.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODINZ

Initialize (clear) the interaction area by deleting any previous contents.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - */
void MODINZ ( void )
/*- - - - - - - - - - - - - - - - - - - - */
```

MODISP

Display or redisplay modal parameters from modals buffer.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - */
void MODISP ( void )
/*- - - - - - - - - - - - - - - - - - - - */
```

MODM

Set the next modal parameter to be displayed in the interaction area, to be a list of items. The items in the list will be pulled down every time the appropriate modal is picked.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - */
void MODM ( List, Choice )
/*- - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *List; /* A string of characters making up a list, each item
/* of the list being separated by a "@" character. List
/* has a maximum length of 30 characters.
/*
/* Output :
/*
int *Choice; /* Sequential number of the default/selected item.
/*- - - - - - - - - - - - - - - - - - - - */
```

MODM2

Set the next modal parameter to be displayed in the interaction area, to be a list of items. The items in the list will be pulled down every time the appropriate modal is picked.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODM2 ( List, Choice, Commode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *List;
/* A string of characters making up a list, each item
/* of the list being separated by a "@" character. The
/* item of the List has a maximum length of 30
/* characters.
/*
/* Input/Output :
/*
int *Choice;
/* Sequential number of the default/selected item.
/*
/* Input :
/*
int *Commode;
/* Complement mode:
/* 0 = No complement
/* 1 = Complement.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODMP2

Set the next modal parameter to be displayed in the interaction area, to be a list of items. The items in the list will be pulled down every time the appropriate modal is picked. When there are two items they are both opened.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODMP2 ( List, Choice, Commode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *List;
/* A string of characters making up a list, each item
/* of the list being separated by a "@" character. The
/* item of the List has a maximum length of 30
/* characters.
/*
/* Input/Output :
/*
int *Choice;
/* Sequential number of the default/selected item.
/*
/* Input :
/*
int *Commode;
/* Complement mode:
/* 0 = No complement
/* 1 = Complement.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODMPL

Set the next modal parameter to be displayed in the interaction area, to be a list of items. The items in the list will be pulled down every time the appropriate modal is picked. When there are two items they are both opened.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODMPL ( List, Choice )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *List;
/* A string of characters making up a list, each item
/* of the list being separated by a "@" character. The
/* item of the List has a maximum length of 30
/* characters.
/*
/* Input/Output :
/*
int *Choice;
/* Sequential number of the default/selected item.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MODR

Define a new numeric (float) modal parameter (in the unit of measure of the part) to be displayed/changed in the interaction area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODR ( Parnam, Reavar, Minval, Maxval )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Parnam;
/* Parameter name with a maximum length of 20
/* characters.
/*
/* Input/Output :
/*
float *Reavar;
/* Float variable to be displayed.
/* Changed float variable.
/*
/* Input :
/*
float *Minval;
/* Minimum value that may be assigned to Reavar.
float *Maxval;
/* Maximum value that may be assigned to Reavar.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - This subroutine differs from MODFX in that MODR takes the unit of measure of the part into account by converting REAVAR from mm. to part units at input, and from part units to mm. at output.
- If PARNAM has a length of over 20 characters, it is truncated, and an error message is displayed.

MODR2

Define a new numeric modal parameter (in the unit of measure of the part) to be displayed/changed in interaction area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MODR2 ( Parnam, Reavar, Minval, Maxval, Commode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Parnam; /* Parameter name with maximum length of 20 characters.
/*
/* Input/Output :
/*
float *Reavar; /* Float variable value to be displayed.
/* Changed float variable.
/*
/* Input :
/*
float *Minval; /* Minimum value that may be assigned to Reavar.
float *Maxval; /* Maximum value that may be assigned to Reavar.
int *Commode; /* Complement mode:
/* 0 = No complement
/* 1 = Complement.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OUTCB

To display a character string on the screen.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OUTCB ( Col, Row, Buf, Nc )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Col; /* Column number ( up to 100 ).
int *Row; /* Row number.
char *Buf; /* The string.
int *Nc; /* Length of the string.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes :

- Right-to-left direction may be set by specifying negative column numbers.
- The second character set may be set by specifying negative row numbers.
- A negative column number is interpreted as follows: -1 = 100, -2 = 99, etc.
- The position specified by COL and ROW marks the spot where the upper left corner of the text will be placed, when a left-to-right direction is specified, or the upper right corner when the direction is right-to-left.

OUTCGB

To display a character string on the screen.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void OUTCGB ( Buf )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
    /*
    /* Input :
    /*
    char *Buf;
    /* The string.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - Use after SETGC.
 - The maximum number of characters is 254.

PROMPT

Display text in the prompt area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void PROMPT ( Text )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
    /*
    /* Input :
    /*
    char *Text;
    /* Text to be displayed as prompt ( up to a maximum of
    /* 20 characters ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

REPLY

Accept YES/NO to the question PROMPT in the prompt area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int REPLY ( Prompt )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
    /*
    /* Input :
    /*
    char *Prompt;
    /* Text to be used as the question ( up to a maximum of
    /* 20 characters ).
    /*
    /* Returns :
    /* .TRUE.
    /* Yes.
    /* .FALSE.
    /* No.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

RTEXT

Read characters from the keyboard (up to a maximum of **MAXMUM**), echo them in the interaction area and store them in the character string **NAME**.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void RTEXT ( Column, Maxmum, Name, Count )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Column;    /* Column number in the interaction area at which to
/* begin echoing the characters. ( Column may not be
/* less than 21. )
int    *Maxmum;    /* Maximum number of characters to accept.
/*
/* Output :
/*
char    *Name;     /* Character string containing the characters input
/* from the keyboard.
int    *Count;     /* Number of characters in NAME. If <EXIT> or <REJECT>
/* is pressed at any stage during the input, Count
/* receives the value of the mouse response. (See Mouse
/* Response in Chapter 2, General Concepts.)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

SETGC

Set the start position on the screen for an output string.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SETGC ( X, Y )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *X;         /*
int    *Y;         /* The display coordinates.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

Note : - Use before OUTCGB.

SETIR

Set an immediate return for PICK functions.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SETIR ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

Note : - Use before PICK functions for immediate return.

SGDCLR

Set the active color for the pixel to be displayed.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SGDCLR ( Numclr )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Numclr;    /* Sequential number of the new color.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TOPOPT

Create a pulldown menu next to the prompt area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void TOPOPT ( Text, Respon )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
char    *Text;    /* String of menu items separated by @.
/*
/* Output :
/*
/*
int    *Respon;    /* User's response. See Chapter 1, General Concepts. If
/* one of the items was picked, Respon will be equal to
/* the sequential number of the item selected from the
/* menu ( i.e., 1 for first, 2 for second etc. ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UHLIGHT

Highlight instance (display in attention mode).

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void uhlight ( id, point, flag, mode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *id;    /* ID of the instance.
float  point[3]; /* Point coordinates (not applicable - dummy)
int    *flag;    /* Display mode (mark / attention / both)
int    *mode;    /* Operation mode (display / erase attention)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UMDCLR

Get the number of the default/selected color from the pull down menu.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void UMDCLR ( Choice )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int    *Choice;    /* Sequential number of the default/selected color.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UMODS Define a new modal parameter string to be displayed/changed in the interaction area.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UMODS ( Parnam, Strval, Strlen, Maxlen )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Parnam; /* Parameter name.
/*
/* Input/Output :
/*
char *Strval; /* String variable to be displayed with a length Strlen.
/* Changed string variable.
int *Strlen; /* The length of string Strval.
/*
/* Input :
/*
int *Maxlen; /* The maximum length of the string.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes :

- The length of PARNAM must be < 20 characters.
- The length of STRVAL must be < 20 characters.
- The combined lengths of PARNAM + STRVAL must be < 29 characters.
- If PARNAM > 20 characters, it is truncated.
- If STRVAL > 20 characters, it is truncated.
- If PARNAM + STRVAL > 29 characters, STRVAL is truncated.

USRERR Sound a bell and display a text message.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void USRERR ( Text )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Text; /* Text to be displayed. Text has a maximum length of
/* 120 characters.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note :

- This routine is useful for producing error or warning messages.

USRMES

Display a message on the screen without sounding the bell.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void USRMES ( Text )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char  *Text;                    /* Text to be displayed. Text has a maximum length of
                                /* 120 characters.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - This routine is useful for producing error or warning messages.

USRMSGDisplay a message on the screen *and* the window pad without sounding the bell.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void USRMSG ( Text )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char  *Text;                    /* Text to be displayed ( maximum length of 120
                                /* characters ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - This routine is useful for producing error or warning messages.



LEVELS

General Description

Routines in this section are for handling levels. A level name may not be more than 6 characters.

CDK_LEVELS_LIST

Get the list of names and number of levels.

Syntax :

```
/*- - - - - */
int CDK_LEVELS_LIST ( list )
/*- - - - - */
/*
/* Output :
/*
char *list; /* List of level names ( six characters per name ).
/* Returns :
/* Number of levels.
/*- - - - - */
```

Note: - CDK_LEVELS_LIST is new function that replaces the old function ULISLE.

CDK_LEV_GSNAME

Get/Set the name of a level by its number in the levels table.

Syntax :

```
/*- - - - - */
void CDK_LEV_GSNAME( Mode, LevNumber, LevName, Status )
/*- - - - - */
/*
/* Input:
/*
int *Mode; /* 1 = Get, 2 = Set.
int *LevNumber; /* The level's number.
/*
/* Input/Output:
/*
char *LevName; /* The level's name.
/*
/* Output:
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

CLRLVB Clear bit corresponding to LEVNUM in LVMASK.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void CLRLVB ( Lvmask, Levnum )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input/Output :
/*
/*
int    Lvmask[32];    /* Level mask.
/*
/* Input :
/*
int    *Levnum;       /* Level number.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DSPLEV Display/erase all entities of a specified level.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void DSPLEV ( Levnum, Mode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Levnum;       /* Level number.
int    *Mode;         /* Display mode:
/*      0 = Erase,
/*      1 = Display
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - The specified level will be displayed/erased until REDRAW (from the Immediate Access Pop-Up Functions) is selected. To make permanent changes, change the level mask of the appropriate view using functions from this section.

GETLVB Get bit corresponding to LEVNUM in LVMASK.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void GETLVB ( Lvmask, Levnum, Value )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int    Lvmask[32];    /* Level mask.
/*
/* Input :
/*
int    *Levnum;       /* Level number.
/*
/* Output :
/*
int    *Value;        /* Value of bit.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GLLVD1

Define a new level but do not set it as active.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GLLVD1 ( Name, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char   *Name;                  /* Name of new level ( up to 6 characters long ).
                                /*
                                /* Output :
                                /*
int     *Status;               /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GLLVDF

Define a new level and set as active.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GLLVDF ( Name, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char   *Name;                  /* Name of new level ( up to 6 characters long ).
                                /*
                                /* Output :
                                /*
int     *Status;               /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GLLVSA

Activate a pre-defined level.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GLLVSA ( Name, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char   *Name;                  /* Name of level to be activated ( up to 6 characters
                                /* long ).
                                /*
                                /* Output :
                                /*
int     *Status;               /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

LEVSET Turn the level On/Off.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void LEVSET ( Xopt, Vpid, Levnum )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int *Xopt; /* 1 = ACTIVE, 2 = DISPLAY, 3 = PROTECT
int *Vpid; /* The ViewPort number.
int *Levnum; /* The level's number.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - For permanent settings use GSMSLA after this function with relevant values LVMASK and ATRIB.

PUTLVB Put bit corresponding to LEVNUM in LVMASK.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void PUTLVB ( Lvmask, Levnum, Value )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input/Output :
/*
/*
int Lvmask[32]; /* Level mask.
/*
/* Input :
/*
/*
int *Levnum; /* Level number.
int *Value; /* Value of bit.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

SETLVB Set bit corresponding to LEVNUM in LVMASK.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void SETLVB ( Lvmask, Levnum )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input/Output :
/*
/*
int Lvmask[32]; /* Level mask.
/*
/* Input :
/*
/*
int *Levnum; /* Level number.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UDBLEV

Looks in the database for entities in the given level.

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void udblev ( mode, level, wfnum, solnum, wfid, solid)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int mode; /* 1 = get no. of entities, 2 = get id's
int level; /* Level where to find all the entities
/*
/* Output :
/*
/*
int *wfnum; /* Number of wireframe entities in level
int *solnum; /* Number of solid entities in level
int *wfid; /* Pointer to entities ID's vector (wireframe)
T_CDK_SOLENTITY *solid; /* Pointer to entities ID's vector solids)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UGTALE

Get name and number of the active level.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void UGTALE ( Lname, Lnum )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
char *Lname; /* Name of the active level.
int *Lnum; /* Number of the active level.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

VIEWLM

Given a view ID, get/set its level mask.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void VIEWLM ( Mode, Id, Lvmask, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int *Mode; /* 1 = Get,
/* 2 = Set.
int *Id; /* The view entity ID ( 0 = model ).
/*
/* Output :
/*
/*
int Lvmask[32]; /* The view's level mask.
int *Status; /* 0 = O.K.
/* -1 = Invalid entity.
/* -2 = Deleted entity.
/* -3 = Invalid input.
/* -4 = View 0 not saved.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```



General Description

These routines are used to handle line attributes.

The routine **STACLA** sets the active line attributes. **ENCHAI** and **ENCHAT** are used to modify attributes of existing entities. **ENCHAI** requires input stating which attribute you wish to change and **ENCHAT** requires as input the ID number of the entity whose attributes you wish to copy.

ENCHA1	Change line attribute(s) of an entity.
---------------	--

Syntax :

```

/*- - - - - */
void ENCHAL ( Att, Num, Id, Status )
/*- - - - - */
    /*
    /* Input :
    /*
char   *Att;
    /* Name of attribute to change. Legal values are:
    /* "LINFONT":
    /* Line font.
    /* "PENNUM":
    /* Pen number.
    /* "COLOR":
    /* Color.
int     *Num;
    /* Sequential number of new attribute.
    /*
    /* Input/Output :
    /*
int     *Id;
    /* ID number of entity whose attributes are to be
    /* changed.
    /* ID of the new entity if a new entity was created
    /* ( see Duplication of Entities in Chapter 2, General
    /* Concepts ).
    /*
    /* Output :
    /*
int     *Status;
    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

ENCHAT

Assign the specified line attribute or the level of the entity IDFROM to the entities IDTO.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENCHAT ( Att, Idfrom, Numb, Idto, Chdisp, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Att; /* Name of attribute to change. Legal values are:
/* "LF" : Line font.
/* "PE" : Pen number.
/* "CO" : Color.
/* "VN" : View number.
/* "LE" : Level.
/* "AL" : All the above 5 attributes.
/* ID of the entity whose attribute is to be copied.
int *Idfrom; /* Number of entities in Idto.
int *Numb; /* Array of entity IDs line attributes or level are to
int Idto[Numb]; /* be changed.
/* 0 = Do not change displayed entity.
int *Chdisp; /* 1 = Change displayed entity immediately.
/*
/* Output :
/*
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GSMSLA

Get/set level mask, active level color, pen, line font.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GSMSLA ( Mode, Lvmask, Atrib )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Mode; /* 1 = Get.
/* 2 = Set.
/*
/* Input/Output :
/*
int Lvmask[32]; /* Level mask.
int *Atrib; /* Active level, color, pen, line font.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Pack or unpack word ATRIB using PKATRB.

PKATRB To pack/unpack the active level, font, pen and color into/from word ATRIB.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void PKATRB ( Mode, Atrib, Level, Font, Pen, Color )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Mode;    /*      1 = Pack.
/*      2 = Unpack.
/*
/* Input/Output :
/*
int    *Atrib;    /* The word as stored in the data base.
int    *Level;    /* The active level.
int    *Font;     /* The current line font.
int    *Pen;      /* The current pen.
int    *Color;    /* The current color.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

STACLA Set active line attributes.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void STACLA ( Att, Num, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char    *Att;    /* Name of attribute to change. Legal values are:
/*      "LINFONT" - Line font.
/*      "PENNUM"  - Pen number.
/*      "COLOR"   - Color.
int     *Num;    /* Sequential number of new attribute.
/*
/* Output :
/*
int     *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



General Description

This section contains general routines needed to “manage” your programs.

The **SETCR** and **SETNCR** routines are used when sending output to the alphanumeric terminal (or window). They must always be used together within a program, i.e. for every call to **SETCR** within a program, a corresponding call to **SETNCR** must exist. See the routines themselves for details of their use.

The routines **DBCLEA** and **DLDELE** delete entities from the data base. Entities can be created as one of the following:

1. Permanent: remaining in the database on a permanent basis.
2. Temporary: generally used to define other entities and removed from the database by calling DBCLEA.

The User Package provides two methods of defining permanent entities (for a detailed explanation of each method, See Chapter 1, General Concepts, Temporary and Permanent Entities):

1. Calling DBFLOF.
2. Calling DEFADD.

To find out which method of creating permanent entities is being used, run **DBFGET**.

The subroutine **DLDELE** will delete one entity. Note that the entity ID passed over as input to **DLDELE** must have been created during the current run of the user function.

To delete entities created before the current session, use the routine **DLFORC**.

The **UMDGET** and **UMDPUT** routines are used for storing and retrieving information about modal parameters. These are part of the Cimatron^{it} data base, and contain, in most cases, information about last used options in their appropriate functions etc. The user can define their own modal parameters for storing, for example, some default values, and read and write from within the applications.

ADD PATH CHG

Add old and new pathnames to the static array of ChangeExternalReference module. **This is an External User routine.**

Syntax :

```

/* - - - - - */
void add_path_chg ( old_pname, new_pname, status )
/* - - - - - */
/* Input:
/*
T_CHAR *old_pname; /* I: old pathname ;max 256 characters
T_CHAR *new_pname; /*   new pathname ;max 256 characters
/* Output:
/*
T_INT  *status; /* O: 0 - if added
/*          /* ARG_ERROR - if old_pname and new_pname are
/*          /*                               Same
/*          /* MEM_ERROR - if MAX_CHANGES exceeds
/*          /* ERROR - otherwise
/* - - - - - */

```

CDK_DSPTOL

Set the display tolerance of curves, surfaces and planar faces.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void CDK_DSPTOL( Id, DispTol, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Input:
int    *Id;          /* The entity ID.
double *DispTol;     /* The display tolerance.
/* Output:
int    *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DBCLEA

Remove all temporary entities from the screen, and delete them from the database.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void DBCLEA ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DBFGET

Get the status of DBFLAG to confirm which method of creating permanent entities is being used.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void DBFGET ( Dbflag )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int    *Dbflag;      /* .TRUE. The mechanism of COMMON /DEFENT/ and
/*                   /* DBCLEA is used.
/*                   /* .FALSE. Some other mechanism is being used.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - To set DBFLAG, use DBFLON or DBFLOF.

DBFLOF

Set the DBFLON flag to off, making all subsequently created entities permanent.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void DBFLOF ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DBFLON

Set the DBFLON flag to on, making all subsequently created entities temporary unless they are specified as permanent by an explicit call to DEFADD.

Syntax :

```
/*- - - - - */
void DBFLON ( void )
/*- - - - - */
```

DEFADD

Write down the list of permanent entities from the first position, or add given IDs to the list of permanent entities.

Syntax :

```
/*- - - - - */
void DEFADD ( Mode, Listid, Nid, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Mode;    /* 0 = Write given list of entities IDs ( LISTID )
/*               /* to the list of permanent entities from the
/*               /* first position.
/*               /* 1 = Add given list of IDs ( LISTID ) to the
/*               /* existing list.
int    *Listid;   /* Given list of IDs of the created entities.
int    *Nid;      /* Number of entity IDs in LISTID.
/*
/* Output :
/*
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

DLDEDO

Delete an entity from the database. The only entities that may be deleted are those created when DISPOF is active in the current running of the user program.

Syntax :

```
/*- - - - - */
void DLDEDO ( Id, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Id;       /* ID of entity to be deleted.
/*
/* Output :
/*
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

DLDELETE

Delete an entity from the database. Only entities created during the run of the current procedure can be deleted.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DLDELETE ( Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Id;
/* ID of entity to be deleted.
/*
/* Output :
/*
int *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DLDELETE1

Delete an entity from the database. Only entities created during the run of the current procedure can be deleted.

Advantages : The ID of deleted entities can be reused.

Disadvantages : Does not inform the association list, so do not use associated entities.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DLDELETE1 ( Id )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int Id;
/* Entity ID.
/* Returns :
/*
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DLDISP

To display an entity.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DLDISP ( Id, Dmode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Id;
/* ID of entity.
int *Dmode;
/* Display mode:
/* 0 = Erase.
/* 1 = Display.
/* 2 = Turn to complement.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DLDISPW Display entity ID in mode MODE in ALL view ports.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DLDISPW( Id, Mode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*      Input:
int  *Id;      /* The entity to display / undisplay          */
int  *Mode;    /* 0 = display off, 1 = display on, 2 = turn to complement. */
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DLFORC Delete an entity from the data base; only pickable entities can be deleted.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DLFORC ( Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*      Input :
int  *Id;      /* ID number of entity to be deleted.
/*      Output :
int  *Status;  /* See General Concepts-Status on page 1-2.
/*      DOCHPN
/*      Changepathnames. This is an External User routine.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DOCHPN Change pathnames. **This is an External User routine.**
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void dochpn( Ier )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*      Output:
int  *Ier;     /* 0-ok <0-errors
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DOCHPN1 Change wireframe pathnames. **This is an External User routine.**
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void dochpn1( )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DOCHPN2 Change solid pathnames. **This is an External User routine.**
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void dochpn2( Ier )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int      *Ier;          /* Output:
                        /* 0-ok <0-errors ( for wireframe assembly,
                        /*                      Ier always = 0 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DSPGET Confirm the status of the DISPON/DISPOF status flag.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void DSPGET ( Flag )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                        /*
                        /* Output :
                        /*
int      *Flag;          /* .TRUE. : Display on.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

IMMAWA Set the AUTO-WINDOW for all the windows.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void IMMAWA ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

INIT_PATH_CHG Initialize no_of_changes (number of external reference changes). This is a static parameter used from external user programs. **This is an External User routine.**
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void init_path_chg ( )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

REDRAW Repaint the display and rebuild the display file (optionally).
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void REDRAW ( Mode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Mode;
/*    0 = Repaint a single window from the display
/*      file.
/*    1 = Rebuild the display file and repaint all
/*      windows.
/*    2 = Current window only; rebuild the display
/*      file and repaint.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

SEGABSET May be called from a user program to kill it upon exit.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SEGABSET ( )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

SETCR Set “on” for output to the alphanumeric terminal or window from the interactive system.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SETCR ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Use with SETNCR.

SETNCR Set “off” to return to the interactive system from the alphanumeric terminal or window.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SETNCR ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - SUN Users. When running Cimatron on a SUN under SunView, SETCR / SETNCR releases / locks Cimatron's window.

TPLCLR

Clear the list of temporary entities. Subsequently, all entities become permanent.

Syntax :

```
/*- - - - - */
void TPLCLR ( void )
/*- - - - - */
```

UBLANK

Blank / unblank the entity.

Syntax :

```
/*- - - - - */
void UBLANK ( Id, Mode, )
/*- - - - - */
/*
/* Input :
/*
int *Id; /* The ID of entity.
int *Mode; /* 0 = Blank.
/* 1 = Unblank.
/*- - - - - */
```

UGTNMERF

Get the number of external references in the current .pfm file.

Syntax :

```
/*- - - - - */
void ugtnmerf( count, status )
/*- - - - - */
P_INT count; /* I - max. number of external references in arrays */
P_INT status; /* O - 0 = OK; <0 error */
/*- - - - - */
```

UMDGET

Retrieve the values of modal parameters.

Syntax :

```
/*- - - - - */
void UMDGET ( Mname, Buf )
/*- - - - - */
/*
/* Input :
/*
char *Mname; /* The name of the given modal parameters group.
/*
/* Output :
/*
int *Buf; /* The buffer holding the retrieved values.
/*- - - - - */
```

UMDPUT

Store the values of modal parameters.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UMDPUT ( Mname, Msize, Buf, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char    *Mname;                /* Name of the given modal parameter group ( should
                                /* begin from 'Q' ).
int     *Msize;                /* Length of the given modal parameter group.
int     *Buf;                  /* Buffer containing values of the given modal parameter
                                /* group.
                                /*
                                /* Output :
                                /*
int     *Status;               /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



MATHEMATICS

General Description

This section contains all the mathematical routines and functions and is divided into the following groups:

1. Conversion Routines
2. Elementary Arithmetic
3. Geometrical Routines

Double functions are to be found in each of these sections.

Conversion Routines

CVI2T Convert an integer NUMBER (± 2147483647) to text.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void CVI2T ( Number, Text, Nch )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Number;    /* Integer number to convert.
/*
/* Output :
/*
char    *Text;    /* Character variable containing the number ( should be
/* at least 11 characters long ).
int    *Nch;    /* Number of characters in TEXT.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

CVR2CV Convert a real number to an array of characters.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void CVR2CV ( Rnum, Nd, Nch, Abuf )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float    *Rnum;    /* Float number to be converted.
int    *Nd;    /* Number of digits after decimal point ( <10 ).
/*
/* Output :
/*
int    *Nch;    /* > 0 : Number of characters used for contents of
/* array, Abuf.
/* -1 : Cannot convert this float number.
char    *Abuf;    /* Array holding characters.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

Note : - ABUF must be large enough to hold the characters. (In this implementation, it is assumed that RNUM*10 is within the limits of the maximum integer.)

CVR2TS

Convert a real number to a character string (very large or small numbers will be returned in E format).

Syntax :

```

/*- - - - - */
void CVR2TS ( Rnum, Nd, Nch, Text, Maxd )
/*- - - - - */
/*
/* Input :
/*
float *Rnum; /* Float number to be converted.
int *Nd; /* Number of digits after decimal point ( <10 ).
/*
/* Output :
/*
int *Nch; /* > 0 : Number of characters in TEXT.
/* -1 : Cannot convert this float number.
char *Text; /* Character string.
/*
/* Input :
/*
int *Maxd; /* Maximum number of bytes allowed for output.
/*- - - - - */

```

CVT2I

Convert a character variable to its corresponding integer.

Syntax :

```

/*- - - - - */
void CVT2I ( Text, Nch, Number, Status )
/*- - - - - */
/*
/* Input :
/*
char *Text; /* Character variable containing sign and digits.
int *Nch; /* Number of characters in TEXT.
/*
/* Output :
/*
int *Number; /* The integer.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CVT2R

Convert a character string to the corresponding real number.

Syntax :

```

/*- - - - - */
void CVT2R ( Text, Nch, Rnum, Status )
/*- - - - - */
/*
/* Input :
/*
char *Text; /* Character variable containing sign, digits and
/* decimal point.
int *Nch; /* Number of characters in TEXT.
/*
/* Output :
/*
float *Rnum; /* Float number after conversion.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

DEGRD

Convert radians to degrees.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
double DEGRD ( Angle )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double *Angle;
/* Angle in radians.
/*
/* Returns :
/* Angle in degrees.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of DEGREE.

DEGREE

Convert radians to degrees.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
float DEGREE ( Angle )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float *Angle;
/* Angle in radians.
/*
/* Returns :
/* Angle in degrees.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

RADD

Convert degrees to radians.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
double RADD ( Angle )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double *Angle;
/* Angle in degrees.
/*
/* Returns :
/* Angle in radians.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of RADIAN.

RADIAN

Convert degrees to radians.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float RADIAN ( Angle )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Angle in degrees.
/*
/* Returns :
/*
/* Angle in radians.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

Elementary Arithmetic

These are useful routines for comparing real or double variables by a comparison to a tolerance.

In addition to the double functions introduced at the beginning of the Mathematics routines, the Elementary Arithmetic section contains two different types of double functions:

1. Double precision functions which have two corresponding non-double precision functions.

In this case, the double precision function enables both the 2D and 3D arguments of the non-double precision function to be used. Simply put 2 or 3 as the first argument to the double precision function.

For example, the functions EQPT2 and EQPT3 check, respectively, whether two 2D or two 3D vectors are equal. Depending upon its first argument, the double precision function EQPTD may also check whether two 2D or two 3D vectors are equal.

2. Double precision functions which use a higher tolerance than other functions.

For example, EQPT2 or EQPT3 compare 2D or 3D real vectors and EQPTD compares double precision vectors. These three functions all use a tolerance of type Real, which is therefore similar for the three functions. The function EQPTDD, however, uses a double precision tolerance. By convention, these functions have a double D (DD) as the last two letters of their names.

CDK_POINT_EQUAL Check if two 3D points are equal (within tolerance).

Syntax:

```
/*- - - - - */
int   cdk_point_equal ( p1, p2, tol)
/*- - - - - */
/*
/* Input :
/*
double *p1;    /* Point no. 1
double *p2;    /* Point no. 2
double  tol;   /* Tolerance
/*
/* Output :
/*
P_CDK_TRACE1 oTrace; /* The structure containing the pierce points, sorted.
/*
/* Return value :
/*   0 = not equal
/*   1 = equal
/*- - - - - */
```

EQMAT Check whether two 3D matrices are equal.

Syntax :

```
/*- - - - - */
void EQMAT ( Mat1, Mat2, Flag )
/*- - - - - */
/*
/* Input :
/*
float  Mat1[9];
float  Mat2[9];
/* Two 3x3 matrices.
/*
/* Output :
/*
int    *Flag;
/*   0 : MAT1 = MAT2.
/*  -1 : MAT1 and MAT2 are different matrices.
/*  -2 : MAT1 and MAT2 define the same plane
/*       ( Z vectors are equal ).
/*- - - - - */
```

EQMATD

Check whether two matrices are equal.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void EQMATD ( Mat1, Mat2, Flag )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
double Mat1[9];
double Mat2[9];
/* Two 3x3 matrices.
/*
/* Output :
/*
/*
int *Flag;
/* 0 = MAT1 = MAT2.
/* -1 = MAT1 and MAT2 are different matrices.
/* -2 = MAT1 and MAT2 define the same plane
/* ( Z vectors are equal ).
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of EQMAT.

EQPT2

Check whether two 2D points are equal.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQPT2 ( P1, P2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float P1[2];
float P2[2];
/* Two 2D points.
/*
/* Returns :
/* .TRUE.
/* Points are equal.
/* .FALSE.
/* Points are not equal.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

EQPT3

Check whether two 3D points are equal.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQPT3 ( P1, P2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float P1[3];
float P2[3];
/* Two 3D points.
/*
/* Returns :
/* .TRUE.
/* Points are equal.
/* .FALSE.
/* Points are not equal.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

EQPTD

Check whether two points are equal.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQPTD ( Dim, P1, P2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Dim;
/* Dimension of the points:
/* 2 = 2D
/* 3 = 3D
double P1[Dim];
/*
double P2[Dim];
/* Two points.
/*
/* Returns :
/* .TRUE.
/* Points are equal.
/* .FALSE.
/* Points are not equal.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - Double precision version of EQPT2 and EQPT3.
 - Uses the same tolerance as EQPT2 and EQPT3.

EQPTDD

Check whether two points are equal.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQPTDD ( Dim, P1, P2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Dim;
/* Dimension of the points:
/* 2 = 2D
/* 3 = 3D
double P1[Dim];
/*
double P2[Dim];
/* Two points.
/*
/* Returns :
/* .TRUE.
/* Points are equal.
/* .FALSE.
/* Points are not equal.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - Double precision version of EQPT2 and EQPT3.
 - Uses a higher tolerance than EQPT2 and EQPT3.

EQV0DD

Check whether A is equal to 0.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQV0DD ( A )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
double *A;                      /*
                                /*
                                /* Returns :
                                /* .TRUE.
                                /* A = 0 ( i.e. ABS( A ) .LT. tolerance value ).
                                /* .FALSE.
                                /* A 0 ( i.e. ABS( A ) .GT. tolerance value ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - Double precision version of EQVAL0.
 - Uses a higher tolerance than EQVAL0.

EQVAL

Check whether A and B are equal.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQVAL ( A, B )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
float *A;                       /*
float *B;                       /* Two float numbers.
                                /*
                                /* Returns :
                                /* .TRUE.
                                /* A = B
                                /* .FALSE.
                                /* A  B
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

EQVAL0

Check whether A is equal to 0.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQVAL0 ( A )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
float *A;                       /*
                                /*
                                /* Returns :
                                /* .TRUE.
                                /* A = 0 ( i.e. ABS( A ) .LT. tolerance value ).
                                /* .FALSE.
                                /* A 0 ( i.e. ABS( A ) .GT. tolerance value ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

EQVALD

Check whether A and B are equal.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQVALD ( A, B )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
                                /*
double *A;                      /*
double *B;                      /* Two double precision numbers.
                                /*
                                /* Returns :
                                /* .TRUE.
                                /* A = B
                                /* .FALSE.
                                /* A B
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - Double precision version of EQVAL.
 - Uses the same tolerance as EQVAL.

EQVL0D

Check whether A is equal to 0.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQVL0D ( A )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
                                /*
double *A;                      /* Double precision number to check.
                                /*
                                /* Returns :
                                /* .TRUE.
                                /* A = 0 ( i.e. ABS( A ) .LT. tolerance value ).
                                /* .FALSE.
                                /* A 0 ( i.e. ABS( A ) .GT. tolerance value ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - Double precision version of EQVAL0.
 - Uses the same tolerance as EQVAL0.

EQVLDD

Check whether A and B are equal.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int EQVLDD ( A, B )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
double *A;
double *B;
/* Two double precision numbers.
/*
/* Returns :
/* .TRUE.
/* A = B
/* .FALSE.
/* A B
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - Double precision version of EQVAL.
 - Uses a higher tolerance than EQVAL.

M3CROS

Multiply M1 by M2.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void M3CROS ( M1, M2, M3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float M1[9];
float M2[9];
/* 3x3 matrices.
/*
/* Output :
/*
/* Matrix containing cross product of M1 and M2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

M3CROT

Multiply M1 by the transpose of M2.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void M3CROT ( M1, M2, Mx )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float M1[9];
float M2[9];
/* 3x3 matrices.
/*
/* Output :
/*
/* Matrix containing cross product of M1 and transpose
/* of M2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

M3CRSD

Multiply M1 by M2.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void M3CRSD ( M1, M2, M3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
double M1[9];
double M2[9];
/* 3x3 matrices.
/*
/* Output :
/*
double M3[9];
/* Matrix containing cross product of M1 and M2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of M3CROS.

M3CRTD

Multiply M1 by the transpose of M2.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void M3CRTD ( M1, M2, Mx )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
double M1[9];
double M2[9];
/* 3x3 matrices.
/*
/* Output :
/*
double Mx[9];
/* Matrix containing cross product of M1 and transpose
/* of M2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of M3CROT.

M3DET

Find the determinant of a 3x3 matrix M.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
float M3DET ( M )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float M[9];
/* A 3x3 matrix.
/*
/* Returns :
/* Determinant of the matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

M3DETD

Find the determinant of a 3x3 matrix M.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
double M3DETD ( M )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double M[9];
/* A 3x3 matrix.
/*
/* Returns :
/* Determinant of the matrix.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of M3DET.

M3INV

Find the inverse matrix of a 3x3 matrix.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void M3INV ( M, Minv, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float M[9];
/* Given 3x3 matrix
/*
/* Output :
/*
float Minv[9];
/* Inverse matrix ( if Status = 0 ),
int *Status;
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

M3INVD

Find the inverse matrix of a 3x3 matrix.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void M3INVD ( M, Minv, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double M[9];
/* Given 3x3 matrix
/*
/* Output :
/*
double Minv[9];
/* Inverse matrix ( if Status = 0 ),
int *Status;
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

M3LNED

Find the solution to 3 linear equations:

$$\begin{pmatrix} M(1) & M(2) & M(3) \\ M(4) & M(5) & M(6) \\ M(7) & M(8) & M(9) \end{pmatrix} \begin{pmatrix} R(1) \\ R(2) \\ R(3) \end{pmatrix} = \begin{pmatrix} V(1) \\ V(2) \\ V(3) \end{pmatrix}$$

Syntax :

```

/*- - - - - */
void M3LNED ( M, V, R, Status )
/*- - - - - */
/*
/* Input :
/*
/* 3x3 matrix.
/* 3D vector.
/*
/* Output :
/*
/* 3D vector containing the results ( R is only
/* meaningful when STATUS=0 ).
/*
int *Status;
/* 0 = A unique solution.
/* -1 = Matrix irregular, no solution.
/*- - - - - */

```

Note : - Double precision version of M3LNEQ.

M3LNEQ

Find the solution to 3 linear equations:

$$\begin{pmatrix} M(1) & M(2) & M(3) \\ M(4) & M(5) & M(6) \\ M(7) & M(8) & M(9) \end{pmatrix} \begin{pmatrix} R(1) \\ R(2) \\ R(3) \end{pmatrix} = \begin{pmatrix} V(1) \\ V(2) \\ V(3) \end{pmatrix}$$

Syntax :

```

/*- - - - - */
void M3LNEQ ( M, V, R, Status )
/*- - - - - */
/*
/* Input :
/*
/* 3x3 matrix.
/* 3D vector.
/*
/* Output :
/*
/* 3D vector containing the results ( R is only
/* meaningful when Status=0 ).
/*
int *Status;
/* 0 = A unique solution.
/* -1 = Matrix irregular, no solution.
/*- - - - - */

```

M3VEC

Multiply a 3D-vector by a 3x3 matrix.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void M3VEC ( Rot, V1, V2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float Rot[9]; /* 3x3 matrix.
float V1[3]; /* 3D vector.
/*
/* Output :
/*
float V2[3]; /* Product vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

M3VECD

Multiply a 3D-vector by a 3x3 matrix.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void M3VECD ( Rot, V1, V2, )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
double Rot[9]; /* 3x3 matrix.
double V1[3]; /* 3D vector.
/*
/* Output :
/*
double V2[3]; /* Product vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

PIFACTEvaluate factor of π .**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float PIFACT ( Factor )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float *Factor; /* Factor of p.
/*
/* Returns :
/* p * FACTOR.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V2ADD Add a 2D scaled vector to another 2D vector: $V3 = V1 + (SC \cdot V2)$
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V2ADD ( V1, V2, Sc, V3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float V1[2];
float V2[2];
float *Sc;
/* Two 2D vectors.
/* Scale factor applied to V2.
/*
/* Output :
/*
float V3[2];
/* Resulting 2D vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V2CROS Calculate the vector product of two 2D vectors: $V1 * V2$.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float V2CROS ( V1, V2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float V1[2];
float V2[2];
/* Two 2D vectors.
/*
/* Returns :
/* Vector product.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V2CRSD Calculate the vector product of two vectors: $V1 * V2$.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
double V2CRSD ( V1, V2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double V1[2];
double V2[2];
/* Two vectors.
/*
/* Returns :
/* Vector product.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V2CROS.

V2DOTCalculate the dot product of two 2D vectors: $V1 \cdot V2$.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
float V2DOT ( V1, V2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float V1[2];
float V2[2];
/* Two 2D vectors.
/*
/* Returns :
/* Dot product of the two vectors.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V2LNED

Find the solution to 2 linear equations:

$$\begin{pmatrix} V1(1) & V2(1) \\ V1(2) & V2(2) \end{pmatrix} \cdot \begin{pmatrix} R(1) \\ R(2) \end{pmatrix} = \begin{pmatrix} B(1) \\ B(2) \end{pmatrix}$$

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void V2LNED ( V1, V2, B, R, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
double V1[2];
double V2[2];
double B[2];
/* Two vectors defining a 2x2 matrix.
/* The right side of the above equation.
/*
/* Output :
/*
double R[2];
/* 2D vector containing the solutions ( only meaningful
/* for Status=0 ).
int *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V2LNEQ.

V2LNEQ

Find the solution to 2 linear equations:

$$\begin{pmatrix} V1(1) & V2(1) \\ V1(2) & V2(2) \end{pmatrix} * \begin{pmatrix} R(1) \\ R(2) \end{pmatrix} = \begin{pmatrix} B(1) \\ B(2) \end{pmatrix}$$

Syntax :

```

/*- - - - - */
void V2LNEQ ( V1, V2, B, R, Status )
/*- - - - - */
/*
/* Input :
/*
float    V1[2];
float    V2[2];
float    B[2];
/* Two vectors defining a 2x2 matrix.
/* The right side of the above equation.
/*
/* Output :
/*
float    R[2];
/* 2D vector containing the solutions ( only meaningful
/* for Status=0 ).
int      *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

V2SCAL

Scale a 2D vector by a factor SC.

Syntax :

```

/*- - - - - */
void V2SCAL ( V1, Sc )
/*- - - - - */
/*
/* Input/Output :
/*
float    V1[2];
/* 2D vector.
/* The scaled vector.
/*
/* Input :
/*
float    *Sc;
/* Scale factor.
/*- - - - - */

```

V2SIZD

Find the length of the 2D vector V1.

Syntax :

```

/*- - - - - */
double V2SIZD ( V1 )
/*- - - - - */
/*
/* Input :
/*
double   V1[2];
/* Vector.
/*
/* Returns :
/* Length of the vector.
/*- - - - - */

```

Note : - Double precision version of V2SIZE.

V2SIZE Find the length of the 2D vector V1.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float V2SIZE ( V1 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
float   V1[2];
                                /* 2D vector.
                                /*
                                /* Returns :
                                /* Length of the vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V2SUB Subtract a 2D vector from another 2D vector: $V3 = V1 - V2$
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V2SUB ( V1, V2, V3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
float   V1[2];
float   V2[2];
                                /* Two 2D vectors.
                                /*
                                /* Output :
                                /*
float   V3[2];
                                /* Resulting 2D vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V2UNIT Change a 2D vector into a unit vector.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V2UNIT ( Vec, Lng, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input/Output :
                                /*
float   Vec[2];
                                /* 2D vector.
                                /* Resulting 2D unit vector.
                                /*
                                /* Output :
                                /*
float   *Lng;
int     *Status;
                                /* Length of the input vector, Vec.
                                /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V2ZERD

Check whether a 2D vector is a zero vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int V2ZERD ( V )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double V[2];
/* 2D vector.
/*
/* Returns :
/* .TRUE.
/* V is a zero vector.
/* .FALSE.
/* V is not a zero vector.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V2ZERO.

V2ZERO

Check whether a 2D vector is a zero vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int V2ZERO ( V )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float V[2];
/* 2D vector.
/*
/* Returns :
/* .TRUE.
/* V is a zero vector.
/* .FALSE.
/* V is not a zero vector.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3ADD

Add a 3D scaled vector to another 3D vector:
 $V3 = V1 + (SC \cdot V2)$

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V3ADD ( V1, V2, Sc, V3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float V1[3];
float V2[3];
float *Sc;
/* Two 3D vectors.
/* Scale factor applied to V2.
/*
/* Output :
/*
float V3[3];
/* Resulting 3D vector.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3CROSCalculate the vector product of two 3D vectors: $V3 = V1 * V2$.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V3CROS ( V1, V2, V3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float V1[3];
float V2[3];
/* Two 3D vectors.
/*
/* Output :
/*
/*
float V3[3];
/* Resulting 3D vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3CRSDCalculate the vector product of two 3D vectors: $V3 = V1 * V2$.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V3CRSD ( V1, V2, V3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
double V1[3];
double V2[3];
/* Two 3D vectors.
/*
/* Output :
/*
/*
double V3[3];
/* Resulting 3D vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V3CROS.

V3DOTCalculate the dot product of two 3D vectors: $V1 \cdot V2$.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float V3DOT ( V1, V2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float V1[3];
float V2[3];
/* Two 3D vectors.
/*
/* Returns :
/* Dot product of the two vectors.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3SCAL Scale a 3D vector by a factor SC.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V3SCAL ( V1, Sc )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input/Output :
                                /*
float   V1[3];                 /* 3D vector.
                                /* The scaled vector.
                                /*
                                /* Input :
                                /*
float   *Sc;                   /* Scale factor.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3SIZD Calculate the length of 3D vector V1.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
double V3SIZD ( V1 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
double  V1[3];                 /* 3D vector.
                                /*
                                /* Returns :
                                /* Length of the vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V3SIZE.

V3SIZE Calculate the length of 3D vector V1.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float V3SIZE ( V1 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
float   V1[3];                 /* 3D vector.
                                /*
                                /* Returns :
                                /* Length of the vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3SUBSubtract a 3D vector from another 3D vector: $V3 = V1 - V2$ **Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V3SUB ( V1, V2, V3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float   V1[3];
float   V2[3];
/* Two 3D vectors.
/*
/* Output :
/*
float   V3[3];
/* Resulting 3D vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3UNIT

Change a 3D vector into a unit vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V3UNIT ( Vec, Lng, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input/Output :
/*
/*
float   Vec[2];
/* 3D vector.
/* Resulting 3D unit vector.
/*
/* Output :
/*
float   *Lng;
int     *Status;
/* The length of the input vector, Vec.
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3ZERD

Check whether 3D vector is a zero vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int V3ZERD ( V )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
double  V[3];
/* 3D vector.
/*
/* Returns :
/* .TRUE.
/* V is a zero vector.
/* .FALSE.
/* V is not a zero vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V3ZERO.

V3ZERO

Check whether 3D vector is a zero vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int V3ZERO ( V )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
float   V[3];                  /* 3D vector.
                                /*
                                /*
                                /* Returns :
                                /* .TRUE.
                                /* V is a zero vector.
                                /* .FALSE.
                                /* V is not a zero vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

VASDDD

Add a scaled vector to another vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void VASDDD ( Dim, V1, V2, Sc, V3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int      *Dim;                 /* Dimension of the vectors.
                                /*   2 = 2D
                                /*   3 = 3D
double   V1[Dim];              /*
double   V2[Dim];              /* Two vectors.
double   *Sc;                  /* Scale factor applied to V2.
                                /*
                                /* Output :
                                /*
double   V3[Dim];              /* Resulting vector.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V2ADD and V3ADD.

VDOTDCalculate the dot product of two vectors: $V1 \bullet V2$.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
double VDOTD ( Dim, V1, V2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Dimension of the vectors.
/*      2 = 2D
/*      3 = 3D
/*
/* Two vectors.
/*
/* Returns :
/* Dot product of the two vectors.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V2DOT and V3DOT.

VECSCL

Scale a vector by a factor.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VECSCL ( Dim, V1, Sc )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Dimension of the vectors.
/*      2 = 2D
/*      3 = 3D
/*
/* Input/Output :
/*
/* Vector.
/* The scaled vector.
/*
/* Input :
/*
/* Scale factor.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V2SCAL and V3SCAL.

VSUBDSubtract a vector from another vector: $V3 = V1 - V2$ **Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VSUBD ( Dim, V1, V2, V3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Dimension of the vectors.
/*   2 = 2D
/*   3 = 3D
/*
/* Two vectors.
/*
/* Output :
/*
/* Resulting vector.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V2SUB and V3SUB.

VUNITD

Change a vector into a unit vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VUNITD ( Dim, Vec, Lng, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Dimension of the vectors.
/*   2 = 2D
/*   3 = 3D
/*
/* Input/Output :
/*
/* Vector.
/* Resulting unit vector.
/*
/* Output :
/*
/* The length of the input vector, VEC.
/*
/* Status;
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Double precision version of V2UNIT and V3UNIT.

Geometrical Routines

CDK_WRKPLN_3PNT_ROT

Set a new work plane, defined as passing through 3 points. A work plane will not be created ONLY if there is a work plane with the same rotation that the 3 points define.

Syntax :

```
/*- - - - - */
void cdk_wrkpln_3pnt_rot ( Point1, Point2, Point3, Status )
/*- - - - - */
/*
/* Input :
/*
double Point1; /* The coordinates of the 1 point (MODEL).
double Point2; /* The coordinates of the 2 point (MODEL).
double Point3; /* The coordinates of the 3 point (MODEL).
/*
/* Output :
/*
int * Status; /* Return value :
/* 0 = OK else ERROR
/*- - - - - */
```

CKPT2D

Check whether points are lying on a plane within a given tolerance.

Syntax :

```
/*- - - - - */
void CKPT2D ( tol, pt, np, plane, ierflg )
/*- - - - - */
/*
/* Input :
/*
double *tol; /* A given tolerance.
double pt[]; /* List of 3D points.
int *np; /* No. of points.
/*
/* Output :
/*
double plane[4]; /* The plane coefficients :  $Ax + By + Cz + D = 0$ ,
/*  $A^2 + B^2 + C^2 = 1$ .
int *ierflg; /* Error flag :
/* 0 = The points defined plane.
/* 1 = The points are collinear, So they do not
/* define a plane.
/* -1= The points defined plane, but they are not
/* lying on it within the given tolerance.
/*- - - - - */
```

DISAXP

Find the distance between an axis and a point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DISAXP ( Mode, Orig, Dir, Pt, R, Dist )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Mode;      /*    2 = 2D space.
/*    3 = 3D space.
float  Orig[3];    /* Coordinates of axis origin.
float  Dir[3];     /* Point indicating axis direction.
float  Pt[3];      /* Coordinates of the point.
/*
/* Output :
/*
float  R[3];       /* Coordinates of closest point on the axis.
float  *Dist;      /* Distance between axis and point.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DIST2

Find the distance between two 2D points.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float DIST2 ( P1, P2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float  P1[2];      /*
float  P2[2];      /* Two 2D points.
/*
/* Returns :
/* Distance between the points.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DIST3

Find the distance between two 3D points.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float DIST3 ( P1, P2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float  P1[3];      /*
float  P2[3];      /* Two 3D points.
/*
/* Returns :
/* Distance between the points.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DISTD

Find the distance between two points.

Syntax :

```

/*- - - - - */
double DISTD ( Dim, P1, P2 )
/*- - - - - */
/*
/* Input :
/*
/* Dimension of the vectors.
/* 2 = 2D
/* 3 = 3D
/*
double P1[Dim];
double P2[Dim];
/* Two points.
/*
/* Returns :
/* Distance between the points.
/*
/*- - - - - */

```

Note : - Double precision version of DIST2 and DIST3.

IND2C

Calculate the intersection points of two circles.

Syntax :

```

/*- - - - - */
void IND2C ( Cent1, R1, Cent2, R2, Pt, Np )
/*- - - - - */
/*
/* Input :
/*
/* Center of first circle.
/* Radius of the first circle.
/* Center of the second circle.
/* Radius of the second circle.
/*
double Cent1[2];
double *R1;
double Cent2[2];
double *R2;
/*
/*
/* First intersection point ( if it exists ).
/* Second intersection point ( if it exists ).
/*
double Pt[4];
/*
/* Output :
/*
/* 2 = There are two intersection points.
/* 1 = There is only one intersection point.
/* 0 = There are no intersection points.
/* -1 = The circles coincide.
/*
/*- - - - - */

```

Note : - Double precision version of INT2C.

INPOLD

Find whether a point lies within a closed polygon.

Syntax :

```

/*- - - - - */
void INPOLD ( Tol, Polbuf, Pntbuf, Num, Pt, Flag )
/*- - - - - */
/*
/* Input :
/*
/*
double *Tol;      /* A given tolerance.
double *Polbuf;   /* The buffer contains the polygons.
int    Pntbuf[Num]; /* The number of points in each polygon.
int    *Num;      /* The number of polygons.
double Pt[2];     /* The tested 2D point.
/*
/* Output :
/*
int    *Flag;     /* 1 = The point lies inside the area.
/*               /* 2 = The point lies on the boundary.
/*               /* 3 = The point lies outside the polygon area.
/*- - - - - */

```

INT2C

Calculate the intersection points of two circles.

Syntax :

```

/*- - - - - */
void INT2C ( Cent1, R1, Cent2, R2, Pt, Np )
/*- - - - - */
/*
/* Input :
/*
/*
float Cent1[2];   /* Center of first circle.
float *R1;        /* Radius of the first circle.
float Cent2[2];   /* Center of the second circle.
float *R2;        /* Radius of the second circle.
/*
/*
/*
float Pt[4];      /* First intersection point ( if it exists ).
/*               /* Second intersection point ( if it exists ).
/*
/* Output :
/*
int    *Np;       /* 2 = There are two intersection points.
/*               /* 1 = There is only one intersection point.
/*               /* 0 = There are no intersection points.
/*               /* -1 = The circles coincide.
/*- - - - - */

```

LSQRPD

Find the least square plane.

Syntax :

```

/*- - - - - */
void LSQRPD ( pts, np, plane, flag )
/*- - - - - */
/*
/* Input :
/*
double pts[]; /* List of 3D points.
int *np; /* No. of points.
/*
/* Output :
/*
double plane[4]; /* The plane coefficients : Ax + By + Cz + D = 0 ,
/* 2 2 2
/* A + B + C = 1 .
int *flag; /* Error flag :
/* = 0 : The points defined plane.
/* = 1 : The points are collinear, So they do not
/* define a plane.
/*- - - - - */

```

PLCOF

Define plane coefficients from three points.

Syntax :

```

/*- - - - - */
void PLCOF ( P1, P2, P3, Pl, Status )
/*- - - - - */
/*
/* Input :
/*
float P1[3]; /*
float P2[3]; /*
float P3[3]; /* Three 3D points.
/*
/* Output :
/*
float Pl[4]; /* Plane coefficients.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

PLCOFD

Define plane coefficients from three points.

Syntax :

```

/*- - - - - */
void PLCOFD ( P1, P2, P3, Pl, Status )
/*- - - - - */
/*
/* Input :
/*
double P1[3]; /*
double P2[3]; /*
double P3[3]; /* Three points.
/*
/* Output :
/*
double Pl[4]; /* Plane coefficients.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - Double precision version of PLCOF.

V2ANGL

Find the angle between two 2D vectors V1, V2.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V2ANGL ( V1, V2, Ang, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float   V1[2];
float   V2[2];
/* Two 2D vectors.
/*
/* Output :
/*
/*
float   *Ang;
int     *Status;
/* Angle between V1 and V2.
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V2PRP

Calculate a perpendicular vector, to the right or left of a given vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V2PRP ( V1, Dir, V2 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float   V1[2];
int     *Dir;
/* Direction:
/*   0 = Right
/*   1 = Left
/*
/* Output :
/*
/*
float   V2[2];
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3ANGL

Find the angle between two 3D vectors V1, V2.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void V3ANGL ( V1, V2, Ang, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float   V1[3];
float   V2[3];
/* Two 3D vectors.
/*
/* Output :
/*
/*
float   *Ang;
int     *Status;
/* The angle between V1 and V2.
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

V3LIN

Determine whether three 3D points lie on one line.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int V3LIN ( V1, V2, V3 )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float V1[3];
float V2[3];
float V3[3];
/* Three 3D points.
/*
/* Returns :
/* .TRUE.
/* The points lie on one line.
/* .FALSE.
/* The points do not lie on one line.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

WVCHK

Check if an entity is lying on the work plane.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void WVCHK ( Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int *Id;
/* Entity ID.
/*
/* Output :
/*
/*
int *Status;
/* 0 = Entity is a 2D curve lying on the work plane.
/* -1 = Entity is a 2D curve that does not lie
/* on the work plane or on a parallel one.
/* -2 = Entity is a 2D curve that is lying on a
/* plane parallel to, and below the work plane
/* ( Z < DEPTH ).
/* -3 = Entity is a 2D curve that is lying on a
/* plane parallel and above the work plane
/* ( Z > DEPTH ).
/* -4 = Entity is a 3D curve.
/* -5 = Entity is a surface.
/* -6 = Entity is another non-geometrical entity.
/* -7 = Error in recognizing entity ID.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - Z = The Z value of the plane on which the 2D entity is lying.
 - DEPTH = The value of Z for the work plane in the work coordinate system.



The following routines are used for enabling (unmasking) and picking entities.

In the interactive system the **SM** (Selection Mask) option allows you to “enable” (unmask) certain types of entities only for picking, if not all types of entities are required. In the User Package this selection is performed via the **ENABLE** routines. These routines allow the enabling of arcs, points, curves and lines. Thus, when one of the **PICK** routines is called, only those entity types that have been enabled will be included in the **PICK**.

Routines **SLCLR**, **SLPOP**, **SLPSHI** and **SLRJCT** work with the selection list. This is an internal buffer for storing information about ‘picked’ entities. For each entity, it contains the ID number, attention mark location, etc.. In some cases these routines may be used instead of **UMPICK** and **UPICK**.

Receive input from the mouse or keyboard.

```

/*- - - - - */
void CDK_CURIN ( Option, X, Y, Respon )
/*- - - - - */
/*
/* Input :
/*
int      *Option;
/*      0 = Normal
/*      1 = Limited to graphics screen
/*      2 = Limited to control panel
/*      3 = Keyboard input only
/*
/* Output :
/*
int      *X;
/* Device coordinate X.
int      *Y;
/* Device coordinate Y.
int      *Respon;
/*      0 = ( 1 ) graphic input
/*      -1 = ( 2 ) exit
/*      -2 = ( 1 - 2 ) reject
/*      -3 = ( 3 ) submenu
/*      -4 = ( 2 - 3 ) pop-up ( control panel )
/*      -8 = ( 1 - 3 ) ( control panel )
/*      -9 = ( 1-2-3 ) glmenu ( control panel )
/*      > 0 = ASCII code
/*- - - - - */

```

CDK_MENU_POPUP

Call the popup menu from a user application.

Syntax :

```

/*- - - - - */
int CDK_MENU_POPUP ( Ds )
/*- - - - - */
/*
/* Input :
/*
int      Ds[2];    /* Device coordinates - X, Y.
/* Returns : user response
/*- - - - - */

```

CDK_MOUSE_INPUT

Receive input from the mouse.

Syntax :

```

/*- - - - - */
int CDK_MOUSE_INPUT( Buttg, Ds, Pt )
/*- - - - - */
/*
/* Output :
/*
int      *Buttg;    /* Mouse Button Event :
/* = 0 : No Button.
/* = 1 : Left Push.
/* = 2 : Mid Push.
/* = 3 : Right Push.
/* = 4 : Left Release.
/* = 5 : Mid Release.
/* = 6 : Right Release.
/*
/* Input/Output :
/*
int      Ds[2];    /* Device coordinates - X, Y.
/* Input - Set the cursor on the X, Y coordinates.
/* Output - Receive cursor's X, Y coordinates.
/*
/* Output :
/*
double   Pt[3];    /* Coordinates of 3D points in current work coordinate
/* system.
/* Returns - Key for mouse response :
/* = 0 (L)      graphic input.
/* = -1 (M)     exit.
/* = -2 (L + M) reject.
/* = -3 (R)     submenu.
/* = -4 (M + R) pop-up (control panel).
/* = -8 (L + R)  (control panel).
/* = -9 (L+M+R) glmenu (control panel).
/*- - - - - */

```

CDK_PICK_WINDOW

Pick a window.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_PICK_WINDOW ( opt )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    opt;    /* = 1 : with prompt,    = 0 : without prompt.
/* Returns :
/* > 0 : The view port ID of the picked window .
/* < 0 : Mouse response.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENA3DC

Enable picking of 3D curves.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENA3DC ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENAALL

Enable picking of all types of entities.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENAALL ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENAARC

Enable picking of arcs.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENAARC ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENABLE

Enable the selection of class and type lists.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENABLE( Class, Type )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
      /*
      /* Input:
      /*
      /*
int   Class[];      /* Class list ( up to 999 ).
int   Type[];       /* Type list ( up to 999 ), if type = -1 for class in
      /* the list - all types are enable.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - 1 See Classes and Types on page 1-1.
 - 2 Example :
 If we want to enable picking lines, circles, conics, 2d & 3d
 splines, cubic splines, 2d & 3d compost, we may use :
 static int class[3] = {1, 2, -1};
 static int type[11] = {1, 2, 3, 4, 15, -1, 2, 3, 4, 15, -1};
 ENABLE (Class, Type);

ENACRV

Enable picking of 2D curves.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENACRV ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENAINS

Enable picking of instances.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENAINS ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENALIN

Enable picking of lines.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENALIN ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENAPLF

Enable picking of planar faces.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENAPLF ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENAPT Enable picking of points.

Syntax

```
/*- - - - - */  
void ENAPT ( void )  
/*- - - - - */
```

ENASAT Enable picking of surfaces.

Syntax :

```
/*- - - - - */  
void ENASAT ( void )  
/*- - - - - */
```

ENAUCS Enable picking of UCS.

Syntax :

```
/*- - - - - */  
void ENAUCS ( void )  
/*- - - - - */
```

GETPD

Get the coordinates of a picked point.

Syntax :

```

/*- - - - - */
void GETPD ( Sys, Point, Option, Respon )
/*- - - - - */
/*
/* Input :
/*
int    *Sys;
/*    1 = Point's coordinates given in the Model
/*    coordinate system.
/*    2 = Point's coordinates given in the current
/*    work coordinate system.
/*
/* Output :
/*
double  Point[3];
/* Coordinates of the 3D point, in current work
/* coordinate system.
/*
/* Input/Output :
/*
int    *Option;
/* Method to be used to pick the point.
/*    0 = Choose from submenu.
/*    1 = SCREEN position.
/*    2 = END point of an entity
/*    3 = MID point of an entity.
/*    4 = INTERSection of two curves.
/*    5 = CENTER of an arc or a conic section.
/*    6 = PIERCE point of a curve and the work plane.
/*    7 = CLOSEst point on a curve to the
/*        indicated point.
/*    8 = PICK an existing point.
/*    9 = KEY IN the coordinates of a point.
/*   11 = SURFace point closest to indication.
/*   12 = SURF-B - one of the corners of a surface.
/*   13 = SRF-C - one of the corners of a surface.
/*   14 = SURF-X - intersection of displayed curves.
/*   16 = UCSORG - UCS origin.
/*   17 = INSORG - instance origin.
/* Option chosen ( if Respon=0 ). This value may differ
/* from the input value, if changed by the operator via
/* the submenu.
/*
/* Output :
/*
int    *Respon;
/* See Chapter 1, General Concepts, Mouse Response.
/*- - - - - */

```

Notes : - The OPTION parameter has both an input and output value.
 - Double precision version of GETPT.

GETPT

Get the coordinates of a picked point.

Syntax :

```

/*- - - - - */
void GETPT ( Point, Option, Respon )
/*- - - - - */
/*
/* Output :
/*
float   Point[3]; /* Coordinates of the 3D point, in current work
/* coordinate system.
/*
/* Input/Output :
/*
int     *Option;  /* Method to be used to pick the point.
/*      0 = Choose from submenu.
/*      1 = SCREEN position.
/*      2 = END point of an entity
/*      3 = MID point of an entity.
/*      4 = INTERSection of two curves.
/*      5 = CENTER of an arc or a conic section.
/*      6 = PIERCE point of a curve and the work plane.
/*      7 = CLOSEst point on a curve to the
/*            indicated point.
/*      8 = PICK an existing point.
/*      9 = KEY IN the coordinates of a point.
/*     11 = SURFace point closest to indication.
/*     12 = SURF-B - one of the corners of a surface.
/*     13 = SRF-C - one of the corners of a surface.
/*     14 = SURF-X - intersection of displayed curves.
/*     16 = UCSORG - UCS origin.
/*     17 = INSORG - instance origin.
/* Option chosen ( if Respon=0 ). This value may differ
/* from the input value, if changed by the operator via
/* the submenu.
/*
/* Output :
/*
int     *Respon;  /* See Chapter 1, General Concepts, Mouse Response.
/*- - - - - */

```

Note : - The OPTION parameter has both an input and output value.

GINPTD

Get the display coordinates of the picked point.

Syntax :

```

/*- - - - - */
void GINPTD ( X, Y, Respon )
/*- - - - - */
/*
/* Output :
/*
int     *X;
int     *Y;
int     *Respon; /* The display coordinates of the picked point.
/* See Chapter 1, General Concepts, Mouse Response.
/*- - - - - */

```

MPICKP Multiple selection option.
Syntax :

```

/*- - - - - */
void MPICKP ( Iopt, Text, Respon )
/*- - - - - */
/*
/* Input :
/*
int    *Iopt;    /* 0 = All options applicable.
/*              /* 1 = SINGLE, BOX and UNPICK only.
char    *Text;    /* Text to be displayed as prompt.
/*
/* Output :
/*
int    *Respon;    /* See Chapter 1, General Concepts, Mouse Response.
/*- - - - - */
Note :    - Use this routine with SLLEN and SLPOP.

```

PICK Listen to input devices. If a graphical input is activated, pick and push the ID and coordinates of the picked entity into the SELECTION LIST.
Syntax :

```

/*- - - - - */
void PICK( Res )
/*- - - - - */
/*
/* Output:
/*
int    *Res
/* = 0. An entity was picked
/* = -1. Mouse exit
/* = -2. Mouse reject
/* = -3. Mouse submenu
/* > 0. when in IMMEDIATE RETURN MODE, the modal
/*       field number which was changed.
/*- - - - - */

```

SELARR Chose one of two opposite displayed directions.
Syntax :

```

/*- - - - - */
void SELARR ( Sys, Baspnt, Dirvec, Respon )
/*- - - - - */
/*
/* Input :
/*
int    *Sys;
/* Coordinate system in which BASPNT and DIRVEC are
/* given:
/* 1 = Model,
/* 2 = Work coordinate system.
double Baspnt[3];
double Dirvec[3];
/* Base point of arrows.
/* Direction of arrows.
/*
/*
int    *Respon;
/*- - - - - */

```

SETMF Enable picking within an instance.
Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SETMF ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

SLCLR Clear the selection list.
Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SLCLR ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

SLEN Return the number of entities in the selection list.
Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SLEN ( Length )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int *Length; /* The number of entries in the selection list.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

SLPOP Pop an entry from the selection list.
Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void SLPOP ( Id, X, Y )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int *Id; /* ID of entity, if ID = -1 the list is empty,
int *X; /*
int *Y; /* Attention mark location ( DISPLAY coordinate system )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
```

SLPOP3 Pop an entry from the selection list and return the XYZ coordinates of the picked point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void SLPOP3( Id, X, Y, Dfpt )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output:
/*
/*
int    *Id;      /* The entity ID.
int    *X;       /* Attention mark location ( DEVICE UNITS ).
int    *Y;       /* Attention mark location ( DEVICE UNITS ).
float  Dfpt[3]   /* XYZ coordinates of the picked point.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - This routine should be used whenever the entry is repushed
 - Use SLPSH3 to repush.

SLPSHI Push an entry into the selection list with dummy attention point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void SLPSHI ( Id )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Id;      /* ID of entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - The routine puts the specified entity into attention mode.

SLPSH3 Push an entry into the selection list.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void SLPSH3( Id, Ppoint )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
int    *Id;      /* The entity ID.
float  Ppoint[3]; /* Attention mark location.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

SLRJCT Remove an entry from the selection list.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void SLRJCT ( void )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

SLTOP3

Get the top entry from the selection list and return the XYZ coordinates of the picked point.

Syntax :

```
/*- - - - - */
void SLTOP3( Id, X, Y, Dfpt )
/*- - - - - */
/*
/* Output:
/*
/* The entity id.
/* Attention mark location ( DEVICE UNITS ).
/* Attention mark location ( DEVICE UNITS ).
/* XYZ coordinates of the picked point.
/*- - - - - */
float Dfpt[3]
```

Notes : - This routine should be used whenever the entry is repushed
 - Use SLPSH3 to repush.

UGETPNT

Get coordinates of a picked point.

Syntax :

```
/*- - - - - */
void UGETPNT ( System, Option, Point, Response, )
/*- - - - - */
/*
/* Input :
/*
/* 1 = Model,
/* 2 = Work coordinate system.
/*
/* Input/Output :
/*
/* Default method to a pick point.
/* Method selected to a pick point.
/*
/* Output :
/*
/* Coordinates of the point.
/* See Chapter 1, General Concepts, Mouse Response.
/*- - - - - */
double Point[3];
int *Response;
```

UMPICK

Pick entities and store their IDs in the array IDS.

Syntax :

```
/*- - - - - */
void UMPICK ( Maxid, Ids, Nid, Respon )
/*- - - - - */
/*
/* Input :
/*
/* Maximum number of IDs that may be stored in the
/* array IDS.
/*
/* Output :
/*
/* Array containing the ID numbers of the picked
/* entities.
/* Number of picked IDs.
/* See Chapter 1, General Concepts, Mouse Response.
/*- - - - - */
int Ids[Maxid];
int *Nid;
int *Respon;
```

UPICK Pick a single entity.
Syntax :

```

/*- - - - - */
int UPICK ( Respon )
/*- - - - - */
      /*
      /* Output :
      /*
int      *Respon;    /* See Chapter 1, General Concepts, Mouse Response.
      /*
      /* Returns :
      /* ID of picked entity.
/*- - - - - */

```

UPICKP Pick a curve, and create a point at the picked position.
Syntax :

```

/*- - - - - */
int UPICKP ( Idpnt, Respon )
/*- - - - - */
      /*
      /* Output :
      /*
int      *Idpnt;    /* ID of the newly created point.
int      *Respon;    /* See Chapter 1, General Concepts, Mouse Response.
      /*
      /* Returns :
      /* ID of picked entity.
/*- - - - - */

```



General Description

Plot entities.

CDK_PLOT_BOX

Immediate plot.

Syntax :

```

/*- - - - - */
int CDK_PLOT_BOX ( Opt, DevNumber, Scale, OutFileName, Box, Unify,
                  PlotFileName, WinSize )
/*- - - - - */
/*
/* Input :
/*
int    Opt;          /* Plot option :
/* = 1 : Automatic window,
/* = 2 : Displayed window,
/* = 3 : Box.
int    DevNumber;    /* The number (begin from 0) of the device which is
/* taken from "device.sdf" file (file must exist, see
/* in <root_cad >/dat directory). if DevNumber is equal
/* to the number of the devices which are defined in
/* "device.sdf" then a plot file is created (in the
/* <root_cad >/var/plot directory) via interaction.
double *Scale;        /* Plotting scale.
char   *OutFileName;  /* The name of a file which is sent to the device.
/* Use NULL for a default from name which is taken from
/* "device.sdf".
int     Box[4];       /* Defines a box (if Opt = 3).
int     Unify;        /* = 1 : Check for overlapping lines and plot only once.
/* = 0 : Do not check for overlapping lines.
char   *PlotFileName; /* The name of a plot file in case DevNumber >= then
/* the number of devices in "device.sdf".
/*
/* Output :
/*
double WinSize[2];    /* Size of plot window.
/* Returns :
/* See General Concepts-Status on page 1-2.
/* = 1 : No plotting input.
/* = 2 : Cannot access to plot file.
/*- - - - - */
- Note: All Device concepts are explained in the Utilities Manual,
        Environment Utilities -> DEVICE INSTALLATION (DINSTALL).

```

CDK_PLOT_DEVICE_LIST

Get the plot device list.

Syntax :

```

/*- - - - - */
int CDK_PLOT_DEVICE_LIST ( )
/*- - - - - */
- Note: All Device concepts are explained in the Utilities Manual,
        Environment Utilities -> DEVICE INSTALLATION (DINSTALL).

```

CDK_PLOT_GET_DEVICE_NAME

Get the name of the device.

Syntax :

```
/*- - - - - */
int CDK_PLOT_GET_DEVICE_NAME ( Index, Name )
/*- - - - - */
/*
/* Input :
/*
int      Index;    /* The Index number of the plotter device.
/*
/* Output :
/*
char      *Name;    /* Name of device.
/* Returns :
/* See General Concepts-Status on page 1-2.
/*- - - - - */
- Note: All Device concepts are explained in the Utilities Manual,
      Environment Utilities -> DEVICE INSTALLATION (DINSTALL).
```

CDK_PLOT_SET_COLORMAP Set color map.

Syntax :

```
/*- - - - - */
int CDK_PLOT_SET_COLORMAP ( Mode, ColorMap )
/*- - - - - */
/*
/* Input :
/*
int      Mode;    /* = 1 : COLOR PEN MAP off.
/* = 2 : COLOR PEN MAP on.
/* = 3 : WISIWIG.
int      ColorMap[15]; /* Color Map.
/* Returns :
/* See General Concepts-Status on page 1-2.
/*- - - - - */
- Note: All Plot parameters are explained in the Fundamentals and General
      Functions Manual - PLOT.
```

CDK_PLOT_SET_OFFSET Set the plotting offsets.

Syntax :

```
/*- - - - - */
int CDK_PLOT_SET_OFFSET ( XOffSet, YOffSet )
/*- - - - - */
/*
/* Input :
/*
double   XOffSet;  /* X distance from the 0,0 point from which the plotter
/* will begin.
double   YOffSet;  /* Y distance from the 0,0 point from which the plotter
/* will begin.
/* Returns :
/* See General Concepts-Status on page 1-2.
/*- - - - - */
- Note: All Plot parameters are explained in the Fundamentals and General
      Functions Manual - PLOT.
```

CDK_PLOT_SET_ROTATE Set plotting rotate flag.
Syntax :

```

/*- - - - - */
void CDK_PLOT_SET_ROTATE ( Rotate )
/*- - - - - */
/*
/* Input :
/*
int Rotate; /* = 0 : Do not rotate.
/* = 1 : Rotate 90 degrees counter clockwise .
/*- - - - - */
- Note: All Plot parameters are explained in the Fundamentals and General
Functions Manual - PLOT.

```

CDK_PLOT_SET_SPEED Set the plotting speed.
Syntax :

```

/*- - - - - */
int CDK_PLOT_SET_SPEED ( Speed )
/*- - - - - */
/*
/* Input :
/*
int Speed; /* Pen Speed (1 - 5).
/* Returns :
/* See General Concepts-Status on page 1-2.
/*- - - - - */
- Note: All Plot parameters are explained in the Fundamentals and General
Functions Manual - PLOT.

```

REPORTS

General Description

Routines in this section are for handling reports.

CDK_QE_REPORT_CREATE Create an electrode report.
Syntax :

```

/*- - - - - */
int cdk_qe_report_create( pszFileName, bStartViewer )
/*- - - - - */
P_CHAR*  PSZFILENAME; /* I/O : char[256], Report file name without ".erp" */
/* extension. If empty, the generated name is */
/* <component_file_name>.erp */
/*
T_INT  BSTARTVIEWER;      /* I : if TRUE - to launch report viewer */
/* Return:  0 = O.K. */
/*          -1 = Unknown ERROR, */
/*          -2 = ERROR: Cannot open report file */
/*          -3 = ERROR: No electrode model in pfm */
/*- - - - - */

```

STATUS

General Description

These routines are used to acquire various data on the current part file.

AXI_TYPE_OPEN

Get the type of the active part.

Syntax :

```

/*- - - - - */
void      AXI_TYPE_OPEN( typ_comp )
/*- - - - - */
/*
/* Output :
/*
int      *typ_comp;
/*      1 = main assembly
/*      2 = sub asm, opened from main asm.
/*      3 = part, opened from main asm.
/*      0 = others ( part, mold etc )
/*      -1 = errors
/*- - - - - */

```

CDK_EXTRACT_SUB_ASSY

Extract as EXTRACT >> EXTRACT SUB ASSEMBLY.

Syntax :

```

/*- - - - - */
int CDK_EXTRACT_SUB_ASSY ( UFileName, NumEnts, Entities, Orig, Scale, Flags )
/*- - - - - */
/*
/* Input :
/*
char      *UFileName;
/* Destination File name.
int      NumEnts;
/* Num of Entities.
int      *Entities;
/* Entities Buffer.
double   Orig[3][3];
/* Three points indicating rotation & shift
/* transformation :
/*      Orig[0] = Origin point.
/*      Orig[1] = X Direction.
/*      Orig[2] = Y Direction.
double   Scale;
/* Scaling transformation.
int      Flags[4];
/* Input flags :
/*      Flags[0] = level flag :
/*          On = export level map.
/*          Off = use active level.
/*      Flags[1] = Explode flag :
/*          On = explode all external.
/*      Flags[2] = Replace flag.
/*          On = Delete all entities from the "Entities
/*              Buffer" and Replace them with an
/*              instance of the extracted sub assembly.
/*      Flags[3] = Reference Assembly.
/*          On = An instance from the assembly file is
/*              placed in the sub assembly file.
/*      Flags[4] = Close Flag.
/*          On = Close the newly created work area.
/*          Off = Keep the newly created work area Open.
/* Note : if Reference Assembly is to be used, then
/*         Replace must be 1.
/* Returns :
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_PRG_INFO Return a string containing the version number, build number, etc.
Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void cdk_prg_info ( ver_str )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input/Output :
                                /*
                                /*
char *ver_str;                  /* A pointer to a 15 character string.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GTLANG Return the translated version.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GTLANG ( Language )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Output :
                                /*
                                /*
int      *Language;            /*      0 = Not defined.
                                /*      1 = English.
                                /*      2 = Japanese.
                                /*      3 = Other.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GTMACH Return the type of machine.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GTMACH ( Type )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Output :
                                /*
                                /*
int      *Type;                /* The type of machine.
                                /*      1 = CROMEMCO
                                /*      2 = IBM PC
                                /*      3 = APOLLO
                                /*      4 = IBM RT
                                /*      5 = VAX
                                /*      6 = SUN
                                /*      7 = HP
                                /*      8 = IBM RS6000
                                /*      9 = DECSTATION
                                /*     10 = Silicon Graphics IRIS
                                /*     11 = Windows NT
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

LASTID Get the last used ID number.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void LASTID ( Id )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int *Id; /* Last used ID number.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OB2DMD Obtain the work mode - 2D/3D.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OB2DMD ( Mode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int *Mode; /* 2 = 2D mode.
/* 3 = 3D mode.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OBACAT Obtain the active font, pen and color.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBACAT ( Font, Pen, Color )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int *Font; /* The current line font.
int *Pen; /* The current pen.
int *Color; /* The current color.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OBACME Get the part file measurement unit.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBACME ( Mesu )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int *Mesu; /* 1 = MM
/* 2 = CM
/* 3 = M
/* 4 = INCH
/* 5 = FEET
/* 0 = Unexpected value.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OBACRTGet the pathname of the Cimatron^{it} root directory.**Syntax :**

```

/*- - - - - */
void OBACRT ( Root, Rl )
/*- - - - - */
/*
/* Output :
/*
/*
char *Root; /* Pathname of root.
int *Rl; /* Length of pathname.
/*- - - - - */

```

U2DMOD

Check if system is in 2D mode.

Syntax :

```

/*- - - - - */
int U2DMOD ( void )
/*- - - - - */

```

UAXI_INDEX_DATA

Get the data of assembly tree components.

Syntax :

```

/*-----*/
void UAXI_INDEX_DATA ( Indx, Name, Type, Parent_indx, Status )
/*-----*/
/*
/* Output :
/*
/* 1 <= indx <= axi_num_asm_comps( )
char *Name; /* Name of component #indx
/* Length must be == MAXPNL
int *Type; /* Type of component:
/* 1 = part
/* 2 = sub-asm
/* 3 = library part
/* 4 = library sub-asm
int Parent_indx; /* Index of parent
int Status; /* - 0 = OK; <0 error
/*- - - - - */

```

UAXI_NUM_ASM_COMPS

Get the number of assembled components on all levels.

Syntax :

```

/*-----*/
void uaxi_num_asm_comps( status )
/*-----*/
/*
/* Output :
/*
/* - 0 = OK; ( -1 ) - error
int *status; /* Returns :
/* Number of assembled components
/*- - - - - */

```

UGTAPL

Get the current application number.

Syntax :

```

/*- - - - - */
void UGTAPL ( Ap )
/*- - - - - */
/*
/* Output :
/*
/*
int    *Ap;
/*    1 = Modeling application.
/*    2 = Drafting application.
/*    3 = NC application.
/*    4 = FEM application.
/*- - - - - */

```

UGTDTM

Get the current date and time.

Syntax :

```

/*- - - - - */
void UGTDTM ( Dttm )
/*- - - - - */
/*
/* Output :
/*
/*
int    Dttm[6];
/* Date and time in the format
/* Year/Month/Day/Hour/Minute/Second.
/*- - - - - */

```

UGTERF

Get the list of external references in the current PFM.

Syntax :

```

/*- - - - - */
void UGTERF ( Pnames, Mnames, Mtypes, Mpfmln, Mmstln, Mnum, Count, Status )
/*- - - - - */
/*
/* Output :
/*
/*
char    *Pnames[];
char    *Mnames[];
/* Array of PFM names of external references
/* Array of master's names of external
/* references ( zero length string when SubAssy )
/* (memory for elements of these two arrays have to be
/* allocated by the user )
int     Mtypes[];
/* Array of master's types of external references
/*
/* Input :
/*
int     Mpfmln;
int     Mmstln;
int     Mnum;
/* Max. length of PFM name
/* Max. length of master name
/* Max. number of external references in arrays
/*
/* Output :
/*
/*
int     Count;
int     Status;
/* Number of retrieved references
/* 0 = OK; <0 error
/* 2 = master table not found
/* 5 = mnum < real number of masters
/* 8 = real data exceeds mpfmln or mmstln
/*- - - - - */

```

UGTNID

Get the node ID.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void UGTNID ( Nodeid )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int    *Nodeid;    /* The node ID in integer format.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UGTPFN

Get the name of the current part file.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void UGTPFN ( Maxlen, Pfnlen, Pfname, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Maxlen;    /* Maximum length of buffer.
/*
/*
/* Output :
/*
/*
int    *Pfnlen;    /* Length of file name.
char    *Pfname;    /* File name ( excluding ".pfm" ). Maximum length of
/* PFNAME is 128 characters.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UGTPNM

Get full path name

Syntax :

```

/*- - - - - */
void UGTPNM ( Mode, Fname, Fnlen, Maxlen, Pname, Pnlen, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Mode;    /* 1 = File must exist.
/*              /* 2 = Could be a non-existent file.
char    *Fname;   /* File name.
int     *Fnlen;   /* Length of FNAME.
int     *Maxlen;  /* Maximum length for path name.
/*
/* Output :
/*
char    *Pname;   /* Full path name.
int     *Pnlen;   /* Length of PNAME.
int     *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UGTSNM

Get the name of a system file/directory.

Syntax :

```

/*- - - - - */
void UGTSNM ( Type, Maxl, Name, Lnth, Status )
/*- - - - - */
/*
/* Input :
/*
int     *Type;    /* The type of the requested name:
/*              /* 1 = <root_cad> directory.
/*              /* 2 = WORK directory ( "\workarea" ).
/*              /* 3 = DAT directory ( "\<root_cad>\dat\" ).
/*              /* 4 = USER directory ( "\<root_cad>\var\user\" ).
/*              /* 5 = USYS directory ( "\<root_cad>\var\usys\" ).
/*              /* 6 = Current directory.
/*              /* 7 = PROFILES directory
/*              /* ( "\<root_cad>\var\profiles\" ).
/*              /* 8 = PROFILES\<user> directory
/*              /* ( "\<root_cad>\var\profiles\<user>" ).
/*              /* 9 = PROFILES\<work_group> directory
/*              /* ( "\<root_cad>\var\profiles\<work_group>" ).
/*              /* 10 = FONTS directory ( "\<root_cad>\dat\fonts\" ).
/*              /* 11 = EUSYS directory ( "\<root_cad>\var\eusys\" ).
int     *Maxl;    /* Maximum length of buffer.
/*
/* Output :
/*
char    *Name;    /* Requested name.
int     *Lnth;    /* Actual length of the requested name.
int     *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UGTVNM

Get the view number of the active view.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UGTVNM ( Vn )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Output :
                                /*
                                /*
int    *Vn;                      /* View number of the active view.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UGVNM

Get view name and type.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UGVNM ( Vnum, Vtype, Vname, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
                                /*
int    *Vnum;                    /* View number.
                                /*
                                /* Output :
                                /*
                                /*
int    *Vtype;                  /* View type.
                                /*    1 = View
                                /*    2 = Drawing
                                /*    3 = Macsys
                                /*    4 = Femsys
                                /*
char    *Vname;                 /* View name.
int    *Status;                 /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

USAVE

Save the current file with a given name.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void USAVE ( Fname, Fnlen, Mode, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Part file name ( without PFM extension ).
/* > 0 : Length of FNAME.
/* = 0 : Current file name.
/* 1 = With backup.
/* 2 = Without backup.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
char *Fname;
int *Fnlen;
int *Mode;
int *Status;

```

VPDRWN

Set the ERASE flag = 0 for a given view port.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VPDRWN ( Vpid )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* The view port ID number.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int *Vpid;

```

VPERSD

Get the ERASE flag for a given view port.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int VPERSD ( Vpid )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* The view port id number.
/*
/* Returns :
/* ERASE flag:
/* 0 = The view port hasn't been erased.
/* 1 = The view port has been erased.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int *Vpid;

```

VPGSDX

Get/Set the display transformation data from the view port table.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VPGSDX ( Mode, Vpid, W, Pan, Zoom, Drot )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
int    *Mode;    /*      1 = Get.
/*      2 = Set.
/*
/* Input :
/*
int    *Vpid;    /* The view port number.
/*
/* Output :
/*
int    W[4];     /* The clipping window.
int    Pan[2];   /*
float  *Zoom;    /* The zoom factor.
float  Drot[9];  /* Display transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

VPGVPN

Get the current view port number.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VPGVPN ( Vpid )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int    *Vpid;    /* The current view port number.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

VPLDIS

Get a list of all displayed windows.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VPLDIS ( Vpcnt, Vplist )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int    *Vpcnt;   /* The number of displayed view ports.
int    Vplist[10]; /* The list of displayed view ports.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



STRING

General Description

These routines are used to perform various string operations.

CHCOMP

Compare two characters with the option of ignoring lower/upper case.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int CHCOMP ( Char1, Char2, Flag )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char    *Char1;                /* First character.
char    *Char2;                /* Second character.
int     *Flag;                 /* If .TRUE., ignore lower/upper case.
                                /*
                                /* Returns :
                                /* .TRUE.
                                /* The characters are equal (case is ignored if FLAG is
                                /* TRUE.)
                                /* .FALSE.
                                /* The characters are not equal.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CONVLO

Convert a string to lower case.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void CONVLO ( String, From, To )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input/Output :
                                /*
char    *String;               /* String to be converted.
                                /* Converted string.
                                /*
                                /* Input :
                                /*
int     *From;                 /* Starting location of string to be converted.
int     *To;                   /* Ending location of string to be converted.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - No action will be taken if FROM .GT. TO.

CONVUP

Convert a string to upper case.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void CONVUP ( String, From, To )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input/Output :
                                /*
char   *String;                /* String to be converted.
                                /* Converted string.
                                /*
                                /* Input :
                                /*
int     *From;                 /* Starting location of string to be converted.
int     *To;                   /* Ending location of string to be converted.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - No action will be taken if FROM .GT. TO.

LENACT

Get the actual length of a string (without trailing blanks).

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void LENACT ( String, Length )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char   *String;                /* Character string.
                                /*
                                /*
                                /*
int     *Length;               /* Actual length ( without trailing blanks ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

LENINT

Find the length of an integer number.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int LENINT ( Ival )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int     *Ival;                 /* The integer number.
                                /*
                                /* Returns :
                                /* The length of IVAL.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

STCOMP

Compare two strings with the option of ignoring lower/upper case. The values of FROM1 and TO1 determine how many characters from STRNG2 will be included in the comparison.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int STCOMP ( Strng1, From1, Tol, Strng2, From2, Flag )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                     /*
                                     /* Input :
                                     /*
char  *Strng1;                       /* First string.
int   *From1;                       /* Starting location in STRNG1.
int   *Tol;                         /* Ending location in STRNG1.
char  *Strng2;                       /* Second string.
int   *From2;                       /* Starting location in STRNG2.
int   *Flag;                        /* If .TRUE., ignore lower/upper case.
                                     /*
                                     /* Returns :
                                     /* .TRUE.
                                     /* The strings are equal ( case is ignored if FLAG is
                                     /* TRUE ).
                                     /* .FALSE.
                                     /* The strings are not equal.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - The result is also .FALSE. if:
 FROM1 .LT. 1
 FROM2 .LT. 1.
 FROM1 .GT. TO1





Chapter 4

Modeling Routines

Introduction

This chapter contains routines associated with modeling activities within Cimatron^{it}. Each routine appears under an appropriate subsection that logically corresponds to the interactive functions of the system.

The subsections contained in this chapter are as follows:

ANALYZE	Calculate the properties of a contour.
CIRCLE	Creation of arc/circle entities.
CONIC	Creation of conic entities.
CONTOUR	Creation of contours.
CORNER	Corner creation routines, (chamfer, corner, fillet).
EXPLODE	Explode masters.
GROUP	Master handling.
LINE	Line creation.
MOVE	Entity copying routines.
PLACE	Creation of instances of existing masters.
PLANAR FACE	Planar face operations.
POINT	Creation of point entities.
PROJECT	Creation of projected entities.
SPLINE	Spline creation.
SURFACE	Surface creation.
SWEEP	Sweep routines.
TRANSFORMATION	Transformation routines.
TRIM	Trim routines.
UCS	UCS associated routines.
VERIFY	Entity verification routines.
WORKPL	Work plane routines.

ANALYZE

General Description

The following routines are used to calculate certain properties of a contour, such as area, center of gravity, and moment of inertia about various axes.

Carry out the following to obtain the various properties:

How To:

1. Call GFSPOC to initialize section properties and define the outer contour.
2. Call GFSPIS to define islands within the contour (if any).
3. Call GFSPUT to obtain the results of the desired calculations.
New islands may be added at this stage by calling GFSPOC, and another call to GFSPUT will produce the new results required.
4. To define a new outer contour, repeat step 1 to clear all previous data.

GFSPIS

Add an island to a closed contour.

Syntax :

```
/*- - - - - */
void GFSPIS ( N, Ids, Status )
/*- - - - - */
/*
/* Input :
/*
int *N; /* Number of curves in the contour of the island.
int Ids[N]; /* Vector containing N IDs that define the island.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

GFSPOC

Initialize contour properties and define an outer contour. The contour must be “well-defined” i.e. the curves defining the contour must meet at their endpoints, and no overlapping of lines is permitted.

Syntax :

```
/*- - - - - */
void GFSPOC ( N, Ids, Status )
/*- - - - - */
/*
/* Input :
/*
int *N; /* Number of curves in the outer contour.
int Ids[N]; /* Vector containing N IDs defining the outer contour.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

GFSPUT

Calculate and output various properties of the contour.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GFSPUT ( Units, Dc, Area, Cg, Ixx, Iyy, Ixy, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Units;    /* The unit of measure to be used for the output:
/*      1 = MM
/*      2 = CM
/*      3 = INCH
/*      4 = FEET
/*      5 = METER
int    *Dc;        /* 1 = Display center of gravity.
/*      0 = Do not display center of gravity.
/*
/*
float  *Area;       /* Area, in the selected units.
/*
/*
float  Cg[2];       /* Center of gravity, in the system's units.
/*
/* Output :
/*
float  *Ixx;        /* Moment of inertia with respect to the X axis.
float  *Iyy;        /* Moment of inertia with respect to the Y axis.
float  *Ixy;        /* Moment of inertia with respect to the XY axis.
int    *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - In this subroutine, the properties are calculated for the contour project
 ed onto the work plane.



CIRCLE

AR2CRV

Create an arc/circle entity on the work plane, tangent to 2 curves.
The entity may also trim the curves to produce a fillet.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int AR2CRV ( Id1, Id2, Radius, Full, Trim, Select, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Id1;
int *Id2;
float *Radius;
int *Full;
int *Trim;
char *Select;
int *Status;
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of created entity.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

AR3PTD

To create an arc/circle defined by 3 points.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int AR3PTD ( Full, Idstart, Idmiddle, Idend, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Full;
int *Idstart;
int *Idmiddle;
int *Idend;
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

AR3PTS

Create an arc/circle entity on the work plane, defined by 3 points.

Syntax :

```

/*- - - - - */
int AR3PTS ( Full, Id1, Id2, Id3, Status )
/*- - - - - */
/*
/* Input :
/*
int *Full; /* 0 = Creates an arc.
/* 1 = Creates a circle.
int *Id1; /*
int *Id2; /*
int *Id3; /* ID numbers of the 3 points. If an arc is defined,
/* it will start at ID1 and end at ID3, passing through
/* ID2.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of created entity.
/*- - - - - */

```

ARC NRACreate an arc/circle entity on the work plane, defined by the ID of its center and its radius **RADIUS**. If an arc is defined, **ANGLE** is its angle relative to the X axis in degrees, and **DELTAN** is the angle relative to **ANGLE**.**Syntax :**

```

/*- - - - - */
int ARC NRA ( Full, Id, Radius, Angle, Deltan, Status )
/*- - - - - */
/*
/* Input :
/*
int *Full; /* 0 = Creates an arc.
/* 1 = Creates a circle.
int *Id; /* ID value of the center of the arc/circle.
float *Radius; /* Radius of the arc/circle.
float *Angle; /* Starting angle of the arc/circle relative to the X
/* axis in degrees.
float *Deltan; /* Angle between the sides of the arc. ( In the case of
/* a circle, DELTAN=360. )
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of created entity.
/*- - - - - */

```

ARCNRAD

Create an arc/circle on the work plane, defined by it's center point and radius.

Syntax :

```

/*- - - - - */
int ARCNRAD ( Full, Idpoint, Radius, Start, Delta, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Full;      /*    0 = Create an arc.
/*    1 = Create a circle.
int    *Idpoint;   /* The ID of the center point.
double *Radius;    /* The radius.
double *Start;     /* Start angle ( DEGREE ).
double *Delta;     /* Delta angle ( DEGREE ).
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

AREX2C

Create an arc/circle entity on the work plane, tangent to 2 curves or their extensions. It may also trim the curves to produce a fillet.

Syntax :

```

/*- - - - - */
int AREX2C ( Id1, Id2, Radius, Full, Trim, Select, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Id1;
int    *Id2;      /* ID numbers of the 2 curves.
float  *Radius;    /* Radius of the fillet.
int    *Full;      /*    0 = Creates an arc.
/*    1 = Creates a circle.
int    *Trim;      /*    0 = Does not trim curves.
/*    1 = Trims curves.
char   *Select;    /* See Chapter 1, General Concepts, Select.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of created entity.
/*- - - - - */

```

BEST_CRC

Find the “best” circle that can be created by a given points sequence.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void BEST_CRC( np, points, centre, radius, ierflg )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    np;          /* Number of points.
double points[];    /* 2-D points
/*
/* Output :
/*
/* The distance :
double centre[2];   /* Centre x, y
double *radius;     /* Radius of the best circle.
int    *ierflg;     /* 0 = OK.
/* -1 = all points on one line / point
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - This circle must pass through the first and last point !!!

CDK_ARC_3CRV

Create arc/circle tangent to three curves.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_ARC_3CRV ( Full, Curve1, Parametr1, Curve2, Parametr2, Curve3,
                  Parametr3, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Full;       /* 1 = Create an arc.
/* 2 = Create a circle.
int    *Curve1;     /* ID of the first curve.
double *Parametr1;   /* The reference parameter on the first curve.
int    *Curve2;     /* ID of the second curve.
double *Parametr2;   /* The reference parameter on the second curve.
int    *Curve3;     /* ID of the third curve.
double *Parametr3;   /* The reference parameter on the third curve.
/*
/* Output :
/*
int    *Status;     /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_ARC_2PNT

Create arc/circle by two points and radius.

Syntax :

```

/*- - - - - */
int CDK_ARC_2PNT ( Full, Startpoint, Endpoint, Sidepoint, Centerside, Radius,
                  Depth, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Full;      /*      1 = Create an arc.
/*      2 = Create a circle.
double Startpoint[2]; /* The coordinates of Start point ( WORK ).
double Endpoint[2];  /* The coordinates of Middle point ( WORK ).
double Sidepoint[2]; /* The coordinates of End point ( WORK ).
int    *Centerside; /*      1 = Center is lying on the Side.
/*      2 = Center is opposite the Side.
double *Radius;     /* The given radius of the arc.
double *Depth;       /* The given depth of the arc in current work plane.
/*
/* Output :
/*
int    *Status;     /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

CDK_ARC_3PNT

To create an arc/circle defined by coordinates of 3 points.

Syntax :

```

/*- - - - - */
int CDK_ARC_3PNT ( Full, Pointstart, Pointmiddle, Pointend, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Full;      /*      1 = Create an arc.
/*      2 = Create a circle.
double Pointstart[3]; /* The coordinates of Start point ( MODEL ).
double Pointmiddle[3]; /* The coordinates of Middle point ( MODEL ).
double Pointend[3];   /* The coordinates of End point ( MODEL ).
/*
/* Output :
/*
int    *Status;     /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

CDK_ARC_CENT_RAD To create a arc/circle on the work plane, defined by it's center point and radius.

Syntax :

```

/*- - - - - */
int CDK_ARC_CENT_RAD ( Full, Center, Radius, Start, Delta, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Full;      /*    0 = Create an arc.
/*    1 = Create a circle.
double Center[3];  /* The coordinates of the center point ( WORK ).
double *Radius;    /* The radius.
double *Start;     /* Start angle ( DEGREE ).
double *Delta;     /* Delta angle ( DEGREE ).
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

CDK_ARC_PNT2CRV Create all circles tangent to two given curves and point.

Syntax :

```

/*- - - - - */
void CDK_ARC_PNT2CRV ( Point, Curve1, Parametr1, Curve2, Parametr2Circles,
                      Numcircles, Status )
/*- - - - - */
/*
/* Input :
/*
double Point[3];   /* The coordinates of the given point ( MODEL ).
int    *Curve1;    /* ID of the first curve.
double *Parametr1; /* The refernce parameter on the first curve.
int    *Curve2;    /* ID of the second curve.
double *Parametr2; /* The refernce parameter on the second curve.
/*
/* Output :
/*
int    Circles[8]; /* The IDs of created circles.
int    *Numcircles; /* The number created circles.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```



CONIC

General Description

The following routines are used for operations on conics.

CDK_CONIC_3PNT

Create conic curve by 3 points.

Syntax :

```

/*- - - - - */
int CDK_CONIC_3PNT ( Startpoint, Endpoint, Slpinterpnt, Coefpq, Status )
/*- - - - - */
/*
/* Input :
/*
double Startpoint[3]; /* The coordinates of start point ( MODEL ).
double Endpoint[3]; /* The coordinates of end point ( MODEL ).
double Slpinterpnt[3]; /* The coordinates of point which intersects start and
/* end slopes ( MODEL ).
double *Coefpq; /* The shape parameter of conic section:
/* For Ellipse:
/* 0.0 < Coefpq < 0.5,
/* For Parabola:
/* Coefpq = 0.5,
/* For Hyperbole:
/* 0.5 < Coefpq < 1.0.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

CDK_CONIC_WP

Create conic curve parallel to Work Plane.

Syntax :

```

/*- - - - - */
int CDK_CONIC_WP ( Startpoint, Startslope, Endpoint, Endslope, Coefpq, Depth,
                  Status )
/*- - - - - */
/*
/* Input :
/*
double Startpoint[2]; /* The coordinates of Start point ( WORK ).
double Startslope[2]; /* The tangented vector on Start point ( WORK ).
double Endpoint[2]; /* The coordinates of End point ( WORK ).
double Endslope[2]; /* The tangented vector on End point ( WORK ).
double *Coefpq; /* The shape parameter of conic section:
/* For Ellipse:
/* 0.0 < Coefpq < 0.5,
/* For Parabola:
/* Coefpq = 0.5,
/* For Hyperbole:
/* 0.5 < Coefpq < 1.0.
double *Depth; /* The depth to create conic curve.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

CDK_ELLIPSE

Create an ellipse entity parallel to the work plane.

Syntax :

```

/*- - - - - */
int CDK_ELLIPSE ( Center, Radmajor, Radminor, Startangle, Deltaangle, Rotangle,
                  Status )
/*- - - - - */
/*
/* Input :
/*
double Center[3]; /* The coordinates of the center point ( WORK ).
double *Radjmajor; /* Major radius.
double *Radminor; /* Minor radius.
double *Startangle; /* Start angle ( DEGREE ).
double *Deltaangle; /* Delta angle ( DEGREE ).
double *Rotangle; /* Rotation angle of the ellipse on the active work
/* plane ( DEGREE ).
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

PARABO

Create an entity that is a parabola, defined by its two endpoints, ID1 and ID2, and the slopes at these points, SL1 and SL2.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int PARABO ( Id1, Id2, Sl1, Sl2, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Id1; /* IDs of the two endpoints.
int *Id2; /*
float Sl1[3]; /* The slopes at ID1 and ID2.
float Sl2[3]; /*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UCRELP

Create an ellipse on the work plane, defined by its center ID, and its major & minor radii A and B. If an arc is defined, ANGLE gives its angle (relative to the X work axis), and the angle subtended by the arc DELTAN. ROT defines rotation on active work plane.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int UCRELP ( Full, Id, A, B, Angle, Deltan, Rot, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Full; /* 0 = An arc is defined.
/* 1 = A full ellipse is defined.
int *Id; /* ID value of the center.
float *A; /* Half of major axis.
float *B; /* Half of minor axis.
float *Angle; /* Start angle ( if FULL = 0 ).
float *Deltan; /* Angle subtended by the arc ( if FULL = 0 ).
float *Rot; /* Rotation angle of the ellipse on the active work
/* plane.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```



CONTOUR

CDK_BALCON

Get all contours which contain a given entity.

Syntax :

```

/*- - - - - */
void CDK_BALCON ( Tol, Id, Idsbuf, Maxids, Numids, Conbuf, Maxcon, Numcon,
                  Status )
/*- - - - - */
/*
/* Input :
/*
double *Tol; /* The tolerance.
int *Id; /* The ID number of the given entity.
/*
/* Output :
/*
int Idsbuf[Maxids]; /* The buffer which holds the ID entities of all
/* contours that were found.
/*
/* Input :
/*
int *Maxids; /* The maximum number of IDs.
/*
/* Output :
/*
int *Numids; /* The number of entities in IDSBUF.
int Conbuf[Maxcon]; /* The buffer which holds the number of entities per
/* contour.
/*
/* Input :
/*
int *Maxcon; /* The maximum number of contours.
/*
/* Output :
/*
int *Numcon; /* The number of contours which were found.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_COMCRV_GET

Obtain the components of the component curve.

Syntax :

```

/*- - - - - */
void CDK_COMCRV_GET ( Comcurve, Num, Components, Status )
/*- - - - - */
/*
/* Input :
/*
int *Comcurve; /* The ID of the composite curve.
/*
/* Input/Output :
/*
int *Num; /* The expected number of components.
/* The real number of components.
/*
/* Output :
/*
int Components[Num]; /* The array of IDs of components curves.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK CONTOUR MULTY Create a 2D multi-contour buffer from a list of 2D curves.

Syntax :

```

/*- - - - - */
void CDK_CONTOUR_MULTY( NumContours, ContSizes, Curves, ContourBuff, Status )
/*- - - - - */
/*
/* Input:
/*
int *NumContours; /* Number of closed contours
/*
int *ContSizes; /* The array of numbers of curves in each contour.
/*
int *Curves; /* The array of the curve IDs in the contour.
/*
/* Output:
/*
int *ContourBuff; /* The resulting contour buffer.
/* ContourBuff[ TotalNumCurves * 7 + NumContours + 7 ]
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

```
Note: ContSizes and Curves must be defined according to the contour.
if NumContours = N :
    ContSizes = S1, S2, ... , SN and
    Curves     = Id1_Contour1, Id2_Contour1 , ... , IdS1_Contour1,
                Id1_Contour2, Id2_Contour2 , ... , IdS2_Contour2,
                .....
                Id1_ContourN, Id2_ContourN , ... , IdSN_ContourN
```

UCOMCR

Given a contour buffer, create a composite curve.

Syntax :

```

/*- - - - - */
int UCOMCR ( Bufcon, Dim, Mode, Status ) - - - - - */
/*- - - - - */
/* Input : */
/* */
/* Contour buffer. */
/* Curve dimension: */
/*     2 = 2D */
/*     3 = 3D */
int *Mode; /* 1 = Delete original */
/* 2 = Keep original */
/* */
/* Output : */
/* */
int *Status; /* See General Concepts-Status on page 1-2. */
/* */
/* Returns : */
/* ID number of the created composite curve */
/* ( if STATUS = 0 ). */
/*- - - - - */

```

UCOMSP

Approximate a contour by a Bezier spline.

Syntax :

```

/*- - - - - */
int UCOMSP ( Tol, Bufcon, Dim, Mode, Status )
/*- - - - - */
/*
/* Input :
/*
/*
double *Tol;      /* Tolerance for approximation.
int *Bufcon;      /* Contour buffer.
int *Dim;         /* Spline dimension:
/*      2 = 2D
/*      3 = 3D
int *Mode;        /*      1 = Delete original
/*      2 = Keep original
/*
/* Output :
/*
int *Status;      /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID number of the created spline ( if STATUS = 0 ).
/*- - - - - */

```

UDFDC

Create a CLOSED contour buffer interactively.

Syntax :

```

/*- - - - - */
void UDFDC ( Defmod, Islmod, Size, Buf, Status )
/*- - - - - */
/*
/* Input :
/*
int *Defmod;      /* Definition mode of the contour.
/*      1 : Curves must lie on a plane which will be
/*            determined by the first curve or by the first
/*            two curves ( in case the first curve is a
/*            line ).
/*      2 : Curves must lie on the WORK plane or parallel
/*            to it. distance to the WORK plane will be
/*            determined by the end point of the first
/*            curve.
/*      3 : Curves must lie on the WORK plane.
/*      4 : Curves can be any 2D or 3D curves:
/*            projection on the WORK plane will be carried.
/*      5 : Curves can be any 2D or 3D curves
/*            ( In that case 3D contour can be defined ).
int *Islmod;      /*      0 = Unable islands definition.
/*      1 = Enable islands definition.
int *Size;        /* Buffer size.
/*
/* Output :
/*
int Buf[Size];    /* Contour buffer ( if STATUS = 0 ).
int *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - Array BUF must be sufficient to hold (5 + 7 * TOTAL + NG + 1) integers,
 where : TOTAL = number of curves in the contourNG = number of islands +1.

UDFDO

Create an OPEN contour buffer interactively.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UDFDO ( Defmod, Size, Buf, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Defmod;    /* Definition mode of the contour.
/*      1 : Curves must lie on a plane which will be
/*            determined by the first curve or by the
/*            first two curves ( in case the first curve
/*            is a line ).
/*      2 : Curves must lie on the WORK plane or
/*            parallel to it. distance to the WORK plane
/*            will be determined by the end point of the
/*            first curve.
/*      3 : Curves must lie on the WORK plane.
/*      4 : Curves can be any 2D or 3D curves:
/*            projection on the WORK plane will be carried.
/*      5 : Curves can be any 2D or 3D curves
/*            ( In that case 3D contour can be defined ).
int    *Size;      /* Buffer size.
/*
/* Output :
/*
int    Buf[Size];  /* Contour buffer ( if STATUS = 0 ).
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Array BUF must be sufficient to hold (19 + 7 * NC) integers,
 where:NC=number of curves in the contour.

USMKCB

Create a contour buffer from a list of curves. The contour must be “well-defined” i.e. the curves defining the contour must meet at their endpoints and no overlapping of lines is permitted.

Syntax :

```

/*- - - - - */
void USMKCB ( Cbmode, Nc, Crvlst, Oc, Cbuf, Status )
/*- - - - - */
/*
/* Input :
/*
int      *Cbmode;
/* The definition mode of the contour:
/*   1 : The definition curves must lie on a plane.
/*       This plane will be determined by the first
/*       curve or by the first two curves ( in case
/*       the first curve is a line ).
/*   2 : The definition curves must lie on the WORK
/*       plane or on a parallel plane. This plane will
/*       be determined by the end points of the first
/*       curve of the contour.
/*   3 : The definition curves must lie on the WORK
/*       plane.
/*   5 : The definition curves can be any 2D or 3D
/*       curves, ( in this case the 3D contour can be
/*       defined ).
int      *Nc;
/* Number of curves
int      Crvlst[Nc];
/* Vector containing NC IDs defining the outer contour.
int      *Oc;
/*   0 = Contour is open.
/*   1 = Contour is closed.
/*
/* Output :
/*
int      *Cbuf;
/* The contour buffer.
int      *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - Array CBUF must be sufficient to hold (7 * (NC + 3 - 2 * OC)) integer
 S.



CORNER

GFCHMF

Create a chamfer as it is done by the function CORNER in the interactive system.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int GFCHMF ( Idlin1, Idp1, Idlin2, Idp2, Len, Ang, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input/Output :
/*
int    *Idlin1;    /* ID of 1st line.
/* ID of duplicate of 1st line ( when working in a
/* VIEW ). See Chapter 1, General Concepts, Duplication
/* of Entities.
/*
/* Input :
/*
int    *Idp1;    /* ID of point on 1st line.
/*
/* Input/Output :
/*
int    *Idlin2;    /* ID of 2nd line.
/* ID of duplicate of 2nd line ( when working in a
/* VIEW ). See Chapter 1, General Concepts, Duplication
/* of Entities.
/*
/* Input :
/*
int    *Idp2;    /* ID of point on 2nd line.
float  *Len;    /* Length of chamfer.
float  *Ang;    /* Angle between the chamfer and the 1st line.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/*
/* ID of created chamfer.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GFCORN

Trim curves at their intersection point as it is done by the function **CORNER** in the interactive system.

Syntax :

```

/*- - - - - */
void GFCORN ( Idcrv1, Idp1, Idcrv2, Idp2, Status )
/*- - - - - */
/*
/* Input/Output :
/*
int    *Idcrv1;
/* ID of 1st curve.
/* ID of duplicate IDCrv1 ( when working with VIEW ).
/* See Chapter 1, General Concepts, Duplication of
/* Entities.
/*
/* Input :
/*
int    *Idp1;
/* ID of point on IDCrv1.
/*
/* Input/Output :
/*
int    *Idcrv2;
/* ID of 2nd curve.
/* ID of duplicate IDCrv2 ( when working with VIEW ).
/* See Chapter 1, General Concepts, Duplication of
/* Entities.
/*
/* Input :
/*
int    *Idp2;
/* ID of point on IDCrv2.
/*
/* Output :
/*
int    *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

GFFILT

Create a fillet corner.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int GFFILT ( Idcrv1, Idp1, Idcrv2, Idp2, Tr, Ac, Rad, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input/Output :
/*
int *Idcrv1; /* ID of 1st curve.
/* ID of duplicate of 1st curve
/* (when working with VIEW and TR=2, i.e., trim mode is
/* "on"). See Chapter 1, General Concepts, Duplication
/* of Entities.
/*
/* Input :
int *Idp1; /* ID of point on 1st curve.
/*
/* Input/Output :
int *Idcrv2; /* ID of 2nd curve.
/* ID of duplicate of 2nd curve
/* (when working with VIEW and TR=2, i.e., trim mode is
/* "on"). See Chapter 1, General Concepts, Duplication
/* of Entities.
/*
/* Input :
int *Idp2; /* ID of point on 2nd curve.
int *Tr; /* Trim mode:
/* 1 = off
/* 2 = on
int *Ac; /* 1 = arc
/* 2 = circle
float *Rad; /* Radius of fillet corner.
/*
/* Output :
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of created fillet.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



EXPLODE

GREXPL

Explode an instance.

Syntax :

```

/*- - - - - */
void GREXPL ( Idinst, Level, Max, N, Ids, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idinst; /* ID of instance to be exploded.
int *Level; /* 1 = source level is assigned to each entity.
/* 2 = active level is assigned to each entity.
int *Max; /* Maximum number of IDs to be extracted from IDINST
/* after explode.
/*
/* Output :
/*
int *N; /* Number of IDs extracted from IDINST.
int Ids[Max]; /* Vector of N IDs ( maximum number of MAX ).
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UEXPLD

Explode an instance by levels.

Syntax :

```

/*- - - - - */
void UEXPLD ( Idinst, Nstopt, Level, Max, N, Ids, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idinst; /* ID of instance to be exploded.
int *Nstopt; /* 1 = one nesting level.
/* 2 = all nesting levels.
int *Level; /* 1 = source level is assigned to each entity.
/* 2 = active level is assigned to each entity.
int *Max; /* Maximum number of IDs to be extracted from IDINST
/* after explode ( all the IDs will be exploded ).
/*
/* Output :
/*
int *N; /* Number of IDs extracted from IDINST.
int Ids[Max]; /* Vector of N IDs ( maximum number of MAX ).
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```



GROUP

General Description

The following routines handle “masters”.

There are two methods to create a master, i.e., a group of entities. The first method requires as input an array containing the IDs of the entities to be grouped. The routine **GRCLCR** should be used for this method.

The second method requires that the programmer indicate where to start collecting entities, and where to stop collecting. All the IDs of the entities created between these two calls will form the master. To use this method, call the routine **GRCOPN** to open the master and **GRCLOS** to close it.

Two routines in this section, **GRIDEM** and **GRIDIM** return the ID number of a specified master when its name is input.

GRIDES and **GRIDSA** may be used to find the ID of an external sub-view or a sub-assembly, respectively. Use **GRDLUN** to delete an unnamed master.

CDK_UMDDEL Delete a modal group.

Syntax:

```
/*- - - - - */
void cdk_umddel ( mnami, status )
/*- - - - - */
/*
/* Input :
/*
/*
char *mnami; /* Group name
int *status; /*
/*- - - - - */
```

GRCLCR Create a master composed of entities, the IDs of which are given in the array IDS.

Syntax :

```
/*- - - - - */
void GRCLCR ( Namlen, Name, Num, Ids, Id1, Id2, Id3, Idgrou, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int *Namlen; /* Length of master NAME, ( 0=unnamed master ).
char *Name; /* Name of the master to be created.
int *Num; /* Number of entities in IDS.
int Ids[Num]; /* Array of IDs of entities.
int *Id1; /*
int *Id2; /*
int *Id3; /* IDs of 3 points defining the local coordinate system
/* for the master.
/*
/* Output :
/*
/*
int *Idgrou; /* ID of the created master.
int *Status; /* See General Concepts-Status on page 1-2.Special
/* Cases:
/*
/* 1 = A previously opened master was not closed.
/* -11 = NAMLEN too long ( > 20 characters ).
/* -12 = NAME already exists in table.
/*
/*- - - - - */
```

GRCLOS

Stop designating entities which will be grouped to form a master in the current session of the interactive **USER** function.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GRCLOS ( Idgrou, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Idgrou;    /* ID of the master created in the call to GRCOPN.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Special Cases:
/*      1  = There was no previous call to GRCOPN;
/*           all entities are included.
/*      -11 = IDGROU is not the ID of the master created.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GRCOPN

Start designating entities which will be grouped to form a master in the current session of the interactive **USER** function.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GRCOPN ( Namlen, Name, Id1, Id2, Id3, Idgrou, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Namlen;    /* Length of the master name, ( 0 = unnamed master ).
char    *Name;     /* Name of the master to be created.
int    *Id1;       /*
int    *Id2;       /*
int    *Id3;       /* IDs of 3 points defining the local coordinate system
/*           for the master.
/*
/* Output :
/*
int    *Idgrou;    /* ID of the created master.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Special Cases:
/*      1  = A previously opened master was not closed.
/*      -11 = NAMLEN too long ( > 20 characters ).
/*      -12 = NAME already exists in table.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GRDLUN

Delete an unnamed master.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GRDLUN ( Idm, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int    *Idm;                    /* ID of the master to be deleted.
                                /*
                                /* Output :
                                /*
int    *Status;                /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GRIDEM

Find the ID of an external master.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int GRIDEM ( Pname, Pnlen, Ename, Enlen, Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char    *Pname;                /* Part file name without the .pfm suffix.
int    *Pnlen;                 /* Length of PNAME in characters excluding the .pfm
                                /* suffix.
char    *Ename;                /* Name of the master entity.
int    *Enlen;                 /* Length of ENAME in characters.
                                /*
                                /* Output :
                                /*
int    *Id;                    /* ID of the master entity.
int    *Status;                /* See General Concepts-Status on page 1-2.
                                /*
                                /* Returns :
                                /* .TRUE.
                                /* Master found.
                                /* .FALSE.
                                /* Master not found.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GRIDES

Find the ID of an external sub-view.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int GRIDES ( Pname, Pnlen, Ename, Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
char *Pname; /* Part file name without the .pfm suffix.
int *Pnlen; /* Length of PNAME in characters excluding the .pfm
/* suffix.
char *Ename; /* Name of the external sub-view ( maximum length of 6
/* characters ).
/*
/*
/* Output :
/*
/*
int *Id; /* ID of the external sub-view.
int *Status; /* See General Concepts-Status on page 1-2.
/*
/*
/* Returns :
/* .TRUE.
/* Master found.
/* .FALSE.
/* Master not found.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GRIDIM

Find the ID of an internal master.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int GRIDIM ( Ename, Enlen, Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
char *Ename; /* Name of the master entity.
int *Enlen; /* Length of ENAME in characters.
/*
/*
/* Output :
/*
/*
int *Id; /* ID of the master entity.
int *Status; /* See General Concepts-Status on page 1-2.
/*
/*
/* Returns :
/* .TRUE.
/* Master found.
/* .FALSE.
/* Master not found.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GRIDIS

Find the ID of an internal sub-view.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int GRIDIS ( Ename, Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Ename; /* Name of internal sub-view entity ( maximum length of
/* 6 characters ).
/*
/* Output :
/*
int *Id; /* ID of the sub-view.
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* .TRUE.
/* Master found.
/* .FALSE.
/* Master not found.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GRIDSA

Find the ID of a sub-assembly.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int GRIDSA ( Pname, Pnlen, Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *Pname; /* Part file name without the .pfm suffix.
int *Pnlen; /* Length of PNAME in characters excluding the .pfm
/* suffix.
/*
/* Output :
/*
int *Id; /* ID of the external sub-view.
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* .TRUE.
/* Master found.
/* .FALSE.
/* Master not found.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UMADEL

Delete a named master.

Syntax :

```

/*- - - - - */
void UMADeL ( Id, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Id;      /* ID of the named master to be deleted.
/*
/* Output :
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

WSNMX1

Export the master from the current PFM file to another file by its full path name.

Syntax :

```

/*- - - - - */
void WSNMX1( Mode_Replace, Master_Name_Cur, Full_Path_File,
             Master_Name_New, Status )
/*- - - - - */
/*
/* Input:
/*
int    *Mode_Replace; /* 1 = If external file has a master with the same
/*                      name replace it.
/*                      2 = Do not replace.
char    *Master_Name_Cur; /* The name of the master in the current PFM file
char    *Full_Path_File; /* Full path name of the target file.
char    *Master_Name_New; /* The name of the new master in the target file.
/*
/* Output:
/*
int    *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - Destination file must exist.



LINE

CDK_LINE_OFFSET

To create an offset line.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_LINE_OFFSET ( Baseline, Offset, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int      *Baseline;           /* ID of base line.
double  *Offset;             /* Offset value: 0 - Right, - Left.
                                /*
                                /* Output :
int      *Status;            /* See General Concepts-Status on page 1-2.
                                /*
                                /* Returns :
                                /* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

LN2ENT

Create a line entity defined by two point entities.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int LN2ENT ( Id1, Id2, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int      *Id1;               /*
int      *Id2;               /* IDs of the 2 points.
                                /*
                                /* Output :
int      *Status;            /* See General Concepts-Status on page 1-2.
                                /*
                                /* Returns :
                                /* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

LN2ENTD

To create a line defined by two points.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int LN2ENTD ( Startpointid, Endpointid, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Startpointid; /* The ID of the start point.
int    *Endpointid;   /* The ID of the end point.
/*
/* Output :
/*
int    *Status;        /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

LN2PNT

Create a line entity defined by two points.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int LN2PNT ( Sys, Point1, Point2, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Sys;           /* Coordinate system in which POINT1 and POINT2 are
/* given:1 = Model.2 = Work.
float   Point1[3];     /* X, Y, Z coordinates of the starting point.
float   Point2[3];     /* X, Y, Z coordinates of the endpoint.
/*
/* Output :
/*
int    *Status;        /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

LN2PNTD

Create a line defined by two points.

Syntax :

```

/*- - - - - */
int LN2PNTD ( System, Startpoint, Endpoint, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int *System; /* The coordinate system: 1 - MODEL, 2 - WORK.
double Startpoint[3]; /* The coordinates of the start point.
double Endpoint[3]; /* The coordinates of the end point.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

LNCVCV

Create a line using the 2 CURVES option, as in the interactive system.

Syntax :

```

/*- - - - - */
int LNCVCV ( Idcrv1, Idp1, Idcrv2, Idp2, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int *Idcrv1; /* ID of 1st curve.
int *Idp1; /* ID of point on 1st curve.
int *Idcrv2; /* ID of 2nd curve.
int *Idp2; /* ID of point on 2nd curve.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created line.
/*- - - - - */

```

LNIPVE

Create a line entity defined by the ID of a point, and a vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int LNIPVE ( Sys, Idp, Vector, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Sys; /* Coordinate system in which VECTOR is defined:
/* 1 = Model.
/* 2 = Work.
int *Idp; /* ID of start point.
float Vector[3]; /* The line will be created from the specified start
/* point, IDP, with the length and direction of the
/* specified vector.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created line.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

LNIPVED

Create a line defined by a point and a vector.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int LNIPVED ( System, Pointid, Vector, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *System; /* The coordinate System:
/* 1 - MODEL,
/* 2 - WORK.
int *Pointid; /* The ID of the start point.
double Vector[3]; /* The vector between start and end points.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

LNLINC

Create a line using LN CURVE option as in the interactive system.

Syntax :

```

/*- - - - - */
int LNLINC ( Idline, Idplin, Idcrv, Idpcrv, Length, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of line.
/* ID of a point on IDLINE.
/* ID of curve.
/* ID of point on IDCRV.
/* Length of line to be created.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created line.
/*
/*- - - - - */

```

LNOFFS

Create an offset line.

Syntax :

```

/*- - - - - */
int LNOFFS ( Idline, Side, Offset, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of base line.
/* Offset side:
/* 1 = left of the original entity
/* -1 = right of the original entity
/* Offset distance.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created line.
/*
/*- - - - - */

```

Note : - The direction of the line is determined when the line is created, i.e. from the starting point to the last endpoint. The direction of the offset is understood to be to the right or left when facing in the direction of the line.

LNPOEN

Create a sequence of lines as an open/closed polygon.

Syntax :

```

/*- - - - - */
void LNPOEN ( Count, Ids, Outar, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int    *Count;    /* Number of vertices in the polygon.
int    Ids[Count]; /* Array containing COUNT IDs of entities of type point.
/*
/* Output :
/*
/*
int    Outar[Count-1]; /* Array containing the IDs of the lines created.
int    *Status;        /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - The length of OUTAR should be equal to or greater than (COUNT-1).

LNPOEND

Create a sequence of lines as a polygon.

Syntax :

```

/*- - - - - */
void LNPOEND ( Numberids, Pointids, Lineids, Status )
/*- - - - - */
/*
/* Input/Output :
/*
/*
int    *Numberids; /* Number of points in polygon.
/* Number of created lines.
/*
/* Input :
/*
int    Pointids[]; /* Array of the IDs of points ( PointIds[NumberIds] ).
/*
/* Output :
/*
/*
int    Lineids[]; /* Array of the IDs of lines ( LineIds[NumberIds] ).
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - The size of LineIds dimension should be greater or equal (NumberOfPoints - 1).

LNPTCV

Create a line using the PT-CURVE option as in the interactive system.

Syntax :

```

/*- - - - - */
int LNPTCV ( Idpt, Idcrv, Idpcv, Mode, Length, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idpt;    /* ID of point.
int    *Idcrv;   /* ID of curve.
int    *Idpcv;   /* ID of point on the curve in the approximate
/* direction.
/* If IDPT falls on IDCRV:
int    *Mode;    /* Mode of line:
/* 1 = Normal to curve
/* 2 = Tangent to curve
float  *Length;  /* Length of line to be created.
/* If IDPT does not fall on IDCRV, the subroutine
/* ignores these values.
/*
/* Output :
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created line.
/*- - - - - */

```

LNPTVE

Create a line entity defined by a point and a vector.

Syntax :

```

/*- - - - - */
int LNPTVE ( Sys, Point, Vector, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Sys;     /* Coordinate system in which POINT1 and VECTOR are
/* given:
/* 1 = Model.
/* 2 = Work.
float  Point[3]; /* X, Y, Z coordinates of starting point.
float  Vector[3]; /* The line will be created from the specified start
/* point, POINT, with the length and direction of the
/* specified vector.
/*
/* Output :
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

LNPTVED

Create a line defined by a point and a vector.

Syntax :

```

/*- - - - - */
int LNPTVED ( System, Startpoint, Vector, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int    *System;    /* The coordinate system:
/*      1 = MODEL,
/*      2 = WORK.
double Startpoint[3]; /* The coordinates of the start point.
double Vector[3];    /* The vector between start and end points.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```



General Description

The following routines allow the user to copy entities. These routines perform tasks similar to those in the **MOVE** function of the interactive system, though the routines explained below do not move entities, but only copy them. To move an entity, copy it and delete the original.

The routines **ENINST** and **ENMUIN** require the user to provide the transformation matrix needed to perform the copy. These matrices may be built by using various routines found in the section on transformations (see **TRANSFORMATION** in Chapter 4). The transformation matrix must define the same scaling factor for X, Y and Z.

The remaining routines in this section use built-in transformations and do not need a transformation matrix from the user.

CDK_OFFSET

Create an entity by offsetting another entity.

Syntax :

```
/*- - - - - */
int CDK_OFFSET ( Duplicatemode, Origin, Offset, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Duplicatemode; /*    0 : As original,
/*    1 : As active.
int    *Origin;        /* The ID of the original entity.
double *Offset;        /* The offset distance:
/*    > 0 : to the right of the curve,
/*    < 0 : to the left of the curve.
/*
/* Output :
/*
int    *Status;        /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */
```

CDK_OFFSET_CONT

Offset of the contour.

Syntax :

```

/*- - - - - */
void CDK_OFFSET_CONT ( Contour, Offset, Cornermode, TrimloopsmodeTolerance, Num,
                      Results, Status )
/*- - - - - */
/*
/* Input :
/*
int    Contour[];    /* The definition buffer of the contour.
double *Offset;      /* The given offset.
int    *Cornermode;  /* 1 = Round corners,
/* 2 = Sharp corners.
int    *Trimloopsmode; /* 0 = No trim loops,
/* 1 = Trim loops.
double *Tolerance;   /* The tolerance to find of loops.
/*
/* Input/Output :
/*
int    *Num;         /* The number of curves to be created.
/* The number of created curves.
/*
/* Output :
/*
int    Results[Num]; /* The array of created curves.
int    *Status;      /* See General Concepts-Status on page 1-2
/*- - - - - */

```

Note : - If the input value of Num < number of entities that can be created,
 the value of Results equals the number of curves requested (input value
 of Num)
 and the output value of Num equals the number of entities that can be cre
 ated.

CDK_SCALE

Create a scaled entity.

Syntax :

```

/*- - - - - */
int CDK_SCALE ( Id, Tol, Pntorg, Rotmat, Scale, Sclpar, Status )
/*- - - - - */
/*
/* Input :
/*
/* The entity's ID number.
/* The tolerance.
/* The original point ( in model system ).
/* The rotation matrix ( in model system ).
/* Scales for X, Y and Z axes.
/* The scale parameters:
/* "sclpar[0]"
/* 1 = Keep original
/* 2 = Delete original
/* "sclpar[1]"
/* 1 = Keep as original
/* 2 = Keep as active
/* "sclpar[2]"
/* 1 = Keep same attributes
/* 2 = Don't keep attributes
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of scaled entity.
/*- - - - - */

```

Notes : - Maximum tolerance and scale value =10000, minimum =0.0001.
 - The rotation matrix must be normalized.

ENINSTD

Create a transformed copy of an entity.

Syntax :

```

/*- - - - - */
int ENINSTD ( Origin, Transform, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int *Origin; /* The ID of the entity to be copied.
double Transform[12]; /* The transformation to be used. ( WORK )
/*
/* Output :
/*
/*
int *Status; /* See General Concepts-Status on page 1-2.
/* 3 = The X, Y and Z scale factors in the
/* transformation matrix are not the same.
/* The X scale factor was used for all.
/*
/* Returns :
/* ID of the created entity.
/*
/*- - - - - */

```

ENMILID

Create a copy of an entity by mirroring about the line.

Syntax :

```

/*- - - - - */
int ENMILID ( Origin, Linestart, Linedir, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int *Origin; /* ID of the entity to be copied.
double Linestart[3]; /* A point on the mirror line. ( WORK )
double Linedir[3]; /* The direction of the mirror line.
/*
/* Output :
/*
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*
/*- - - - - */

```

ENMIPLD

Create a copy of an entity by mirroring about the plane.

Syntax :

```

/*- - - - - */
int ENMIPLD ( Origin, Plane, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int *Origin; /* ID of the entity to be copied.
double Plane[4]; /* The plane coefficients ( WORK ).
/*
/* Output :
/*
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*
/*- - - - - */

```

ENMIPOD

Create a copy of an entity by mirroring through the point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int ENMIPOD ( Origin, Point )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int      *Origin;               /* ID of the entity to be copied.
double   Point[3];             /* The point coordinates ( WORK ).
                                /* Status
                                /* See General Concepts-Status on page 1-2.
                                /*
                                /* Returns :
                                /* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENMUIND

To create multiply copies of an entity.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void ENMUIND ( Origin, Transform, Num, Instances, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int      *Origin;               /* ID of the entity to be copied.
double   Transform[12];         /* The transformation to be used.
                                /*
                                /* Input/Output :
                                /*
int      *Num;                  /* = 1 : Move the entity.
                                /* > 1 : The number of copies to be created.
                                /* Output - The number of created copies.
                                /*
                                /* Output :
                                /*
int      Instances[Num];        /* The array containing the ID-s of created copies.
int      *Status;               /* See General Concepts-Status on page 1-2.
                                /*      3 = The X, Y and Z scale factors in the
                                /*      transformation matrix are not the same.
                                /*      The X scale factor was used for all.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ENOFFS

Create an offset entity at a distance, OFFSET.

Syntax :

```

/*- - - - - */
int ENOFFS ( Id, Offset, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID value of original entity.
/* Offset distance
/* > 0 OFFSET is to the right of the original entity.
/* < 0 OFFSET is to the left of the original entity.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

Note : - The direction of the curve is determined when the curve is created, i.e. from the starting point to the last endpoint. The direction of the offset is understood to be to the right or left when facing in the direction of the curve.

ENROAXD

Create a copy of an entity by rotation it about the axis.

Syntax :

```

/*- - - - - */
int ENROAXD ( Origin, Startaxis, Diraxis, Angle, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the entity to be copied.
/* The coordinates of a point on the axis ( WORK ).
/* The direction of the axis ( WORK ).
/* The rotation angle ( DEGREE ).
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

ENSCA1D

To copy an entity and transform it by a given scale factor, relative to a fixed point. The point remains unscaled.

Syntax :

```

/*- - - - - */
int ENSCA1D ( Origin, Refpoint, Scale )
/*- - - - - */
/*
/* Input :
/*
int *Origin; /* ID of the entity to be copied and transformed.
double Refpoint[3]; /* The coordinates of the fixed point ( WORK ).
double *Scale; /* The scaling factor.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

ENSHIFD

Create a copy of an entity by shifting the vector.

Syntax :

```

/*- - - - - */
int ENSHIFD ( Origin, Shift )
/*- - - - - */
/*
/* Input :
/*
int *Origin; /* ID of the entity to be copied.
double Shift[3]; /* The shift vector ( WORK ).
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

UDRAG

Drag an entity interactively.

Syntax :

```

/*- - - - - */
void UDRAG ( Id, Refpoint, Status )
/*- - - - - */
/*
/* Input :
/*
int *Id; /* The ID of the entity to drag.
/*
/* Input/Output :
/*
double *Refpoint; /* The coordinates of reference point before dragging
/* ( MODEL ).
/* The final coordinates of reference point.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```



PLACE

General Description

The following routines are used to create instances of existing masters

GRINST requires as input the transformation matrix to be used to create the instance, while GRIN3P and GRINRO simply perform rotations, mirroring and scaling based on the ID of the entities and a scaling factor.

GRIN3P

Create an instance of a previously defined master, relating the 3 reference points of the master to 3 other points (ID1, ID2, and ID3) and scaling by a factor.

Syntax :

```
/*- - - - - */
int GRIN3P ( Idgrou, Id1, Id2, Id3, ScaleStatus )
/*- - - - - */
/*
/* Input :
/*
int *Idgrou; /* ID of the master to be placed.
int *Id1; /*
int *Id2; /*
int *Id3; /* IDs of 3 points that define the instance location.
float *Scale; /* Scaling factor.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created instance.
/*- - - - - */
```

GRINRO

Create an instance of a previously defined master, by shifting its origin to **ID1**, rotating it by an angle **ANGLE** about the Z axis, mirroring it about the X or Y axis, and scaling it.

Syntax :

```

/*- - - - - */
int GRINRO ( Idgrou, Id1, Angle, Mirror, Scale, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idgrou;    /* ID of master to be placed.
int    *Id1;       /* ID of position point of instance.
float  *Angle;     /* Rotation angle.
int    *Mirror;    /* 0 = No mirror.
/*              1 = Mirror about the X axis of the work
/*              coordinate system.
/*              2 = Mirror about the Y axis of the work
/*              coordinate system.
float  *Scale;     /* Scaling factor.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created instance.
/*- - - - - */

```

GRINST

Create an instance of a previously defined master.

Syntax :

```

/*- - - - - */
int GRINST ( Idgrou, Transf, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idgrou;    /* ID of the master to be placed.
float  Transf[12]; /* Transformation matrix to be used to create instance.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*              3 = The X, Y and Z scale factors in the
/*              transformation matrix are not the same.
/*              The X scale factor was used for all.
/*
/* Returns :
/* ID of the created instance.
/*- - - - - */

```

PLEXGR

Create an instance of an external master entity.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int PLEXGR ( Pthnam, Pthlen, Grpnam, Grplen, Transf, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                     /*
                                     /* Input :
                                     /*
char   *Pthnam;                      /* The full path name of the file containing the
                                     /* external master ( excluding the .pfm suffix ).
int     *Pthlen;                     /* Number of characters in the path name (excluding the
                                     /* .pfm suffix).
char     *Grpnam;                    /* External master name.
int     *Grplen;                     /* Number of characters in external master name.
float    Transf[12];                 /* Matrix transformation ( of the new instance. )
                                     /*
                                     /* Output :
                                     /*
int     *Status;                     /* See General Concepts-Status on page 1-2.
                                     /*
                                     /* Returns :
                                     /* ID of the created instance.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

□

PLANAR_FACE

General Description

The following routines are used for planar face operations.

CDK_PLF_BOXES

Get the boxes for the given planar face.

Syntax :

```

/*- - - - - */
void CDK_PLF_BOXES( PlFace, LocalBox, ModelBox, Status )
/*- - - - - */
/*
/* Input:
/*
int *PlFace; /* The ID of the planar face to verify.
/*
/* Output:
/*
double LocalBox[4]; /* The box of the local coordinate system.
double ModelBox[6]; /* The box of the model coordinate system.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_PLF_CONCRV

Get the number of curves in a single contour.

Syntax :

```

/*- - - - - */
void CDK_PLF_CONCRV( PlFace, Contour, NumCurves )
/*- - - - - */
/*
/* Input:
/*
int *PlFace; /* The ID of the planar face to verify.
int *Contour; /* The index of contours in the planar face definition.
/* ( the first index = 1 )
/*
/* Output:
/*
int *NumCurves; /* The number of curves in the contour.
/* if NumCurves < 0 = error
/*- - - - - */

```

CDK_PLF_CREATE

Create the planar face entity.

Syntax :

```

/*- - - - - */
int CDK_PLF_CREATE( DupMode, ContourBuff, SymbolSize, Status )
/*- - - - - */
/*
/* Input:
/*
int *DupMode; /* 0 = keep original curves
/* 1 = delete original curves
int *ContourBuff; /* The contour buffer.
double *SymbolSize; /* The triangle symbol size.
/*
/* Output:
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns: The ID of the created Planar Face.
/*- - - - - */

```

CDK_PLF_EXPLODE

Explode the planar face.

Syntax :

```

/*- - - - - */
void CDK_PLF_EXPLODE( PlFace, Curves, NumCurves )
/*- - - - - */
/*
/* Input:
/*
int *PlFace; /* The ID of planar face to explode.
/*
/* Output:
/*
int *Curves; /* The array IDs of created curves.
int *NumCurves; /* > 0 = Number of created curves, < 0 = Error
/*- - - - - */

```

CDK_PLF_GETCRV

Get the ID of a curve from the planar face definition.

Syntax :

```

/*- - - - - */
void CDK_PLF_GETCRV( PlFace, Contour, Curve, Id )
/*- - - - - */
/*
/* Input:
/*
int *PlFace; /* The ID of the planar face to verify.
int *Contour; /* The index of contours in the planar face definition.
/* ( the first index = 1 )
int *Curve; /* The index of curves in the contour definition.
/* ( the first index = 1 )
/*
/* Output:
/*
int *Id; /* The ID of the component curve.
/* if Id < 0 = error
/*- - - - - */

```

CDK_PLF_TOTCRV

Get the number of contours and total number of curves.

Syntax :

```

/*- - - - - */
void CDK_PLF_TOTCRV( PlFace, NumContours, NumCurves )
/*- - - - - */
/*
/* Input:
/*
int *PlFace; /* The ID of the planar face to verify.
/*
/* Output:
/*
int *NumContours; /* The number of contours defining the planar face.
int *NumCurves; /* The total number of curves defining the planar face.
/* if NumCurves < 0 = error
/*
/*- - - - - */

```

UPLFTS

Convert a planar face to a trimmed surface.

Syntax :

```

/*- - - - - */
void UPLFTS ( Tol, Idplf, Idtrs, Status )
/*- - - - - */
/*
/* Input :
/*
double *Tol; /* Tolerance ( Approximation to a polygon ).
int *Idplf; /* ID of the planar face.
/*
/* Output :
/*
int *Idtrs; /* ID of the created trimmed surface.
int *Status; /* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```

UTPFWP

Trim a planar face by the WORK plane.

Syntax :

```

/*- - - - - */
int UTPFWP ( Idplf, Tol, Delorg, Arrsize, Rp, Res )
/*- - - - - */
/*
/* Input :
/*
int *Idplf; /* ID of planar face to trim.
float *Tol; /* Tolerance for checking intersections:( 0.01 ( TOL
/* * MM/UNIT ) 10, recommended = 0.1 ).
int *Delorg; /* 1 = Delete the original planar face,
/* 0 = Keep the original planar face.
float *Arrsize; /* Size of datum symbol of the planar face.
float Rp[3]; /* Coordinates of indication point in MODEL UCS.
/*
/* Output :
/*
int *Res; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the trimmed planar face.
/*- - - - - */

```

UTRMPF

Trim a planar face by contours.

Syntax :

```

/*- - - - - */
int UTRMPF ( Id, Crvlst, Ncon, Tol, Rp, Delorg, Arrsize, Res )
/*- - - - - */
/*
/* Input :
/*
int *Id; /* ID of planar face to trim.
int *Crvlst; /* Buffer of trimming curves :
/* ( - )NC_1, IDCrv_1_1, IDCrv_1_2, ...IDCRV_1_NC_1,
/* ( - )NC_2, IDCrv_2_1, IDCrv_2_2, ...IDCRV_2_NC_2,
/* . . . .
/* . . . .
/* ( - )NC_NCON, IDCrv_NCON_1, IDCrv_NCON_2,
/* ...IDCRV_NCON_NC_NCON If CONTOUR(N) is open : NC_N =
/* -NC N
int *Ncon; /* Number of trimming contours.
float *Tol; /* Tolerance for checking intersections:( 0.01 ( TOL
/* * MM/UNIT ) 10, recommended = 0.1 ).
float Rp[3]; /* Coordinates of indication point in MODEL UCS.
int *Delorg; /* 1 = Delete the original planar face,
/* 0 = Keep the original planar face.
float *Arrsize; /* Size of datum symbol of the planar face.
/*
/* Output :
/*
int *Res; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the trimmed planar face.
/*- - - - - */

```



POINT

CDK_PNT_BSPLINE

Create points of a B-Spline.

Syntax :

```

/*- - - - - */
int CDK_PNT_BSPLINE( IdBsp, Opt, NumId, Ids )
/*- - - - - */
/*
/* Input :
/*
int    IdBsp;    /* ID of required B-Spline.
int    Opt;      /* Points option:
/*      1 : Control points.
/*      2 : Through points.
/*
/* Input/Output :
/*
int    *NumId;   /* I - MAX IDs to obtain.
/*              /* 0 - The number IDs created.
/*              /* If there isn't enough memory return -33.
/*              /* Use 0 when you need to know the amount of memory
/*              /* to allocate.
/*
/* Output :
/*
int    Ids[];    /* Buffer of points IDs.
/* Returns :
/* See Chapter 1, General Concept, Status.
/*- - - - - */

```

CDK_PNT_COOR

Create a point entity defined by its coordinates.

Syntax :

```

/*- - - - - */
int CDK_PNT_COOR ( System, Coordx, Coordy, Coordz )
/*- - - - - */
/*
/* Input :
/*
int    *System;  /* 1 = Model, 2 = Work coordinate system.
double *Coordx;  /* The 'X' coordinate of the point.
double *Coordy;  /* The 'Y' coordinate of the point.
double *Coordz;  /* The 'Z' coordinate of the point.
/*
/* Returns :
/* ID of the created point. : ERROR.
/*- - - - - */

```

PNTEX3

Create a point entity defined as the intersection of two entities ID1 and ID2 or their extensions closest to two given points IDP1 and IDP2.

Syntax :

```

/*- - - - - */
int PNTEX3 ( Id1, Idp1, Id2, Idp2, Tol, Status )
/*- - - - - */
/*
/* Input :
/*
int *Id1; /* ID of the first curve.
int *Idp1; /* ID of the point on the first curve close to the
/* intersection point.
int *Id2; /* ID of the second curve.
int *Idp2; /* ID of the point on the second curve close to the
/* intersection point.
double *Tol; /* The tolerance.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

PNTIN3

Create a point entity defined as the intersection of two entities ID1 and ID2 closest to two given points IDP1 and IDP2.

Syntax :

```

/*- - - - - */
int PNTIN3 ( Id1, Idp1, Id2, Idp2, Tol, Status )
/*- - - - - */
/*
/* Input :
/*
int *Id1; /* ID of the first curve.
int *Idp1; /* ID of the point on the first curve close to the
/* intersection point.
int *Id2; /* ID of the second curve.
int *Idp2; /* ID of the point on the second curve close to the
/* intersection point.
double *Tol; /* The tolerance.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

PTCDIS

Create point entities as in POINT >> MULTI-POINTS >> BY DISTANCE. See Chapter 1, of the **Modeling Manual**.

Syntax :

```

/*- - - - - */
void PTCDIS ( Idc, Idp, Iddirp, Outar, Count, Dist, Res )
/*- - - - - */
/*
/* Input :
/*
int *Idc; /* ID of a curve.
int *Idp; /* ID of a point on the curve.
int *Iddirp; /* ID of a direction point ( as in the Interactive
/* System ).
/*
/* Output :
/*
int *Outar; /* Array containing IDs of the created points (if <= 0).
/*
/* Input :
/*
int *Count; /* Number of points to create.
float *Dist; /* Distance between points.
/*
/* Output :
/*
int *Res; /* Error flag:
/* -1 = Error
/* 0 = O.K. Outar contains Count IDs of created
/* points.
/*- - - - - */

```

PTCENT

Create a point entity defined as the center point of an arc/circle.

Syntax :

```

/*- - - - - */
int PTCENT ( Id, Status )
/*- - - - - */
/*
/* Input :
/*
int *Id; /* ID of the arc/circle.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

PTCOMO

Create a point defined by its coordinates in the model system.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int PTCOMO ( X, Y, Z )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float *X;
float *Y;
float *Z;
/* Coordinates of the point entity.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

PTCOWO

Create a point defined by its coordinates in the work system.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int PTCOWO ( X, Y, Z )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
float *X;
float *Y;
float *Z;
/* Coordinates of the point entity.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

PTEND

Create a point entity defined as the endpoint of a curve.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int PTEND ( Id, Select, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int *Id;
char *Select;
/* ID of the curve.
/* See Chapter 1, General Concepts, Select.
/*
/* Output :
/*
/*
int *Status;
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

PTEXIN

Create a point entity defined as the intersection of two entities ID1 and ID2, or their extensions. Only those points defining the intersection of coplanar entities are created.

Syntax :

```

/*- - - - - */
int PTEXIN ( Id1, Id2, Select, Status )
/*- - - - - */
/*
/* Input :
/*
int *Id1;
int *Id2;
char *Select;
/* IDs of the intersecting entities.
/* See Chapter 1, General Concepts, Select.
/*
/* Output :
/*
int *Status;
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

PTGET

Create a point via interaction with the user.

Syntax :

```

/*- - - - - */
int PTGET ( Option, Respon )
/*- - - - - */
/*
/* Input/Output :
/*
int *Option;
/* Method to be used to define the point:
/* 0 = Choose from submenu.
/* 1 = SCREEN position.
/* 2 = END point of an entity.
/* 3 = MID point of an entity.
/* 4 = INTERSection of two curves.
/* 5 = CENTER of an arc or a conic.
/* 6 = PIERCE point of a curve and the work plane.
/* 7 = CLOSEst point on a curve to the point
/* indicated.
/* 8 = PICK existing point entity.
/* 9 = KEY IN coordinates of a point.
/* Option chosen ( if RESPON=0 ). This value may differ
/* from the input value if changed by the operator via
/* the POINT submenu.
/*
/* Output :
/*
int *Respon;
/* See Chapter 1, General Concepts, Mouse Response.
/*
/* Returns :
/* ID of the created point ( if RESPON=0 ).
/*- - - - - */

```

PTINCR

Find all intersection points of two coplanar curves.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void PTINCR ( Id1, Id2, Parint, Maxint, Numpnt, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Id1;      /* ID of 1st curve.
int    *Id2;      /* ID of 2nd curve.
/*
/* Output :
/*
double *Parint;    /* Parameters of the intersection points on both curves.
/*
/* Input :
/*
int    *Maxint;    /* Maximum number of intersection points.
/*
/* Output :
/*
int    *Numpnt;    /* Number of intersection points found.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - The parameters in PARINT are given in pairs:PARINT (U11, U21, ... U1n, U2n) where Uij is the parameter of the jth intersection on the ith curve, i [1, 2].

PTINTR

Create a point entity defined as the intersection of two entities, ID1 and ID2. Only those points defining the intersection of coplanar entities are created.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int PTINTR ( Id1, Id2, Select, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Id1;      /*
int    *Id2;      /* IDs of the intersecting entities.
char    *Select;   /* See Chapter 1, General Concepts, Select.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - To create a point entity of the intersection of two entities that do not meet unless extended, use the function PTEXIN.

PTM2PT

Create a point at a specified distance, **DIST**, from the midpoint of a diagonal or one of the edges of a rectangle. The rectangle is defined by the endpoints of one diagonal.

How To:

1. Identify the two points which define a diagonal of the rectangle. The direction, required later when SIDE is specified, is assumed to be from the first point, IDP1, to the second point, IDP2.
2. Use SELECT to identify the line from whose midpoint the distance, DIST, will be measured.
3. Use SIDE to indicate the side of the selected line on which the point entity is to be created. It will be either to the right, or to the left, when facing in the direction established in Step 1. For the straight edges of a rectangle, the side will be approximately right or left.
4. Specify the distance, DIST, between the new point and the midpoint of the selected straight edge or the diagonal.

Syntax :

```

/*- - - - - */
int PTM2PT ( Idp1, Idp2, Select, Side, Dist, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of 1st point.
/* ID of 2nd point.
/* Specify the line from whose midpoint the distance
/* will be measured. The direction is from IDP1 to IDP2:
/* "XLARGE" : Right straight edge of the rectangle.
/* "XSMALL" : Left straight edge of the rectangle.
/* "YLARGE" : Top straight edge of the rectangle.
/* "YSMALL" : Bottom straight edge of the rectangle.
/* "ZLARGE/ZSMALL" :
/* Diagonal of the rectangle defined by
/* IDP1 and IDP2.
int *Side;
/* 1 = Create the new point on the LEFT of the
/* chosen line.
/* 2 = Create the new point on the RIGHT of the
/* chosen line.
float *Dist;
/* Distance between the new point and the midpoint of
/* the chosen line.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created point ( if STATUS=0 ).
/*- - - - - */

```

PTMIDD

Create a point entity defined as the midpoint of a curve.

Syntax :

```

/*- - - - - */
int PTMIDD ( Id, Status )
/*- - - - - */
/*
/* Input :
/*
int *Id; /* ID of the curve.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

UPNTCENT

Create a point entity defined as the center of the curve.

Syntax :

```

/*- - - - - */
int UPNTCENT ( Idcrv, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idcrv; /* The ID of the curve.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/* -31 = "IdCrv" is more than one periodic curve.
/*
/* Returns :
/* ID of the created point.
/*- - - - - */

```

UPNTCM

Create a point entity defined by its coordinates in the Model System.

Syntax :

```

/*- - - - - */
int UPNTCM ( Coordinatex, Coordinatey, Coordinatéz )
/*- - - - - */
/*
/* Input :
/*
double *Coordinatex; /* The 'X' coordinate of the point ( MODEL ).
double *Coordinatey; /* The 'Y' coordinate of the point ( MODEL ).
double *Coordinatéz; /* The 'Z' coordinate of the point ( MODEL ).
/*
/* Returns :
/* ID of the created point. : ERROR.
/*- - - - - */

```

UPNTCW

Create a point entity defined by its coordinates in the WORK System.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int UPNTCW ( Coordinatex, Coordinatey, Coordinatez )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double *Coordinatex; /* The 'X' coordinate of the point ( WORK ).
double *Coordinatey; /* The 'Y' coordinate of the point ( WORK ).
double *Coordinatez; /* The 'Z' coordinate of the point ( WORK ).
/*
/* Returns :
/* ID of the created point. : ERROR.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UPNTEND

Create a point entity defined as the endpoint of the curve.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int UPNTEND ( Idcrv, Select, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Idcrv; /* ID of the curve.
char *Select; /* See Chapter 2, Section 2.1, Select.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created point.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UPNTGET

Create a point entity interactively.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int UPNTGET ( Option, Response )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input/Output :
/*
/*
int    *Option;    /* In : Default method to pick the point:
/*      0 = Choose from submenu.
/*      1 = SCREEN position.
/*      2 = END point of an entity.
/*      3 = MID point of an entity.
/*      4 = INTERSection of two curves.
/*      5 = CENTER of an arc or a conic.
/*      6 = PIERCE point of a curve and work plane.
/*      7 = CLOSEst point on a curve to the indicated
/*            point.
/*      8 = PICK existing point entity.
/*      9 = KEY IN coordinates of the point.
/* OUT : Choice method to pick point.
/*
/* Output :
/*
int    *Response;  /* See Chapter 1, General Concepts, Mouse Response.
/*
/* Returns :
/* ID of the created point.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UPNTMID

To create a point entity defined as a middle point of the curve.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int UPNTMID ( Idcrv, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Idcrv;    /* ID of the curve.
/*
/* Output :
/*
int    *Status;   /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created point.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```



PROJECT

CDK_PROJ_CNTR_PLANE

Project a curve onto the active work plane.

Syntax:

```

/*- - - - - */
int    CDK_PROJ_CNTR_PLANE (cont, sizeofCont, idup, moddup, prjtyp, dir, idPrj)
/*- - - - - */
/*
/* Input :
/*
int    cont[];    /* contour database.
int    sizeofCont; /* size of contour buffer.
int    idup;       /* =1 : Create a new projected entity
/* =2 : Project the entity and delete the original one
/*
int    moddup;     /* = 0 : Duplicate with the original entity's
/*          attributes : Level, Line-Font, Color, Pen
/* = 1 : Duplicate with the active values
/* =10 : As 0 but with NGD attributes
/* =11 : As 1 but with NGD attributes
/*
int    prjtyp;     /* = 1 : Perpend. Projection
/* = 2 : Direction Projection
double dir[];      /* the direction of the projection, (when prjtyp == 2)
/*
int    idPrj[];    /* the new projected Curves IDs.
/*
/* Return value:
/* > 0, the num of curves whose projection succeeded.
/* < 0, status.
/*- - - - - */
- Note: the idPrj buffer must be sufficient for all new created
      curves IDs.

```

CDK_PROJ_CNTR_SRF Project a contour onto a surface.
Syntax:

```

/*- - - - - */
int   CDK_PROJ_CNTR_SRF (cont, sizeofCont, tol, srfID, pt_ind, dir, idPrj)
/*- - - - - */
                        /*
                        /* Input:
                        /*
int     cont[];         /* contour database.
int     sizeofCont;     /* size of contour buffer.
double  tol;            /* tolerance.
int     srfID;          /* the ID of surface that is being projected onto.
double  pt_ind[];       /* Indication point on the surface. If pt_ind is not
                        /* NULL - the projection is LOCAL! (Projection will not
                        /* pierce the surface). To get GLOBAL projection send
                        /* NULL! (i.e. Project on all faces of surface.
double  dir[];          /* the direction of the projection.
                        /*
                        /* Output:
int     **idPrj;        /* id's of curves that were projected.
                        /*
                        /* Return value:
                        /* >= 0, the num of curves whose projection succeeded.
                        /* < 0, status
/*- - - - - */
Notes:  - 1. must call afterwards to "CDK_PROJ_CNTR_SRF_FREE()".
        - 2. A projection of one contour assembled from x curves might
            resolve creation of more x new curves. Therefore, it is important
            to check the return value, in order to find out how many new
            curves were created.

```

CDK_PROJ_CNTR_SRF_FREE Free memory that was allocated in
 CDK_PROJ_CNTR_SRF.

```

/*- - - - - */
void CDK_PROJ_CNTR_SRF_FREE()
/*- - - - - */

```

CDK_PROJ_CURVE_PLANE

Project a curve onto the active work plane.

Syntax:

```

/*- - - - - */
int   CDK_PROJ_CURVE_PLANE (idup, moddup, prjtyp, dir, curveID)
/*- - - - - */
/*
/* Input :
/*
int   idup;      /* =1 : Create a new projected entity
/*              /* =2 : Project the entity and delete the original one
/*
int   moddup;    /* = 0 : Duplicate with the original entity's
/*              /* attributes : Level, Line-Font, Color, Pen
/*              /* = 1 : Duplicate with the active values
/*              /* =10 : As 0 but with NGD attributes
/*              /* =11 : As 1 but with NGD attributes
/*
int   prjtyp;    /* = 1 : Perpend. Projection
/*              /* = 2 : Direction Projection
double dir[];    /* the direction of the projection, (when prjtyp == 2)
/*
int   curveID;   /* Curve ID.
/*
/* Return value
/*
/*
/* > 0 : ID. of the created entity.
/* =-1 : status
/*- - - - - */

```

CDK_PROJ_CURVE_SRF

Project a curve onto a surface.

Syntax:

```

/*- - - - - */
int   CDK_PROJ_CURVE_SRF ( crvID, tol, srfID, pt_ind, dir, newIDs)
/*- - - - - */
/*
/* Input:
/*
int   crvID;     /* id of curve to project.
double tol;      /* tolerance.
int   srfID;     /* the ID of surface that is being projected on.
double pt_ind[]; /* Indication point on the surface. if pt_ind is not
/*              /* NULL - the projection is LOCAL! (Projection will not
/*              /* pierce the surface). To get GLOBAL projection send
/*              /* NULL! (i.e. Project on all faces of surface.
double dir[];    /* the direction of the projection.
/*
/* Output:
/*
int   **newIDs;  /* address of ptr to buffer of the new IDs.
/*              /* THE BUFFER IS ALLOCATED IN THE FUNCTOIN.
/*
/* Return value:
/*              /* >= 0 -> num of new projected curves.
/*              /* other -> status.
/*- - - - - */

```

Notes: - 1. Must call afterwards to "cdk_proj_curve_srf_free()".
 2. A projection of one curves might resolve the creation of more than one new curve. For example, a spline whose middle fell out of the surface: two new curves will be created, the start of the spline, and the end of the spline.

CDK_PROJ_CURVE_SRF_FREE

Free memory that was allocated in
CDK_PROJ_CURVE_SRF.

Syntax:

```
/*- - - - - */
void CDK_PROJ_CURVE_SRF_FREE()
/*- - - - - */
```

CDK_PROJ_POINT_PLANE

Project a point onto a plane.

Syntax:

```
/*- - - - - */
int  CDK_PROJ_POINT_PLANE (pt, prjtyp, dir)
/*- - - - - */
/*
/* purpose : Project a point onto active workplane, and
/*          return its ID.
/*
/* Input:
/*
double  pt[]; /* the point to be projected.
int      prjtyp; /* normal projection: prjtyp = 1,
/* directional projection : prjtyp = 2.
double  dir[]; /* the direction of the projection, (when prjtyp == 2)
/*
/* Return value:
/* on success, the id of the new created point.
/* on failure, return status.
/*
/*- - - - - */
```

CDK_PROJ_POINT_SRF

Project a point onto a surface.

Syntax:

```

/*- - - - - */
int   CDK_PROJ_POINT_SRF (pt, srfID, dir, tol, side, newPt, newIDs)
/*- - - - - */
                        /*
                        /* Input:
                        /*
double  pt[];           /* the coordinates of the point to be projected.
int     srfID;          /* id of the surface that is being projected onto.
double  dir[];          /* the direction of the projection.
double  tol;            /* the tolerance.
int     side;           /* 1 : Both sides of "dir" are active
                        /* 2 : Only one side of "dir" is active
double  newPt[];        /* NULL : do global projection
                        /* else : do local projection by projecting
                        /* globally and then choosing the closest
                        /* to this point in MODEL
                        /*
                        /* Output:
                        /*
int     **newIDs;        /* the IDs of the new created points. memory is allocated
                        /* in the function, send the address of a ptr, and use
                        /* CDK_PROJ_POINT_SRF_FREE afterwards.
                        /*
                        /* Return value
                        /*   >= 0 num of new created points.
                        /*   < 0 status.
/*- - - - - */
Note: remember to use CDK_PROJ_POINT_SRF_FREE after calling the function.

```

CDK_PROJ_POINT_SRF_FREE

Free memory that was allocated in CDK_PROJ_POINT_SRF.

Syntax:

```

/*- - - - - */
void CDK_PROJ_POINT_SRF_FREE ()
/*- - - - - */

```

CDK_PROJ_WPDIR

Create Projected Entities onto the work plane according to a given direction.

Syntax :

```

/*- - - - - */
void CDK_PROJ_WPDIR ( Copymode, Duplicatemode, Direction, Originals, Numberids,
                    Projections, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Copymode;    /*    1 = Keep original,
/*    2 = Delete original.
int    *Duplicatemode; /*    0 = As original,
/*    1 = As active.
double Direction[3]; /* The project direction ( MODEL ).
int    Originals[];  /* Array if IDs of entities to project.
/*
/* Input/Output :
/*
int    *Numberids;   /* Number of entities to project.
/* Number of created projected entities.
/*
/* Output :
/*
int    Projections[]; /* Array if IDs of created projected entities.
int    *Status;       /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_PROJ_WPNORM

Create Projected Entities onto the work plane according to the Normal direction.

Syntax :

```

/*- - - - - */
void CDK_PROJ_WPNORM ( Copymode, Duplicatemode, Originals, Numberids,
                    Projections, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Copymode;    /*    1 = Keep original,
/*    2 = Delete original.
int    *Duplicatemode; /*    0 = As original,
/*    1 = As active.
int    Originals[];  /* Array if IDs of entities to project.
/*
/* Input/Output :
/*
int    *Numberids;   /* Number of entities to project.
/* Number of created projected entities.
/*
/* Output :
/*
int    Projections[]; /* Array if IDs of created projected entities.
int    *Status;       /* See General Concepts-Status on page 1-2.
/*- - - - - */

```


PRCRSR

Create the projection of a curve on a surface.

Syntax :

```

/*- - - - - */
void PRCRSR ( Tol, Idcrv, Optlim, Lim, Idsrf, Uvini, Dir, Maxcrv, Idc, Nc )
/*- - - - - */
/*
/* Input :
/*
/* User-defined tolerance.
/* The curve to be projected.
/* 1 = Take curve limits from database.
/* 2 = Take curve limits from LIM.
/* Curve limits if OPTLIM = 2.
/* The surface.
/* The initial surface parameters.
/* Projection direction.
/* Maximal number of curves to be created.
/*
/* Output :
/*
/* Projected curve( s ).
/* Number of created curves. If 0 - error.
/*
/*- - - - - */
double *Tol;
int *Idcrv;
int *Optlim;
double Lim[2];
int *Idsrf;
double Uvini[2];
double Dir[3];
int *Maxcrv;
int Idc[Nc];
int *Nc;

```

PRJDR1

Create entities projected onto the XY plane of the active UCS, in the direction DIR.

Syntax :

```

/*- - - - - */
void PRJDR1 ( Dir, Idup, Opline, Moddup, Count, Ids, Outar, Status )
/*- - - - - */
/*
/* Input :
/*
/* Array containing the model system coordinates
/* defining the direction in which projection will take
/* place.
/* 1 = Project entities and keep originals.
/* 2 = Project entities and delete the originals.
/* 1 = Connect the endpoints of the original
/* entities to the corresponding endpoints
/* of the projected entities.
/* 2 = Do not connect the endpoints.
/* 1 = Copy the attributes of the original entity
/* ( Level, Line-Font, Color, Pen ), to the
/* projected entity.
/* 2 = Assign the active attributes to the projected
/* entity.
/* Number of entities in IDS.
/* Array of IDs of entities to be projected.
/*
/* Output :
/*
/* Array containing IDs of created entities.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */
float Dir[3];
int *Idup;
int *Opline;
int *Moddup;
int *Count;
int Ids[Count];
int Outar[Count];
int *Status;

```

Note : - If OPLINE = 1, the length of OUTAR must be sufficient to hold the IDs of the connecting lines.

PRJPD1

Create entities projected perpendicularly onto the XY plane of the active UCS.

Syntax :

```

/*- - - - - */
void PRJPD1 ( Idup, Opline, Moddup, Count, Ids, Outar, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idup;      /* 1 = Project the entities and keep originals.
/* 2 = Project the entities and delete the originals.
int    *Opline;    /* 1 = Connect the endpoints of the original
/* entities to the corresponding endpoints of
/* the projected entities.
/* 2 = Do not connect the endpoints.
int    *Moddup;    /* 1 = Copy the attributes of the original entity
/* ( Level, Line-Font, Color, Pen ), to the
/* projected entity.
/* 2 = Assign the active attributes to the projected
/* entity.
int    *Count;     /* Number of entities in IDS.
int    Ids[Count]; /* Array of IDs of entities to be projected.
/*
/* Output :
/*
int    Outar[Count]; /* Array containing IDs of created entities.
int    *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - If OPLINE = 1, the length of OUTAR must be sufficient to hold the IDs of the connecting lines.

PROJDR

Create entities projected onto the current work plane, in the direction DIR.

Syntax :

```

/*- - - - - */
void PROJDR ( Dir, Idup, Opline, Moddup, Count, Ids, Outar, Status )
/*- - - - - */
/*
/* Input :
/*
float   Dir[3];
/* Array containing the model system coordinates
/* defining the direction in which projection will take
/* place.
int     *Idup;
/* 1 = Project entities and keep originals.
/* 2 = Project entities and delete the originals.
int     *Opline;
/* 1 = Connect the endpoints of the original
/* entities to the corresponding endpoints of
/* the projected entities.
/* 2 = Do not connect the endpoints.
int     *Moddup;
/* 1 = Copy the attributes of the original entity
/* ( Level, Line-Font, Color, Pen ), to the
/* projected entity.
/* 2 = Assign the active attributes to the projected
/* entity.
int     *Count;
/* Number of entities in IDS.
int     Ids[Count];
/* Array of IDs of entities to be projected.
/* Output :
/*
int     Outar[Count];
/* Array containing IDs of created entities.
int     *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - If OPLINE = 1, the length of OUTAR must be sufficient to hold the IDs of the connecting lines.

PROJPD

Create entities projected perpendicularly onto the current work plane.

Syntax :

```

/*- - - - - */
void PROJPD ( Idup, Opline, Moddup, Count, Ids, Outar, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idup;      /* 1 = Project the entities and keep originals.
/*                /* 2 = Project the entities and delete the
/*                /* originals.
int    *Opline;    /* 1 = Connect the endpoints of the original
/*                /* entities to the corresponding endpoints
/*                /* of the projected entities.
int    *Moddup;    /* 2 = Do not connect the endpoints.
/*                /* 1 = Copy the attributes of the original entity
/*                /* ( Level, Line-Font, Color, Pen ), to the
/*                /* projected entity.
/*                /* 2 = Assign the active attributes to the
/*                /* projected entity.
int    *Count;     /* Number of entities in IDS.
int    Ids[Count]; /* Array of IDs of entities to be projected.
/*
/* Output :
/*
int    Outar[Count]; /* Array containing IDs of created entities.
int    *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - If OPLINE = 1, the length of OUTAR must be sufficient to hold the IDs of the connecting lines.



Create a polygon by stepping along a curve within tolerance.

```

/*- - - - - */
void CDK_CRV2POLY( idcrv, opt, tol, param, npar, max_np, pol, par_pol, np )
/*- - - - - */
    /*
    /* Input :
    /*
    /* Curve ID.
    /* = 0 : Return polygon & parameters of polygon on
    /* curve.
    /* = 1 : Return polygon only.
    /* = 2 : Return parameters of polygon on curve only.
    double tol;
    /* Tolerance.
    double param[];
    /* Curve parameters ( if npar > 0 only !! ).
    /* Gives the option to force a point of the polygon to
    /* be a point from the curves parameters points.
    /* Use NULL when this option is not needed.
    int npar;
    /* = 0 : parameters of curve are not given.
    /* > 0 : Number of parameters in param[].
    int max_np;
    /* Maximum number points in polygon.
    /*
    /* Output :
    /*
    double pol[];
    /* The polygon buffer (size of buffer must be at least
    /* 3 * max_np ).
    double par_pol[];
    /* The parameters buffer ( size of buffer must be at
    /* least max_np ).
    /* Use NULL if opt = 1.
    int *np;
    /* > 0 : Number of points returned.
    /* < 0 : See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_CRV_APPROX

Approximation curves by continuous arcs and lines.

Syntax :

```

/*- - - - - */
void CDK_CRV_APPROX ( Curveid, Tol, Rmin, Rmax, On_free, Keep_deleteOrig_active,
                    Nument, Entarray, Bpnumber, Bparray, Status )
/*- - - - - */

/*
/* Input :
/*
int    *Curveid;    /* ID of the curve to be approximated.
float  *Tol;        /* Approximation tolerance.
float  *Rmin;       /* Minimum arc radius allowed.
float  *Rmax;       /* Maximum arc radius allowed.
int    *On_free;    /* 1 = Force the function to place start and end
/*                  points of the approximated chain ( not for
/*                  every line and arc ) on the original curve.
/*                  2 = Try to find the best solution (no limitations
/*                  for start and end point ).
int    *Keep_delete; /* 1 = Delete original curve after performing
/*                  operation.
/*                  2 = Keep original curve after performing
/*                  operation.
int    *Orig_active; /* 1 = Use original curve color, pen and line
/*                  font for the new lines and arcs.
/*                  2 = Use active color, pen and line font for
/*                  the new lines and arcs.
/*
/* Input/Output :
/*
int    *Nument;     /* Maximum number of entities to be created.
/*                  If Status == 0 :
/*                  Number of created entities.
/*                  If Status == 1 :
/*                  The possible number of entities to be created
/*                  with the current input.
/*                  if Status == -1 :
/*                  Number of entities created before an emergency
/*                  return.
/*
/* Output :
/*
int    Entarray[];  /* Array of identifiers of created entities. Size of
/*                  array must be equal to NumEnt.
int    *Bpnumber;   /* Number of break points.
double Bparray[];   /* Array of break point coordinates.
/*                  Size of array must be equal to NumEnt * 3.
int    *Status;     /* See General Concepts-Status on page 1-2.
/*                  1 : Size of array EntArray is less than the
/*                  required amount ( see also output value of
/*                  NumEnt parameter ).
/*- - - - - */

```

CDK_CRV_DST_CRV

Find the closest points between two curves.

Syntax:

```

/*- - - - - */
int CDK_CRV_DST_CRV ( crvID1, crvID2, par1, par2, pt1, pt2 )
/*- - - - - */
int      crvID1;      /* I  : ID of the first curve.      */
int      crvID2;      /* I  : ID of the second curve.     */
double   *par1;        /* I/O: IN - initial parameter of the first curve. */
                        /*      OUT - output parameter of the first curve. */
double   *par2;        /* I/O: IN - initial parameter of the second curve. */
                        /*      OUT - output parameter of the second curve. */
double   pt1[3];       /* O  : Point on the first curve.    */
double   pt2[3];       /* O  : Point on the second curve.   */
                        /* R  : Status.                      */
/*- - - - - */

```

CDK_CRV_TRMLOOP

Prepare given planar curve to chain of curves without loops.

Syntax :

```

/*- - - - - */
void CDK_CRV_TRMLOOP ( Curve, Tol, Duplicatemode, Numresults, Trmcurves,
                        Status )
/*- - - - - */
                        /*
                        /* Input :
                        /*
int      *Curve;      /* The ID of planar non periodic curve to be prepared.
double   *Tol;        /* The given tolerance.
int      *Duplicatemode; /* 0 = as origin,
                        /* 1 = as active.
                        /*
                        /* Input/Output :
int      *Numresults; /* The expected number of resulting curves.
                        /* The number of resulting curves.
                        /*
                        /* Output :
int      Trmcurves[]; /* The sequence of created curves.
int      *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - If the input value of NumResults < number of entities that can be created, the value of TrmCurves equals the number of curves requested (input value of NumResults) and the output value of NumResults equals the number of entities that can be created.

CDK_CURVE_FAIR

Execute fairing on a curve.

Syntax :

```

/*- - - - - */
int CDK_CURVE_FAIR ( Curve, Tol, Attrmode, Slopemode, Slopes, Status )
/*- - - - - */
/*
/* Input :
/*
int *Curve; /* The ID of the curve to be faired.
double *Tol; /* The required tolerance.
int *Attrmode; /* 1 = as active,
/* 2 = as origin.
int *Slopemode; /* Define the boundary conditions.
/* 0 = Free tangents at the end points.
/* 1 = 1st tangent is given in Slopes[0..Dim - 1].
/* 2 = 2nd tangent is given in Slopes[0..Dim - 1].
/* 3 = Both tangents are given Slopes[0..Dim*2 - 1].
double Slopes[]; /* The tangents at the end points ( Unit vectors ).
/* Dim = 2 : Slopes are defined in LOCAL coordinates.
/* Dim = 3 : Slopes are defined in MODEL coordinates.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

CDK_SPLINE_ADJOIN

Adjoin curves.

Syntax :

```

/*- - - - - */
int CDK_SPLINE_ADJOIN( dim, gap, ncrv, idcrv, gapsz )
/*- - - - - */
/*
/* Input :
/*
int dim; /* Dimension of curves ( 2 : 2D, 3 : 3D ).
/* If dim = 2, WORK plane will be used.
double gap[2]; /* gap[0] = Automatically adjoined gaps.
/* gap[1] = Gaps to be displayed.
int ncrv; /* Number of curves to be checked.
int idcrv[]; /* Buffer of curve IDs.
/*
/* Output :
/*
double *gapsz; /* Size of largest gap found.
/* gap_buf[0] < gap_sz < gap_buf[1].
/* Returns:
/* Number of gaps found. ( < 0 : Error. )
/*- - - - - */

```

CDK_SPLINE_CP

Create a B spline by defining its control points.

Syntax :

```

/*- - - - - */
int CDK_SPLINE_CP ( Dim, Depth, Deg, Perflag, Weightflag, Numpoints, Points,
                  Weights, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Dim;          /*      2 = Create 2D-Spline,
/*      3 = 3D-Spline.
double *Depth;        /* The depth of the splineEs plane ( if Dim == 2 ).
int    *Deg;          /* The degree of the blending functions.
int    *Perflag;       /*      0 = Create non-periodic spline,
/*      1 = periodic.
int    *Weightflag;    /*      0 = Create rational spline ( with weights ),
/*      1 = non rational ( no weights ).
int    *Numpoints;     /* The Number control points of the spline.
/*
/* Output :
/*
double Points[];       /* The coordinates of control points
/*
/* Input :
/*
double *Weights;        /* The buffer contains the weights, this buffer is used
/* only when WeightFlag == 0 ( Weights[NumPoints] ).
/*
/* Output :
/*
int    *Status;         /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

CDK_SPLINE_CREATE Create a NURBS spline entity (with knots).
Syntax:

```

/*- - - - - */
void CDK_SPLINE_CREATE ( Dim, Deg, OpenClose, PerFlag, optknt, WeightsFlag,
                        optpts, NumPoints, lim, knt, Weights, Points, Spline, Status )
/*- - - - - */

/*
/* Input:
/*
int    *Dim;          /* = 2 : 2D spline
/* = 3 : 3D spline
int    *Deg;          /* The degree of the spline
int    *OpenClose;    /* = 0 : Open      ; = 1 : Closed
int    *PerFlag;      /* = 0 : Non-Periodic ; = 1 : Periodic
int    *optknt;       /* = 1 : Knots are given , lim[] is NOT given
/* = 0 : Knots & lim[3] are given
/* = -1 : Knots are not given (Default: uniform NURBS)
/*          So, in this case lim[] is a dummy argument
int    *WeightsFlag;  /* = 0 : With weights; = 1 : No weights
/* = -1 : With weights (but Weights is not given yet)
int    *optpts;       /* = 0 : points are given
/* = -1 : points are not given yet
int    *NumPoints;    /* No. of control points
double  lim[3];       /* The limits (Only if *optknt == 0)
double  knt[];        /* knt[(*NumPoints)+(*deg)+1] : The Knot sequence
double  *Weights;     /* Weights[*NumPoints], The weights
double  *Points;      /* Points[(*dim)*(*NumPoints)]: The NURBS control points
/*
/* if dim=3: 3D points in MODEL (x0,y0,z0,x1,y1,z1,..)
/* if dim=2: 2D points in WORK  (x0,y0,x1,y1,..)
/*          The plane depth will be 0 (zero)
/*          The points (if dim=2 : 2D points in WORK
/*                  if dim=3 : 3D points in MODEL)
/*
/* Output:
/*
int    *Spline;       /* The B-spline ID
/* = -1 : Error !
int    *Status;       /* Status.
/*          See General Concepts, Chapter 1, Status.
/*- - - - - */

```

CDK_SPLINE_CUBIC

Create a cubic spline.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_SPLINE_CUBIC ( Dim, Points, Numpoints, Slopemode, Slopes, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Dim;          /* 2 = 2D-spline ; 3 = 3D-spline
/*
/* Output :
/*
double  Points[];     /* The coordinates of the through points.
/*
/* Input :
/*
int    *Numpoints;    /* The number of points.
int    *Slopemode;     /* Define the boundary conditions.
/*      0 = Free tangents at the end points.
/*      1 = Start tangent is given in Slopes.
/*      2 = End tangent is given in Slopes.
/*      3 = Both tangents are given.
double  Slopes[];     /* The vectors of Start & End slopes ( if SlopeMod 0 )
/* Dim = 2 :
/*      StartSlope = { Slopes[0], Slope[1] }
/*      EndSlope   = { Slopes[2], Slope[3] }
/*      The vectors are defined in Current Work Plane.
/* Dim = 3 :
/*      StartSlope = { Slopes[0], Slope[1], Slope[2] }
/*      EndSlope   = { Slopes[3], Slope[4], Slope[5] }
/*      The vectors are defined in Model UCS.
/*
/* Output :
/*
int    *Status;       /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/*
/* ID of the created entity.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SPLINE_FP

Create a B spline by defining its fairing points.

Syntax :

```

/*- - - - - */
int CDK_SPLINE_FP ( Dim, Depth, Tol, Numberpoints, Points, Slopemode, Slopes,
                  Status )
/*- - - - - */
/*
/* Input :
/*
int    *Dim;          /* 2 = 2D-spline ; 3 = 3D-spline
double *Depth;        /* The depth of the splineEs plane ( if Dim == 2 )
double *Tol;          /* The required tolerance.
int    *Numberpoints; /* The number of the fairing points.
double Points[];      /* The coordinates of the fairing points.( Points[Dim *
/* NumberPoints] )
int    *Slopemode;    /* Define the boundary conditions.
/*      0 = Free tangents at the end points.
/*      1 = 1st tangent is given in Slopes[0..Dim - 1].
/*      2 = 2nd tangent is given in Slopes[0..Dim - 1].
/*      3 = Both tangents are given Slopes[0..Dim*2 - 1].
double Slopes[6];     /* The tangents at the end points ( Unit vectors ).
/*
/* Output :
/*
int    *Status;       /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

CDK_SPLINE_NURBS_DATA

Get control points and knots of a a B-spline.

Syntax:

```

/*- - - - - */
int CDK_SPLINE_NURBS_DATA (id, tol, limcrv, ncp, ctrlPoints, knots, weights)
/*- - - - - */
/*
/* Input:
/*
int    id;           /* Curve Id
double tol ;         /* Distance tolerance
double limcrv[2] ;   /* (Ustart,Uend) - given limits
/* NULL = get limits from data base
int    ncp;          /* number of control points.
/*
/* Output:
/*
double ctrlPoints[]; /* buffer containing the control points.
double knots[];      /* Knots sequence.
double weights[];    /* The NURBS weights (may be NULL)
/*
/* Return value:
/* See General Concepts, Chapter 2, Status.
/*- - - - - */

```

Note: allocations that must be made:

- ctrlPoints -> [dim*ncp] (number of control points * dimension).
- Knots -> [ncp+ord] (number of control points + Order).
- weights -> [ncp]
- IN ORDER TO GET THE PARAMETERS, use function CDK_SPLINE_NURBS_SIZE.

CDK_SPLINE_NURBS_SIZE

Get parameters (dim, ncp, ord) of a NURBS spline, in order to know how much to allocate for knots and control points. Use before function CDK_SPLINE_NURBS_DATA.

Syntax:

```
/*- - - - - */
int CDK_SPLINE_NURBS_SIZE (id, tol, limcrv, dim, ncp, ord)
/*- - - - - */
/*
/* Input:
/*
/*
int    id;      /* Curve Id
double tol ;    /* Distance tolerance
double limcrv[2] ; /* (Ustart,Uend) - given limits
/* NULL = get limits from data base
/*
/* Output :
/*
int    *dim;    /* 2D/3D
int    *ncp;    /* number of control points.
int    *ord;    /* order of the spline (degree + 1).
/*
/* Return value:
/* See General Concepts, Chapter 2, Status.
/*- - - - - */
```

CDK_SPLINE_TP

Create a B spline by defining its through points.

Syntax :

```
/*- - - - - */
int CDK_SPLINE_TP ( Dim, Depth, Numberpoints, Points, Slopemode, Slopes,
/*- - - - - */
/*
/* Input :
/*
/*
int    *Dim;    /*      2 : 2D-spline
/*      3 : 3D-spline
double *Depth;  /* The depth of the splineEs plane ( if Dim == 2 )
int    *Numberpoints; /* The number of the through points.
double Points[]; /* The coordinates of the through points.( Points[Dim *
/* NumberPoints] )
int    *Slopemode; /* Define the boundary conditions.
/*      0 = Free tangents at the end points.
/*      1 = 1st tangent is given in Slopes[0..Dim - 1].
/*      2 = 2nd tangent is given in Slopes[0..Dim - 1].
/*      3 = Both tangents are given Slopes[0..Dim*2 - 1].
double Slopes[6]; /* The tangents at the end points ( Unit vectors ).
/*
/* Output :
/*
int    *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */
```

CLOSCV

Check whether a curve is closed and/or periodic.

Syntax :

```

/*- - - - - */
int CLOSCV ( Idc, Period )
/*- - - - - */
/*
/* Input :
/*
int    *Idc;    /* ID number of the curve.
/*
/* Output :
/*
int    *Period; /*    0 = not periodic.
/*                /*    1 = periodic.
/*
/* Returns :
/* .TRUE.
/* If curve is closed.
/* .FALSE.
/* If curve is open.
/*- - - - - */

```

GTCRPT

For a given point on the curve and a distance (by curve) from it to another point on the curve, get the coordinates and derivatives of the second point.

Syntax :

```

/*- - - - - */
void GTCRPT ( Idc, Idp, Dist, Pnt, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idc;    /* ID number of the curve.
int    *Idp;    /* ID number of the first point on the curve.
float  *Dist;    /* Distance between points ( can be of any sign ).
/*
/* Output :
/*
double Pnt[9];  /* "Pnt[0-2]"
/*                /* X, Y, Z coordinates of the second point.
/*                /* "Pnt[3-5]"
/*                /* Xu, Yu, Zu derivatives of the second point.
/*                /* "Pnt[6-8]"
/*                /* Xuu, Yuu, Zuu derivatives of the second point.
int    *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

GTCRPTD

Get coordinates and derivatives of the point on the curve, according to the distance (by curve) to the given curve's point.

Syntax :

```

/*- - - - - */
void GTCRPTD( IdCrv, IdPnt, Dist, Point, Status )
/*- - - - - */
/* Input: */
int *IdCrv; /* ID number of the curve */
int *IdPnt; /* ID number of the given point on the curve */
double *Dist; /* Distance between points ( can be of any sign ) */
/* Output: */
double Point[9]; /* PNT( 1-3 ) - X, Y, Z coordinates of the second point. */
/* PNT( 4-6 ) - Xu, Yu, Zu derivations in the second point */
/* PNT( 7-9 ) - Xuu, Yuu, Zuu derivations in the second point */
int *Status; /* See General Concepts-Status on page 1-2. */
/*- - - - - */

```

SPLNCR

Create a spline.

Syntax :

```

/*- - - - - */
int SPLNCR ( Mode, Ids, N, Idb1, Idp1, Idb2, Idp2, Status )
/*- - - - - */
/* Input : */
int *Mode; /* 2 = 2D. */
/* 3 = 3D. */
int Ids[N]; /* List of IDS of N points to be connected by a spline. */
int *N; /* Number of points ( up to 200 ). */
int *Idb1; /* ID of the 1st point defining the slope at the start of the spline. Set IDB1 to 0 to let the program define the slope. */
int *Idp1; /* ID of the 2nd point defining the slope at the start of the spline ( ignored when IDB1=0 ). */
int *Idb2; /* ID of the 2nd point defining the slope at the end of the spline. Set IDB2 to 0 to let the program define the slope. */
int *Idp2; /* ID of the 2nd point defining the slope at the end of the spline ( ignored when IDB2=0 ). */
/* Output : */
int *Status; /* See General Concepts-Status on page 1-2. */
/* Returns : */
/* ID of the created spline ( if STATUS=0 ). */
/*- - - - - */

```

UBSPLN

Create a B-spline entity.

Syntax :

```

/*- - - - - */
void UBSPLN ( Dim, Deg, Ocflag, Optper, Optwgh, Np, Wghs, Ptclds, Id, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Dim;      /*      2 = 2D-spline.
/*      3 = 3D-spline.
int    *Deg;      /* The degree of the spline ( 1 < DEG < 31 ).
int    *Ocflag;   /*      0 = Open.
/*      1 = Closed.
int    *Optper;   /*      0 = Non-Periodic.
/*      1 = Periodic.
/* OPTWGT
/*      0 = With weights.
/*      1 = No weights
/*     -1 = With weights ( but Wghs is not given yet ).
/*
/*
int    *Optwgh;   /*
/*
/* Input :
/*
int    *Np;       /* No. of control points.
double Wghs[Np];  /* The weights of control points.
double *Ptclds;   /* The coordinates of control points;
/* ( if dim=2 : 2D points in WORK if dim=3 : 3D points
/* in MODEL ).
/*
/* Output :
/*
int    *Id;       /* The B-spline ID.
/* If ID = -1 : The spline is not created.
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UCRBSP

Create a Bezier spline.

Syntax :

```

/*_ _ _ _ _ */
int UCRBSP ( Dim, Deg, Nseg, Smoothopt, Segdef, Seginx, Noinx, Cnp, Status ) */
/*_ _ _ _ _ */
/* Input : */
/* */
int *Dim; /* 2 = Create a 2D Bezier spline ( in WORK plane ). */
/* 3 = Create a 3D Bezier spline ( in MODEL system ). */
int *Deg; /* The degree of the Bezier segments ( 2 DEG 3l ). */
int *Nseg; /* The no. of Bezier segments. */
int *Smoothopt; /* 1 = The next direction will be the same as the */
/* previous direction. */
/* 2 = The previous direction will be redefined by */
/* the next direction. */
/* 3 = The previous & next directions will be the */
/* average direction. */
int *Segdef; /* Indicate default: */
/* 0 = break point, */
/* 1 = smooth. */
int *Seginx; /* Segments indices which differ from the default.Each */
/* segment index : 1 Seginx[i] NSEG-1Indices */
/* should be in increasing order. */
int *Noinx; /* No. of segments which differ from default. */
/* */
/* Input/Output : */
/* */
double *Cnp; /* The control points of the Bezier spline. */
/* If DIM=2: */
/* These points are given in the WORK */
/* ( XW1, YW1, XW2, YW2, .. ) */
/* If DIM=3: */
/* These points are given in the MODEL */
/* ( XM1, YM1, ZM1, XM2, YM2, ZM2, ... ) */
/* CNP size: */
/* DIM * ( NSEG * DEG + 1 ) */
/* */
/* Output : */
/* */
int *Status; /* See General Concepts-Status on page 1-2. */
/* Returns : */
/* ID of the spline ( if STATUS = 0 ). */
/*_ _ _ _ _ */

```

UCRVLN

Calculate the length of a curve.

Syntax :

```

/*- - - - - */
void UCRVLN ( Idc, Dim, Length, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID number of the curve.
/* 2 = Calculate the projected length on the WORK
/* plane.
/* 3 = Calculate the real length.
/*
/* Output :
/*
/* The curve length.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */
int *Idc;
int *Dim;
double *Length;
int *Status;

```

UEVCRV

Given a curve parameter, get the coordinates of a point on it and the derivatives of this point.

Syntax :

```

/*- - - - - */
void UEVCRV ( Idc, Opteva, System, Ud, Pnt, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID number of the curve.
/* Evaluation option:
/* if given parameter, defines break point,
/* æleft derivativeÆ ( for u <= UD )
/* with OPTEVA = 1
/* and æright derivativeÆ ( for u => UD )
/* with OPTEVA = 2.
/*
/* Coordinates system for evaluation:
/* 1 = MODEL
/* 2 = WORK
/*
/* Given parameter value of the curve.
/*
/* Output :
/*
/* "Pnt[0-2]"
/* X, Y, Z coordinates of point on the curve.
/* "Pnt[3-5]"
/* Xu, Yu, Zu derivatives of this point
/* "Pnt[6-8]"
/* Xu, Yu, Zu second derivatives of this point
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */
int *Idc;
int *Opteva;
int *System;
double *Ud;
double Pnt[9];
int *Status;

```

UGPARM

Given the ID of the curve and coordinates of the point, retrieve the parameter on the curve closest to indication.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UGPARM ( Idcrv, Point, U, Pmin, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Idcrv; /* ID of the curve.
double Point[3]; /* The coordinates of the point in MODEL.
/*
/* Output :
/*
double *U; /* Parameter on curve closest to indication.
double Pmin[3]; /* The coordinates of the point on the curve according
/* to found parameter.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UGSSRF

Get/Set refine factor of the spline.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UGSSRF ( Mode, Idcurve, Reffactor, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Mode; /* 1 = Get refine factor,
/* 2 = Set refine factor.
int *Idcurve; /* The ID of the curve.
double *Reffactor; /* The refine factor.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ULMCRD

Get curve limits from data base.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void ULMCRD ( Idc, Lim, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Idc; /* ID number of the curve.
/*
/* Output :
/*
double Lim[2]; /* Parameter limits : Ustart and Uend.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

USPLNP

Get number of control points

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void USPLNP ( Idcrv, Np, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*                                     */
int      *Idcrv;                /* ID of spline.
                                /*                                     */
                                /* Output :
                                /*                                     */
int      *Np;                   /* Number of control points.
int      *Status;               /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



SURFACE

CDK_BLEND_REGION

Create a trimmed surface given its outer loop as a 3D contour.

Syntax :

```

/*- - - - - */
int CDK_BLEND_REGION ( ConBuf, Opt, Tol, RefEnts, NumEnts, DspSec, DspXSec )
/*- - - - - */
/*
/* Input :
/*
int    ConBuf[];    /* Contour buffer.
int    Opt;          /* Plane option :
/* = 0 : Plane not defined.
/* = 1 : Work plane is reference plane.
double Tol;         /* Tolerance.
int    RefEnts[];   /* IDs Buffer of refence Entities (curves/points).
int    NumEnts;     /* The number of refence Entities.
int    DspSec;      /* The number of display sections.
int    DspXSec;     /* The number of display cross sections.
/* Returns :
/* > 0 : Surface ID,
/* < 0 : See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_BLEND_SECTIONS Create a blend section surface.
Syntax :

```

/*- - - - - */
int CDK_BLEND_SECTIONS ( SecIds, NumSec, BoundIds, NumBound, SlopeOpt1,
                        SlopeVec1, SlopeOpt2, SlopeVec2, Tol, DspSec, DspXSec )
/*- - - - - */

/*
/* Input :
/*
CDKT_BLEND_ID          /*
    SecIds[];          /* The sections IDs (check cdk_type for CDKT_BLEND_ID).
int    NumSec;          /* The Number of Sections.
CDKT_BLEND_ID          /*
    BoundIds[2];       /* The Boundary IDs.
int    NumBound;        /* The Number of Boundaries (0 / 1 / 2) .
int    SlopeOpt1;       /* Slope option for first section :
                        /*    0 : Free slope.
                        /*    1 : Constant.
                        /*    2 : Linear.
                        /*    3 : Surface.
                        /*    4 : Plane.
                        /*    5 : Surface - Normal
double  SlopeVec1[6];   /* Slope data for first section :
                        /*    For Constant Slope or SlopeVec[0-2] is a
                        /*    constant direction.
                        /*    For Linear Slope SlopeVec[0-2] is the direction
                        /*    of slope of the first edge point of the section
                        /*    ( closest to SecIds[.pt] ).
                        /*    SlopeVec[3-5] is the direction of slope of the
                        /*    second edge point of section.
                        /*    For Plane SlopeVec[0-2] is a plane normal which
                        /*    defines a constant direction.
int    SlopeOpt2;       /* Slope options for last section :
                        /*    Same as SlopeOpt1.
double  SlopeVec2[6];   /* Slope data for last section :
                        /*    Same as SlopeVec1.
double  Tol;            /* Tolerance.
int    DspSec;          /* The number of display sections.
int    DspXSec;         /* The number of display cross sections.
                        /* Returns :
                        /*    > 0 : Surface ID,
                        /*    < 0 : See General Concepts-Status on page 1-2.
                        /*    -2 : May not satisfy tolerance.
                        /*    -3 : Inconsistent input.
/*- - - - - */

```

CDK_PLN_INTERSEC Calculate the intersection points of the plane and curve.
Syntax :

```

/*- - - - - */
int CDK_PLN_INTERSEC( idc, tol, Coef, maxnp, np, par )
/*- - - - - */

/*
/* Input :
/*
int    idc;             /* The curve ID.
double tol;             /* The tolerance for the plane-curve intersection.
double Coef[4];         /* The plane coefficients ( WORK ).
int    maxnp;           /* Maximum available number of intersection points.
                        /*
                        /* Output :
                        /*
int    *np;             /* No. of intersection points.
double par[];           /* The intersection parameters on the curve.
/*- - - - - */

```

CDK_SLA_CREATE

Create an SLA surface.

Syntax :

```

/* - - - - - */
int CDK_SLA_CREATE( Status )
/* - - - - - */
/*
/* Output :
/*
int *Status;
/* 0 = OK, other = ERROR.
/*
/* Result: The ID of the created entity.
/* - - - - - */

```

CDK_SLA_EXIT

Close the CDK_SLA package and clear allocated memory.

Syntax :

```
/*- - - - - */
void CDK_SLA_EXIT( )
/*- - - - - */
```

CDK SLA FACE ADD

Create an SLA face.

Syntax :

```

/*- - - - - */
int   CDK_SLA_FACE_ADD( Vertices )
/*- - - - - */
/*
/* Input :
/*
int   Vertices[3]; /* The indices of the vertices define the face
/* and the normal ( clockwise sequence )
/*
/* Result: 0 = OK, other = ERROR
/*- - - - - */

```

CDK SLA INI

Activate the CDK SLA package.

Syntax :

```

/* - - - - - */
void    CDK_SLA_INI( )
/* - - - - - */

```

CDK SLA NODE ADD

Create the SLA node.

Syntax :

```

/* - - - - - */
int  CDK_SLA_NODE_ADD( NodeCoor )
/* - - - - - */
/* Input:
float  NodeCoor[3];  /* The coordinates of the node ( MODEL )
/*
/*
/* Returns: The index of created node
/* - - - - - */

```

CDK_SRF_BOX

Calculate a surface enclosing box in the MODEL coordinate system.

Syntax :

```
/*- - - - - */
int CDK_SRF_BOX( Id, Sys, Box )
/*- - - - - */
/*
/* Input :
/*
int      Id;      /* Surface ID.
int      Sys;     /* = 1 : Model coordinate system.
               /* = 2 : Work coordinate system.
               /*
               /* Output :
               /*
double   Box[6];  /* { Xmin, Ymin, Zmin, Xmax, Ymax, Zmax }.
               /*
               /*
               /* Returns :
               /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

CDK_SRF_CRPIP

Create a NURBS surface as in the following DRIVE options of the Modeling application:

DRIVE >> SPINE >> ONE SECTION
 DRIVE >> SPINE >> TWO SECTIONS
 DRIVE >> SPINE & PLANE

Syntax :

```
int CDK_SRF_CRPIP( optpln, tol, idcrv, optcrv, dspsec, dspdrv, status )
/*- - - - - */
/*
/* Input :
/*
int      optpln;   /* 1 = no definition of plane.
               /* 2 = define plane.
double   tol;     /* A given tolerance for approximation.
int      idcrv[3]; /* idcrv[0]: The ID of the 1st section curve.
               /* idcrv[1]: The ID of the 2nd section curve.
               /* If idcrv[1] = 0 : Only one section !
               /* idcrv[2]: The ID of the drive curve.
int      optcrv[3]; /* optcrv[0]: Define the 1st section curve limits.
               /* optcrv[1]: Define the 2nd section curve limits.
               /* optcrv[2]: Define the drive curve limits.
               /* The meaning of optcrv according to its value :
               /* 1 = The start & end parameters defined in limcrv.
               /* 2 = The start & end parameters are the start &
               /* end parameters of the curve in the data base.
               /* 3 = The start & end parameters are the end &
               /* start parameters of the curve in the data
               /* Base.
int      dspsec;   /* No. of displayed section curves.
int      dspdrv;   /* No. of displayed drive curves.
               /*
               /* Output :
               /*
int      *status;  /* = 0 = O.K.
               /* < 0 = See General Concepts-Status on page 1-2.
               /*
               /* Returns:
               /* ID of the created surface if status = 0.
               /* -1 otherwise.
/*- - - - - */
```

CDK_SRF_DRPRL

Create a drive surface by moving section contour along path defined by a drive contour so that it is always parallel to itself.

Syntax :

```

/*- - - - - */
int CDK_SRF_DRPRL( ConDrv, ConSec, DspDrv, DspSec )
/*- - - - - */
/*
/* Input :
/*
int *ConDrv; /* Buffer of contour of drive.
int *ConSec; /* Buffer of contour of section.
int DspDrv; /* No. of display curves that will be in the
/* direction of the drive curve.
int DspSec; /* No. of display curves that will be in the
/* direction of the section curve.
/* Returns:
/* > 0 = Entity ID.
/* < 0 = See General Concepts-Status on page 1-2.
/*- - - - - */
Note : - Contour must be smooth.

```

CDK_SRF_FAIR

Execution of fairing per surface.

Syntax :

```

/*- - - - - */
int CDK_SRF_FAIR ( Origsurface, Tolboundary, Tolsurface, Attrmode, Slopemode,
/*- - - - - */
/* Status )
/*
/* Input :
/*
int *Origsurface; /* ID of the original surface.
double *Tolboundary; /* Tolerance required for approximating the boundaries
/* of the surface.
double *Tolsurface; /* The required tolerance for approximating the
/* internal surface.
int *Attrmode; /* 1 = As active,
/* 2 = As original.
int *Slopemode; /* 0 = Free slopes,
/* 1 = Keep slopes of original surface.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* The ID of the created surface.
/*- - - - - */
Note : - TolSurface must be TolBoundary.

```

CDK_SRF_INTERSEC

Calculate the intersection points surface & curve.

Syntax :

```

/*- - - - - */
int CDK_SRF_INTERSEC( idsrf, idc, maxnp, tol, np, par, uv )
/*- - - - - */
/*
/* Input :
/*
/* The curve ID.
/* The surface ID.
/* Maximum available number of intersection points.
/* The tolerance for the surface-curve intersection.
/*
/* Output :
/*
/* No. of intersection points.
/* par[*np] The intersection parameters on the curve.
/* uv[*2*np] The intersection parameters on surface.
/*
/* Returns:
/* Status ( See General Concepts-Status on page 1-2 ).
/*
/*- - - - - */

```

CDK_SRF_INTERSECTION

Calculating the GLOBAL OR LOCAL intersection between two surfaces.

Syntax :

```

/*- - - - - */
void CDK_SRF_INTERSECTION (Glb_loc_mod,Uv1,Uv2,Tol,Srf1,Srf2,MaxNumCurves,Curves,NumC
urves)
/*- - - - - */
/*
/* Input :
/*
/* = 0 : Local intersection.
/* = 1 : Global intersection.
/*
/* The indicated u,v on the 1st surface (for LOCAL)
/*
/* The indicated u,v on the 2nd surface (for LOCAL)
/*
/* A given tolerance.
/*
/* The 1st surface ID
/*
/* The 2nd surface ID
/*
/* Maximum number of curves to create.
/*
/* Output :
/*
/* IDs of created curves (Curves[MaxNumCurves]).
/*
/* Number of created curves.
/*
/* >= 0 : Number of intersections found.
/*
/* < 0 : Error.
/*
/*
/*- - - - - */

```

CDK	SRF	MODIFY	LIM	Modify surface limits.
------------	------------	---------------	------------	------------------------

Syntax:

```

/* - - - - - */
int CDK_SRF_MODIFY_LIM ( srfID, lim )
/* - - - - - */
int      srfID;      /* I  : Surface ID. */
double   lim[4];     /* I  : New surface limits. */
/* - - - - - */
/* R  : Status. */
/* - - - - - */

```

CDK SRF MULTY DRIVE Create a multiple drive surface.

Syntax :

```

/*- - - - - */
int CDK_SRF_MULTY_DRIVE( Dim, Tol, OptPln, NrmPln, Nsec, IdSec, IdDrv, DspSec,
                        DspDrv, DirSec )
/*- - - - - */

/*
/* Input :
/*
int      Dim;      /* Drive contour dimension ( 2D or 3D ).
double   Tol;      /* Tolerance for section approximation.
int      OptPln;    /* = 0 : Do not define plane. = 1 : Define plane.
double   NrmPln[3]; /* Normal to plane ( if OptPln = 1 ).
int      Nsec;      /* Number of sections ( MAX 32 ).
int      IdSec[];   /* IDs of section curves.   Size = nsec.
int      IdDrv[];   /* IDs of drive   curves.   Size = nsec-1.
int      DirSec[];  /* Direction of section curves : Size = nsec.
int      DspSec;    /* No. of displayed section curves.
int      DspDrv;    /* No. of displayed drive curves.
/* = 1 : The start & end parameters are the START & END
/*          parameters of the curve in the data base.
/* = -1: The start & end parameters are the END & START
/*          parameters of the curve in the data base.
/* Returns :
/* > 0 : ID of created surface.
/* < 0 : See General Concepts-Status on page 1-2.
/* = -( 50+n )   Two identical IDs were found in IdDrv.
/*              ( n == index of the second one ).
/* = -( 100+n )  Drive curve is not smooth.
/*              ( n == index of ID of curve in IdDrv ).
/* = -( 150+n )  Two sequential drive curves do not
/*              coincide.
/*              ( n == index of ID of the second one ).
/* = -( 200+n )  Two sequential drive curves are not
/*              tangent.
/*              ( n == index of ID of the second one ).
/* = -( 250+n )  One of section curves has an identical
/*              ID as one of the drive curves.
/*              ( n == index of ID of section curve ).
/* = -( 300+n )  All sections which are composite curves
/*              must have the same number of curves and
/*              one of them did not.
/*              ( n == index of ID of section curve ).
/*- - - - - */

```

CDK_SRF_NORMAL

Get the normal of a surface at an indicated point.

Syntax :

```

/*- - - - - */
int CDK_SRF_NORMAL( IdSrf, Sys, Coord, Iuv, Nor )
/*- - - - - */
/*
/* Input :
/*
/* Surface ID.
/* Coordinates system :
/* = 1 : MODEL
/* = 2 : WORK
/* Coordinates of a point on the surface.
/* The U & V parameters on the surface
/* Note : If iuv == NULL then U & V are obtained from
/* the Data Base.
/* If iuv != NULL the coord is neglected, and
/* can be input as NULL.
/*
/* Output :
/*
/* The surface normal ( which is a normalized vector ).
/* Return :
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```

CDK_SRF_REFINE

Get/Set the refine factor of a surface.

Syntax :

```

/*- - - - - */
void CDK_SRF_REFINE( GetSet, IdSrf, RefFactorU, RefFactorV, Status )
/*- - - - - */
/*
/* Input :
/*
/* 1 = Get refine factor. 2 = Set refine factor.
/* The surface ID
/*
/* Input / Output :
/*
/* The refine factor in U direction ( parametric value )
/* The refine factor in V direction ( parametric value )
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```

CDK_SRF_RULCS

Create a draft-angle ruled surface between a given contour and a surface.

Syntax :

```

/*- - - - - */
void CDK_SRF_RULCS ( Contbuffer, Surface, Direction, Refpoint, ToleranceAngle,
                    Trimopt, Srfopt, Numu, Numv, Numruleds, Ruleds, Status )
/*- - - - - */
/*
/* Input :
/*
int    Contbuffer[]; /* Buffer of contour data.
/*
/* Input/Output :
/*
int    *Surface;     /* ID of given surface.
/* The ID of trimmed curve ( if TrimOpt is 1 ).
/*
/* Input :
/*
double Direction[3]; /* The projection direction ( MODEL ).
double Refpoint[3]; /* The point that indicates the projection of the
/* contour on the surface.
/*
double *Tolerance;   /* A given tolerance.
double *Angle;       /* Draft angle from direction ( RADIAN ).
int    *Trimopt;     /* 1 = Trim the given surface,
/* 0 = not trim.
int    *Srfopt;      /* 1 = Create single surface,
/* 2 = multy surface.
int    *Numu;        /* The number of display lines in U direction.
int    *Numv;        /* The number of display lines in V direction.
/* NumRuleds
/* The expected number of ruled surfaces to be created.
/*
/* Output :
/*
int    *Numruleds;   /* The number of created ruled-surfaces.
int    Ruleds[];     /* The array of IDs of created ruled-surfaces.
int    *Status;      /* See General Concepts-Status on page 1-2.
/* 1 = Nothing was projected.
/*
/*- - - - - */

```

CDK_SRF_SECWPL

Create the splines which are intersections between a surface and a work plane.

Syntax :

```

/*- - - - - */
void CDK_SRF_SECWPL ( Tol, Idsrf, Numsections, Sections, Status )
/*- - - - - */
/*
/* Input :
/*
double *Tol;         /* A given tolerance.
int    *Idsrf;       /* The surface ID.
int    *Numsections; /* Max. number of sections.
/* NumSections
/* Number of created sections.
/*
/* Output :
/*
int    Sections[];   /* Array of sections ( Sections[NumSections] ).
int    *Status;      /* See General Concepts-Status on page 1-2.
/* -33 = Max. NumSections too small,
/* 1 = IdSrf is on the plane.
/*
/*- - - - - */

```

CDK_SRF_TRM_CON

Trim a surface/trimmed surface by a contour lying on it.

Syntax :

```

/*- - - - - */
void CDK_SRF_TRM_CON( OptDiv, Tol, RefPar, Surface, ContBuf, NumTrs, TrimSrf,
                    Status )
/*- - - - - */

/*
/* Input :
/*
int    *OptDiv;    /* 1 = Trim.      2 = Divide.
double *Tol;       /* The tolerance for approximation
double  RefPar[2]; /* The U & V of the indicated point on the remaining
/* side of the surface ( only for optdiv = 1 ).
int    *Surface;   /* The surface/trimmed surface ID.
int    *ContBuf;   /* The buffer of the trimming contour.
/*
/* Input / Output :
/*
int    *NumTrs;    /* Size of array "TrimSrf" / The number of created
/* Ids.
/*
/* Output :
/*
int    *TrimSrf;   /* The array of IDs of the new surfaces.
int    *Status;    /* Error flag:
/* 0 = O.K.
/* 1 = Not all surfaces were created.
/* 2 = Size of "TrimSrf" too small,
/* only part of surfaces were created,
/* "NumTrs" returns the number of surfaces,
/* that it is possible to create.
/* -1= Wrong input data
/* -3= Trimming algorithm failed. No surface created.
/*
/*- - - - - */

```

CDK SRF TRM PAR

Trim a surface/trimmed surface by a parameter.

Syntax :

```

/* - - - - - */
void CDK_SRF_TRM_PAR( OptDiv, RefPar, Surface, ConstPar, TrimPar, NumTrs,
                     TrimSrf, Status )
/* - - - - - */

/*
/* Input :
/*
int    *OptDiv;
/* 1 = Trim.      2 = Divide.
double  RefPar[2];
/* The U & V of the indicated point on the remaining
/* side of the surface ( only for optdiv = 1 ).
int    *Surface;
/* The surface/trimmed surface ID.
int    *ConstPar;
/* 1 : U = Const.      2 : V = Const.
double *TrimPar;
/* The trimming parameter
/*
/* Input / Output :
/*
int    *NumTrs;
/* Size of array "TrimSrf" / The number of created
/* Ids.
/*
/* Output :
/*
int    *TrimSrf;
/* The array of IDs of the new surfaces.
int    *Status;
/* Error flag:
/* 0 = : O.K.
/* 1 = : Not all surfaces were created.
/* 2 = : Size of "TrimSrf" too small,
/*        only part of surfaces were created,
/*        "NumTrs" return the number of surfaces,
/*        that possible to create.
/* -1= : Wrong input data
/* -3= : Trimming algorithm failed. No surface created.
/* - - - - - */

```

CDK_SRF_TRM_PCON Trim a surface/trimmed surface by a projected contour.
Syntax :

```

/*- - - - - */
void CDK_SRF_TRM_PCON( OptDiv, Tol, RefSide, RefProj, Surface, ContBuf, DirProj,
                      Angle, NumTrs, TrimSrf, Status )
/*- - - - - */

/*
/* Input:
/*
int    *OptDiv;    /* 1 = Trim,    2 = Divide.
int    *Tol;       /* Tolerance for approximation
double RefSide[2]; /* U & V parameters indicating point on the remaining
/* side of the surface ( only for OptDiv = 1 ).
double RefProj[2]; /* U & V parameters indicated point of projection.
int    *Surface;   /* The surface/trimmed surface ID.
int    *ConBuf;    /* Contour buffer.
double DirProj[3]; /* Project Direction.
double *Angle;     /* Project angle.
int    *NumTrs     /* Size of array "TrimSrf" / The number of created
/* Ids.
/*
/* Output:
/*
int    *TrimSrf;   /* ID of new surface( s )/trimmed surface( s ).
int    *Status;    /* Error flag:
/* 0 =    O.K
/* 1 =    Intersection was found, not all surfaces
/* were created ( divide option ).
/* 2 =    Size of "TrimSrf" is too small, only part
/* of surfaces were created."NumTrs" indicates
/* it this number.
/* -1=    Wrong input data.
/* -3=    Trimming Algorithm failed. No surface
/* Created.
/*- - - - - */

```

CDK_SRF_TRM_SRF

Trim a surface/trimmed surface by a surface.

Syntax :

```

/*- - - - - */
void CDK_SRF_TRM_SRF( OptDiv, OptBoth, Tol, RefPar1, RefPar2, Surfacel,
                      Surface2, NumTrs, TrimSrf, Status )
/*- - - - - */

/*
/* Input :
/*
int    *OptDiv;      /* 1 = Trim.      2 = Divide.
int    *OptBoth;     /* 1 = Trim only 1st surface.
                      /* 2 = Trim both surfaces.
double *Tol;         /* The tolerance for approximation.
double  RefPar1[2];  /* The U & V of the indicated point on the remaining
                      /* side of the surface to be trimmed ( only for
                      /* OptDiv = 1 ).
double  RefPar2[2];  /* The U & V of the indicated point on the remaining
                      /* side of the trimming surface.
                      /* ( only for OptDiv = 1 and OptBoth = 2 )
int     *Surfacel;   /* The surface/trimmed surface ID.
int     *Surface2;   /* The trimming surface ID.
/*
/* Input / Output :
/*
int     *NumTrs;     /* Size of array "TrimSrf" / The number of created
                      /* Ids.
/*
/* Output :
/*
int     *TrimSrf;    /* The array of IDs of the new surfaces.
int     *Status;     /* Error flag:
                      /* 0 = O.K.
                      /* 1 = Not all surfaces were created.
                      /* 2 = Size of "TrimSrf" is too small,
                      /*    only part of surfaces were created,
                      /*    "NumTrs" return the number of surfaces
                      /*    that possible to be created.
                      /* -1= Wrong input data.
                      /* -3= Trimming algorithm failed. No surface created.
/*- - - - - */

```

CDK_SRF_TRM_SRF1

Trim a base surface by multi surfaces.

Syntax:

```

/*- - - - - */
void cdk_srf_trm_srf1 ( OptDev, OptBoth, tol, RefPar1, RefPar2, idTrmd,
/*- - - - - */
                        TrmSrfBuf, numU, numV, NumTrs, TrimSrf, Status )
/*- - - - - */
/*
/* Input :
/*
int *OptDev; /* =1 : Trim ; =2 : Divide
int *OptBoth; /* =1 : trim only 1st surface
/* =2 : trim both surfaces
double *tol; /* The tolerance for approximation
/*
double RefPar1[2]; /* The uv of the indicated point on the remaining side
/* of the surface to be trimmed (only for OptDev = 1)
double RefPar2[]; /* The uv of the indicated point on
/* the remaining side of the trimming surfaces.
/* (only for OptDev = 1 and &OptBoth = 2 )
int *idTrmd; /* The surface/trimmed surface ID
int TrmSrfBuf[]; /* The trimming surfaces IDs buffer.
int numV; /* Num of displayed U lines
int numU; /* Num' of displayed U lines
/*
/* Input / Output :
/*
int *NumTrs; /* Size of array "TrimSrf" / The number of created IDs.
/*
/* Output:
/*
int *TrimSrf; /* The array of IDs of the new surfaces.
int *Status; /* Error flag
/* = 0 : o.k.
/* = 1 : not all surfaces were created
/* = 2 : size of "TrimSrf" too small,
/* created only part of surfaces,
/* "NumTrs" return the number of surfaces,
/* that were possible to create.
/* = -1 : wrong input data
/* = -2 : intersection was not found
/* = -3 : trimming algo. Failed; no surface created
/*- - - - - */

```

CDK_SRF_TRM_WPL

Trim a surface/trimmed surface by the work plane.

Syntax :

```

/*- - - - - */
void CDK_SRF_TRM_WPL( OptDiv, Tol, RefPar, Surface, NumTrs, TrimSrf, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int *OptDiv; /* 1 = Trim. 2 = Divide.
double *Tol; /* The tolerance for approximation
double RefPar[2]; /* The U & V of the indicated point on the remaining
/* SIDE of the surface ( only for optdiv = 1 ).
int *Surface; /* The surface/trimmed surface ID.
/*
/* Input / Output :
/*
int *NumTrs; /* Size of array "TrimSrf" / The number of created
/* Ids.
/*
/* Output :
/*
int *TrimSrf; /* The array of IDs of the new surfaces.
int *Status; /* Error flag:
/* 0 = : O.K.
/* 1 = : Not all surfaces were created.
/* 2 = : Size of "TrimSrf" is too small,
/* only part of surfaces were created,
/* "NumTrs" return the number of surfaces,
/* that possible to create.
/* -1= : wrong input data
/* -3= : trimming algorithm failed .no surface
/* Created.
/*- - - - - */

```

CDK SURFACE CREATE

Create a NURBS surface in the DataBase.

Syntax :

```

/*_ _ _ _ _ */
int CDK_SURFACE_CREATE (props, knt_u, knt_v, limsrfs, pts, wgh, idsrf) /*_
/*_ _ _ _ _ */
/* Input: */
/*_
int props[9]; /* The NURBS surface properties */
/* props[0]= ncp_u : The upper index of U control pts */
/* props[1]= ncp_v : The upper index of V control pts */
/* props[2]= deg_u : The degree in U direction */
/* props[3]= deg_v : The degree in V direction */
/* props[4]= clos_u : =0: Open in U ; =1: Closed */
/* props[5]= clos_v : =0: Open in V ; =1: Closed */
/* props[6]= weight : =0: With weights ; =1: No weight */
/* props[7]= per_u : =0: U Non-periodic; =1: Periodic */
/* props[8]= per_v : =0: V Non-Periodic; =1: Periodic */
/*_
double knt_u[]; /* The surface knot sequence in U direction */
/* Note : If knt_u= NULL : Only space will be reserved */
/*_
/* IN ANY CASE : The knt_u in the NURBS data-base will */
/* start from 0 ! (By subtracting knt_u[0] from knt_u) */
/*_
double knt_v[]; /* The surface knot sequence in V direction */
/* Note : If knt_v= NULL : Only space will be reserved */
/*_
/* IN ANY CASE : The knt_u in the NURBS data-base will */
/* start from 0 ! (By subtracting knt_v[0] from knt_v) */
/*_
double limsrfs[6]; /* The surface limits : (U_start, Delta_U, Step_U, */
/* V_start, Delta_V, Step_V) */
/*_
/* Note : If limsrfs=NULL : Default will be used */
/*_
double pts[]; /* The NURBS surface control points */
/* Note : If pts = NULL : Only space will be reserved */
/*_
double wgh[]; /* The NURBS surface control points weights */
/* Note : If wgh = NULL and props[6]=0 (With weight) */
/* A default of weigths will be set (all of */
/* them will be 1) */
/*_
/* Output : */
/*_
int *idsrf; /* The NURBS surface ID */
/*_
/* Return value: */
/* See General Concepts, Chapter 1, Status. */
/*_ _ _ _ _ */

```

CDK_SURFACE_NURBS_DATA

Get control points and knots of a NURBS surface.

Syntax:

```
/*- - - - - */
int CDK_SURFACE_NURBS_DATA (idsrf, ord_u, ord_v, ncp_u, ncp_v,
                           knt_u, knt_v, pts, wgh)
/*- - - - - */
/*
/* Input:
/*
int    idsrf;    /* id of the NURBS surface.
int    ord_u;    /* order of U.
int    ord_v;    /* order of V.
int    ncp_u;    /* number of control points in U direction.
int    ncp_v;    /* number of control points in V direction.
/*
/* Output:
/*
double knt_u[];  /* knt_u[ncp_u+ord_u] : Knots sequence in U direction
double knt_v[];  /* knt_v[ncp_v+ord_v] : Knots sequence in V direction
double pts[];    /* pts[3*ncp_u*ncp_v] : The NURBS control points
/*
/* Input\Output:
/*
double wgh[];    /* Input: if known that there are no weights enter NULL
/* otherwise:
/* Output: allocate wgh[ncp_u*ncp_v], if there are no
/* weights : The NURBS weights will be all '0'.
/* if there are weights they will be returned.
/*
/* Return value:
/* See General Concepts, Chapter 1, Status.
/*- - - - - */
Note: to obtain ord_u, ord_v, ncp_u, and ncp_v, which is also essential for knowing
how much memory must be allocated for knt_u[], knt_v[], pts[] and wgh[], use function
CDK_SURFACE_NURBS_SIZE. Please allocate enough room.
```

CDK_SURFACE_NURBS_SIZE Get ord_u, ord_v, ncp_u, ncp_v of a NURBS surface in order to know how much to allocate for knots and control points.
Use before function CDK_SURFACE_NURBS_DATA.

Syntax:

```
/*- - - - - */
int CDK_SURFACE_NURBS_SIZE (idsrf, ord_u, ord_v, ncp_u, ncp_v)
/*- - - - - */
/*
/* Input:
/*
int    idsrf;    /* id of the NURBS surface.
/*
/* Output:
/*
int    *ord_u;   /* order of U.
int    *ord_v;   /* order of V.
int    *ncp_u;   /* number of control points in U direction.
int    *ncp_v;   /* number of control points in V direction.
/*
/* Return value:
/* See General Concepts, Chapter 1, Status.
/*- - - - - */
```

CPTPCS

Calculate the closest point to an iso-parametric curve surface.

Syntax :

```

/*- - - - - */
void CPTPCS ( optpar, parc, limpar, bufsrf, pt, pt_cls, par_cls, ierflg )
/*- - - - - */
/*
/* Input :
/*
int      optpar;      /* 0 = Parameter U is constant.
/* 1 = Parameter V is constant.
double   parc;        /* The constant parameter.
double   limpar[2];   /* Start & End parameters of varying parameter.
T_BUFSRF *bufsrf;     /* The surface buffer.
double   pt[3];       /* A given 3D point in the MODEL system.
/*
/* Output :
/*
double   pt_cls[3];   /* The closest point to "pt".
double   *par_cls;    /* The parameter of this point.
int      *ierflg;     /* Error flag.
/*      0 = O.K.
/*     < 0 = Error.
/*- - - - - */

```

DBDSBX

Get/Set the enclosing box of a surface.

Syntax :

```

/*- - - - - */
void DBDSBX ( getset, ids, box, ierflg )
/*- - - - - */
/*
/* Input :
/*
int      *getset ;    /* =1: Get ; =2: Set;
int      *ids      ;  /* The surface ID
/*
/* Input / Output :
/*
double   box[6] ;     /* box( 1-3 ) : ( X_min , Y_min , Z_min )
/*                ( 4-6 ) : ( X_max , Y_max , Z_max )
/*                in MODEL coord.
/*
/* Output :
/*
int      *ierflg ;    /* Error flag:
/*      0 = O.K.
/*     -1= Error
/*- - - - - */

```

DRPARL

Create a drive surface by moving a section curve along a path defined by a drive curve so that it is always parallel to itself.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DRPARL ( Drivid, Select, Sectid, Dspdrv, Dspsec, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Drivid;    /* ID of the drive curve.
char    *Select;    /* Drive motion direction. ( See Chapter 1, General
/* Concepts. )
int    *Sectid;    /* ID of the section curve.
int    *Dspdrv;    /* Number of display curves that will be in the
/* direction of the drive curve.
int    *Dspsec;    /* Number of display curves that will be in the
/* direction of the section curve.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

PRDRSR

Project a point on a surface at a given direction.

Syntax :

PRDRSR Project a point on a surface at a given direction.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void PRDRSR ( Tol, Pt, Dir, Ids, Ptsrf, Uvpt, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double *Tol;    /* A given tolerance.
double Pt[3];    /* The point to be projected ( in MODEL system ).
double Dir[3];    /* The projection direction ( in MODEL system ).
int    *Ids;    /* ID number of the surface.
/*
/* Output :
/*
double Ptsrf[3];    /* The projected point on the surface.
double Uvpt[2];    /* The parameters of the projected point.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

PRNRSR

Project points on a surface in the normal direction.

Syntax :

```

/*- - - - - */
void PRNRSR ( Optout, Ptbuf, Np, Ids, Ptsrf, Uvpt, Ptflag, Status )
/*- - - - - */
/*
/* Input :
/*
int *Optout; /* 1 - Points only.
/* 2 - Parameters only.
/* 3 - Points and parameters.
/*
/* Output :
/*
double *Ptbuf; /* Point coordinates to be projected (in MODEL system).
int *Np; /* Number of points.
/*
/* Input :
/*
int *Ids; /* ID number of the surface.
/*
/* Output :
/*
double *Ptsrf; /* Coordinates of the projected points on the surface.
double *Uvpt; /* The parameters of the projected points.
int Ptflag[Np]; /* Flag per point:
/* 1 - Projection was not found.
/* 0 - Projection O.K.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

REVOLS

Create a surface of revolution by rotating a specified curve around an axis.

Syntax :

```

/*- - - - - */
int REVOLS ( Crvid, Axstrt, Axsend, Rotang, Dspdrv, Dspsec, Status )
/*- - - - - */
/*
/* Input :
/*
int *Crvid; /* ID of the curve to be rotated.
float Axstrt[3]; /* Coordinates of the start point of the rotation axis.
float Axsend[3]; /* Point coordinates defining direction of the rotation
/* axis.
/*
float *Rotang; /* Counter-clockwise rotation angle in degrees.
int *Dspdrv; /* Number of display curves that will be in the
/* direction of the drive curve.
int *Dspsec; /* Number of display curves that will be in the
/* direction of the section curve.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - */

```

Note : - Axstrt and Axsend are given in coordinates of the work system.

RULEDS

Create a ruled surface by joining the corresponding points on two curves by a set of straight line segments.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int RULEDS ( Crvid1, Selct1, Crvid2, Selct2, Dsplin, Dspcrv, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Crvid1;    /* ID of the first curve to be used to create the ruled */
/* surface.
char    *Selct1;    /* Indicate the direction of the first curve. See the */
/* explanation of SELECT in Chapter 2, General Concepts.
int    *Crvid2;    /* ID of the second curve to be used to create the */
/* ruled surface.
char    *Selct2;    /* Indicate the direction of the second curve. See the */
/* explanation of SELECT in Chapter 2, General Concepts.
int    *Dsplin;    /* Number of straight lines in one direction that will */
/* be used to display the ruled surface.
int    *Dspcrv;    /* Number of straight lines in the other direction that */
/* will be used to display the ruled surface.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

SRFPTD

Find the picked point indication parameters (uc, vc) on a surface/trimmed surface, immediately after the picking.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void SRFPTD ( Optpt, Optc, Idsrf, Ptm, Uc, Vc, Ierflg )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Optpt;    /* 1 = The parameters define any point on the surface
/* 2 = The parameters will define any point on the
/*      surface boundaries
/* 3 = The parameters will define any point on the
/*      surface corners
/* 4 = The parameters will define any point on the
/*      surface inters. display points.
int    *Optc;    /* Currently this option is not used ( old parameter )
/* =1 : Rough search
/* =2 : Rough search + refinement
int    *Idsrf;    /* The trimmed surface ID
/*
/* Input / Output :
/*
float   Ptm[3];    /* The given point in MODEL
/* As input : Should be given only for trimmed
/* Surface.
/* As output: The selected point in the MODEL coord.
/*
/* Output :
/*
float   *Uc;    /* The parameters on the surface
float   *Vc;    /*
int     *Ierflg; /* 0 = O.k.
/* -1= Error !
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Remarks :

1. The parameter **Optc** (Rough search/Rough search + refinement) is not used (so has no influence). Just give **Optc** a value of zero (0).
2. Routines SLPOP3(ID, IX, IY, DFPT), SLTOP3(ID, IX, IY, DFPT) return the picking point (in 3D display file coord. = MODEL)
3. The point **Ptm** is actually needed as input ONLY if **Idsrf** is an ID of a trimmed surface (if **Idsrf** is an ID of a surface it does not have to be initialized). Anyway, as output it always returns the picking point in 3D display coord. system (= MODEL system).
4. The specific edge closest to the picking point may be derived from the (u, v) indication point :

```

scl1 = fabs( u-umin )/( umax-umin )
scl2 = fabs( u-umax )/( umax-umin )
scl3 = fabs( v-vmin )/( vmax-vmin )
scl4 = fabs( v-vmax )/( vmax-vmin )

```

The min. value out of {scl1, scl2, scl3, scl4} will define the boundary as {u=umin; u=umax; v=vmin; v=vmax} respectively.

TRSGID

Get the surface ID on which the Trimmed surface is defined.

Syntax :

```

/*- - - - - */
void TRSGID ( IdTrs, IdSrf )
/*- - - - - */
/*
/* Input:
/*
int    *IdTrs;    /* The ID of the Trimmed surface ( TRS ).
/*
/* Output:
/*
int    *IdSrf     /* The surface ID.
/* -1 = Error ( This TRS does not exist ) !
/*- - - - - */

```

TRSGSZ

Get the number of polygons and the UV polygon size of a trimmed surface.

Syntax :

```

/*- - - - - */
void TRSGSZ ( Idtrs, Npol, Polsiz )
/*- - - - - */
/*
/* Input :
/*
int    *Idtrs;    /* The ID of the Trimmed surface.
/*
/* Output :
/*
int    *Npol;     /* The number of polygons in this trimmed surface.
int    *Polsiz;   /* The polygons size ( -1 = Error ).
/*- - - - - */
Note :    - Use this subroutine before using UGTRPO.

```

TSD_BOX

Get/Set the parametric/geometric enclosing box.

Syntax :

```

/*- - - - - */
void TSD_BOX( idtrs, get_set, opt, box, ierflg )
/*- - - - - */
/*
/* Input :
/*
int    idtrs;     /* Trimmed surface id.
int    get_set;   /* Get/Set.
int    opt;       /* = 1 Parametric box.
/*               /* = 2 Geometric box.
/*
/* Input / Output :
/*
double box[6];    /* The parametric/geometric enclosing box
/* Parametric box - {Umin, Vmin, Umax, Vmax}
/* Geometric box -
/* {Xmin, Ymin, Zmin, Xmax, Ymax, Zmax}
/*
/* Output :
/*
int    *ierflg;   /* Error flag:
/* 0 = OK.
/* -1 = Error.
/*- - - - - */

```

UBSSPW

Get/Set a list of control points & weights (One section).

Syntax :

```

/*- - - - - */
void UBSSPW ( Getset, Mode, Idsrfr, Vindstart, Uindstart, Num, Ptscrds, Ptswgts,
              Status )
/*- - - - - */
/*
/* Input :
/*
int      *Getset;    /*      1 = Get data,
/*                  /*      2 = Set data.
int      *Mode;      /*      1 = Points only.
/*                  /*      2 = Weights only ( if defined ).
/*                  /*      3 = Points + Weights ( if defined ).
int      *Idsrfr;    /* The surface ID.
int      *Vindstart; /* The row index to be get/set ( Start index is 0 ).
int      *Uindstart; /* The index of the 1st control point to be get/set.
int      *Num;       /* No. of points to get/set.
/*
/* Input/Output :
/*
double   Ptscrds[3*Num]; /* The coordinates of control points buffer ( MODEL ),
/*                        /* ( in "Set" mode ).
/*                        /* The coordinates of control points buffer ( MODEL ),
/*                        /* ( in "Get" mode ).
double   Ptswgts[3*Num]; /* The control points weights buffer, ( in "Set" mode ).
/*                        /* The control points weights buffer, ( in "Get" mode ).
/*
/* Output :
/*
int      *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - After using this routine with the "Set" Mode, use the following sequence:

```
Mode = 0;  
DLDISP( IdSrf, &Mode );  
Mode = 1;  
DLDISP( IdSrf, &Mode );
```

UBSURF

Create a Nurbs surface.

Syntax :

```

/*- - - - - */
int UBSURF ( Numu, Numv, Nsec, Ncrs, Degu, Degv, Peru, Perv, Nrat, Wgh, Cnp,
              Status )
/*- - - - - */
/*
/* Input :
/*
int *Numu;      /* Number of displayed U-curves.
int *Numv;      /* Number of displayed V-curves.
int *Nsec;      /* Number of sections.
int *Ncrs;      /* Number of cross-sections.
int *Degu;      /* Degree of the U parametric curve.
int *Degv;      /* Degree of the V parametric curve.
int *Peru;      /* 1 = Periodic in the U direction.
/*              /* 0 = Non-periodic in the U direction.
int *Perv;      /* 1 = Periodic in the V direction.
/*              /* 0 = Non-periodic in the V direction.
int *Nrat;      /* 0 : With weights.
/*              /* != 0 : Without weights.
double *Wgh;     /* Weights of the control points ( if NRAT = 0, else
/*              /* ignored ).
double *Cnp;     /* Coordinates of the control points.
/*              /*
/* Output :
/*
int *Status;     /* See General Concepts-Status on page 1-2.
/*              /*
/* Returns :
/*              /*
/* ID of the created surface ( if STATUS = 0 ).
/*              /*
/*- - - - - */

```

UBZSRF

Create a Bezier Mesh surface.

Syntax :

```

/*- - - - - */
int UBZSRF ( Pmesh, Nsec, Ncrs, Umesh, Vmesh, Numu, Numv, Status )
/*- - - - - */
/*
/* Input :
/*
double *Pmesh;  /* The coordinates of the control points.
int *Nsec;      /* Number of the section curves.
int *Ncrs;      /* Number of the cross-section curves.
double Umesh[Ncrs-1]; /* The U-parametric sequence.
double Vmesh[Nsec-1]; /* The V-parametric sequence.
int *Numu;      /* Number of displayed cross-section curves.
int *Numv;      /* Number of displayed section curves.
/*              /*
/* Output :
/*
int *Status;    /* See General Concepts-Status on page 1-2.
/*              /*
/* Returns :
/*              /*
/* ID of the created surface ( if STATUS = 0 ),
/*              /*
/* otherwise -1.
/*              /*
/*- - - - - */

```

Note : - NUMP = (3*(NCRS-1)+1)*(3*(NSEC-1)+1) : number of control points

UMESH(1) : Parameter of section # 2 (4, ..)
 VMESH(1) : Parameter of cross section # 2 (.., 4)
 Parameters of control point (1, 1) are always (0.0, 0.0)
 If NSEC = 2 : VMESH(1) = 1 If NCRS = 2 : UMESH(1) = 1

UCRBCM

Create a bi-cubic mesh surface (free slopes).

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int UCRBCM ( Idc, Nsec, Ncrs, Numu, Numv, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* The IDs of the section curves ( 2D/3D ).
/* ( If IDC[I] < 0 : the opposite is taken ).
int *Idc;
/* Number of section curves.
int *Nsec;
/* Number of cross-sections to be created.
int *Ncrs;
/* Number of U=Const. displayed curves.
int *Numu;
/* Number of V=Const. displayed curves.
int *Numv;
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
int *Status;
/*
/* Returns :
/* ID of the created surface.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - NCRS must be > 2.

UCR2SP

Create a Bezier Mesh surface as in the DRIVE >> TWO SPINES command of the Modeling application.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int UCR2SP ( Tol, Modsec, Idcrv, Optcrv, Limcrv, Dspsec, Dspdrv, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double *Tol;      /* Tolerance for approximation.
int *Modsec;      /* 1 = Constant height.
/* 2 = Variable height.
int Idcrv[4];     /* "IDCRV[0]":
/* ID of the 1st section curve.
/* "IDCRV[1]":
/* ID of the 2nd section curve(if "IDCRV[1]" = 0 - only
/* one section ).
/* "IDCRV[2]":
/* ID of the 1st drive curve.
/* "IDCRV[3]":
/* ID of the 2nd drive curve ( if "IDCRV[3]" = 0 - the
/* 2nd drive curve is a point ).
int Optcrv[4];   /* "OPTCRV[i]"
/* The method for defining "IDCRV[i]" limits:
/* 1 = The start & end parameters in LIMCRV.
/* 2 = The start & end parameters are the start &
/* end parameters of the curve in the data base.
/* 3 = The start & end parameters are the end &
/* start parameters of the curve in the data
/* base.
double Limcrv[9]; /* Pairs of curves limits (0:1 for the 1st curve etc.).
/* "LIMCRV[6-8]"
/* The coordinates of the drive point ( spine & point
/* option, IDCRV[3] = 0 ).
int *Dspsec;     /* Number of displayed section curves.
int *Dspdrv;     /* Number of displayed drive curves.
/*
/* Output :
/*
int *Status;     /* Error Flag:
/* 0 = O.K.<0 = Error ( see General Notes )
/*
/* Returns :
/* ID of the created surface ( if STATUS = 0 ),
/* otherwise -1.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UCRPIP

Create a Bezier Mesh surface as in the following DRIVE options of the Modeling application:

DRIVE >> SPINE >> ONE SECTION
DRIVE >> SPINE >> TWO SECTIONS
DRIVE >> SPINE & PLANE

Syntax :

```

/*- - - - - */
int UCRPIP ( Optpln, Tol, Idcrv, Optcrv, LimcrvDspsec, Dspdrv, Status ) */
/*- - - - - */
/* Input : */
/* */
int *Optpln; /* 1 = No plane definition. */
/* 2 = Plane was defined. */
double *Tol; /* Tolerance for approximation. */
/* */
/* */
int Idcrv[3]; /* ID of the 1st section curve. */
/* ID of the 2nd section curve(if IDCVR(2)=0 - only one */
/* section ). */
/* ID of the drive curve. */
/* */
/* Output : */
/* */
int Optcrv[3]; /* Method used for defining Idcrv[i] limits: */
/* 1 = The start & end parameters defined in LIMCRV. */
/* 2 = The start & end parameters are the start & */
/* end parameters of the curve in the data base. */
/* 3 = The start & end parameters are the end & */
/* start parameters of the curve in the data */
/* base. */
/* */
/* Input : */
/* */
double Limcrv[6]; /* Pairs of curves limits (1:2 for the 1st curve etc.). */
int *Dspsec; /* Number of displayed section curves. */
int *Dspdrv; /* Number of displayed drive curves. */
/* */
/* Output : */
/* */
int *Status; /* See General Concepts-Status on page 1-2. */
/* */
/* Returns : */
/* ID of the created surface ( if STATUS = 0 ), */
/* otherwise -1. */
/*- - - - - */

```

UEVSRD

Given surface parameters get coordinates of point on it and derivations in this point.

Syntax :

```

/*- - - - - */
void UEVSRD ( Ids, System, U, V, Ptrsrf, Status )
/*- - - - - */
/*
/* Input :
/*
int      *Ids;      /* ID number of the surface.
int      *System;   /* Coordinate system for evaluation:
/*      1 = MODEL,
/*      2 = WORK
double *U;          /*
double *V;          /* Given parameters.
/*
/* Output :
/*
double  Ptrsrf[18]; /* "Ptrsrf[0-2]"
/* X, Y, Z coordinates of point on the surface.
/* "Ptrsrf[3-5]"
/* Xu, Yu, Zu derivations in this point.
/* "Ptrsrf[6-8]"
/* Xv, Yv, Zv derivations in this point.
/* "Ptrsrf[9-11]"
/* Xuu, Yuu, Zuu 2nd derivations in this point.
/* "Ptrsrf[12-14]"
/* Xvv, Yvv, Zvv 2nd derivations in this point.
/* "Ptrsrf[15-17]"
/* Xuv, Yuv, Zuv 2nd derivations in this point.
int      *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UFILLC

Create a fillet surface between two surfaces with a constant radius.

Syntax :

```

/*- - - - - */
void UFILLC( Ids1, Ids2, Nsgn, Rad, ParIni, Tol, NumU, NumV, TrmFlg, EdgFlg, Idf, Sta
tus )
/*- - - - - */

    /* Input:                                     */
    /* The ID of the 1st surface.                  */
int     *Ids1;
    /* The ID of the 2nd surface.                  */
int     *Ids2;
    /* Directions of fillets:                      */
int     Nsgn[2];
        /*      1 = In the direction of the normal vector.          */
        /*      -1 = In the opposite direction of the normal       */
        /*            Vector.                                         */
double   *Rad;
    /* Radius of fillet.                                          */
double   ParIni[4];
    /* The initial surface parameters.                    */
double   *Tol;
    /* The tolerance.                                             */
int      *NumU;
    /* The number of u display lines.                        */
int      *NumV;
    /* The number of v display lines.                        */
int      *TrmFlg;
        /*      1 = keep original.                                  */
        /*      2 = trim first.                                    */
        /*      3 = trim second.                                   */
        /*      4 = trim both.                                      */
int      *EdgFlg;
        /*      1 = Extend OFF                                       */
        /*      2 = Extend ON                                        */
    /* Output:                                                 */
    /* The IDs of 3 fillets and 2 trimmed surfaces.           */
int      Idf[5];
        /* -1 = The ID does not exist.                             */
int      *Status;
    /* See General Concepts-Status on page 1-2.                */
        /* -3 = Non-smooth surfaces.                               */
/*- - - - - */

```

UFILLV

Create a fillet surface between two surfaces with variable radius.

Syntax :

```

/*- - - - - */
void UFILLV ( Ids1, Ids2, Nsgn, Rads, Parini, Tol, Flagpl, Plane, Ptref, Numu,
              Numv, Trmflg, Idf, Status )
/*- - - - - */

/*
/* Input :
/*
int    *Ids1;    /* The 1st surface ID.
int    *Ids2;    /* The 2nd surface ID.
int     Nsgn[2]; /* Directions of the fillets:
/*      1 = In the direction of the normal vector,
/*      -1 = In the opposite direction of the normal
/* vector.
double Rads[2];  /* The RADII of the fillet.
double Parini[4]; /* The initial surface parameters.
double *Tol;     /* The tolerance.
int     *Flagpl; /*      0 = Without limit planes.
/*      1 = With limit planes.
double Plane[8]; /* The coefficients of limit planes.
double Ptref[6]; /* The reference points.
int     *Numu;   /* The number of u display lines.
int     *Numv;   /* The number of v display lines.
int     *Trmflg; /*      1 = Keep original.
/*      2 = Trim surfaces.
/*
/* Output :
/*
int     Idf[3];  /* The IDs of 2 trimmed surfaces and fillet.
/*      -1 = The ID does not exist.
int     *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UGNRBS

Get properties of NURBS.

Syntax :

```

/*- - - - - */
void UGNRBS ( Idsrf, Nctrlpointsu, Nctrlpointsv, Degu, Degv, Closeu, Closev,
              Weightflag, Peru, Perv, Status )
/*- - - - - */

/*
/* Input :
/*
int    *Idsrf; /* The surface ID.
/*
/* Output :
/*
int     *Nctrlpointsu; /* Number of Control Points to U - spline definition.
int     *Nctrlpointsv; /* Number of Control Points to V - spline definition.
int     *Degu;         /* The degree of the U parameter.
int     *Degv;         /* The degree of the U parameter.
int     *Closeu;       /*      1 = U - spline is closed,
/*      0 = not closed.
int     *Closev;       /*      1 = V - spline is closed,
/*      0 = not closed.
int     *Weightflag;   /* The surface was created:
/*      0 : with sequences of weighs of control points
/*      !=0 : without sequences of weighs of control points.
int     *Peru;         /* The periodic flag of U parameter.
int     *Perv;         /* The periodic flag of V parameter.
/*      1 = Periodic,
/*      0 = No periodic.
int     *Status;       /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UGPARS

Given the ID of a surface and parameter, create a parametric spline.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UGPARS ( Idsrf, Tol, Opt, Par, Idspl, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Idsrf;    /* The ID of the surface.
double *Tol;      /* A given tolerance for the approximation.
int    *Opt;      /* 1 = U=Const. parametric curve.
/*          2 = V=Const. parametric curve.
double *Par;      /* The surface constant parameter.
/*
/* Output :
/*
int    *Idspl;    /* The created spline ID.
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UGSBZP

Get/Set control points of a Bezier Mesh surface.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UGSBZP ( Getset, Optpar, Id, Iu, Iv, Np, Points, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Getset;   /* 1 = Get.
/*          2 = Set.
int    *Optpar;   /* 1 = U=Const. parametric curve.
/*          2 = V=Const. parametric curve.
int    *Id;       /* ID of surface.
int    *Iu;       /*
int    *Iv;       /* The initial U & V indices.
/*          The 1st index is 1.
int    *Np;       /* Number of points to get/set.
/*
/* Output :
/*
double *Points;   /* The coordinates of Bezier points in Model.
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - If OPTPAR = 1 : 0 < NP (3 * NPATU + 1)
 If OPTPAR = 2 : 0 < NP (3 * NPATV + 1)
 Use UNPBZS for receipt of NPATU and NPATV

If GETSET = 2, use DLDISP (ID, 0) before UGSBZP, and then DLDISP (ID, 1).

UGSPRM

Given the ID of a surface and coordinates of a point, retrieve the parameters of the surface closest to the indicated point.

Syntax :

```

/*- - - - - */
void UGSPRM ( Idsrf, Opt, Tol, Point, Uv, Ptsrf, Ptflag, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idsrf;    /* ID number of the surface.
int    *Opt;      /* 1 = On the surface only.
              /* 2 = May be outside the surface.
double *Tol;      /* The tolerance.
double Point[3];  /* The coordinates of the point.
/*
/* Output :
/*
double Uv[2];     /* Parameters of the surface closest to the point of
              /* indication.
double Ptsrf[3];  /* The coordinates of the point on the surface.
int    *Ptflag;   /* 1 = PTSRF lies on the surface.
              /* 0 = PTSRF is the closest point.
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UGSSDC

Get/Set a number of displayed curves to display a surface.

Syntax :

```

/*- - - - - */
void UGSSDC ( Getset, Idsrf, Numu, Numv, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Getset;   /* 1 = Get data,
int    *Idsrf;    /* 2 = Set data.
              /* The surface ID.
/*
/* Input/Output :
/*
int    *Numu;     /* No. of displayed U=const curves ( in "Set" mode ).
int    *Numv;     /* No. of displayed U=const curves ( in "Get" mode ).
              /* No. of displayed V=const curves ( in "Set" mode ).
              /* No. of displayed V=const curves ( in "Get" mode ).
/*
/* Output :
/*
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UGTRPO

Given the ID of a trimmed surface, retrieve the UV polygon's data.

Syntax :

```

/*- - - - - */
void UGTRPO ( Idtrs, Maxsiz, Maxnum, Polbuf, Pntbuf, Polsiz, Polnum, Status )
/*- - - - - */
/*
/* Input :
/*
/* The ID of the trimmed surface.
/* Maximum size of polygons.
/* Maximum number of polygons.
/*
/* Output :
/*
/* The UV polygons.
/* The number of points in each polygon.
/* The size of polygon's buffer.
/* The number of polygons.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */
Note :      - Use TRSGSZ to search for MAXSIZ and MAXNUM. In order to retrieve if the
              point with given parameters within a trimmed area or not use INPOLD.

```

ULMSRF

Get surface limits from data base.

Syntax :

```

/*- - - - - */
void ULMSRF ( Ids, Limu, Limv, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID number of the surface.
/*
/* Output :
/*
/* U parameter limits : Ustart and Uend.
/* V parameter limits : Vstart and Vend.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```

UNPBZS

Get the number of patches defining a bezier surface.

Syntax :

```

/*- - - - - */
void UNPBZS ( Ids, Npatu, Npatv, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the surface.
/*
/* Output :
/*
/* Number of patches in U direction.
/* Number of patches in V direction.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */
Note :      - Number of control points is:in U direction - 3*NPATU+1, in V direction -
              3*NPATV+1.

```

UREVOLS

Create a surface of revolution by rotating a curve around an axis.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int UREVOLS ( Idcurve, Orjpoint, Axis, Angle, Numu, Numv, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                     /*
                                     /* Input :
                                     /*
int      *Idcurve;                  /* The ID of the curve to be rotated.
double   Orjpoint[3];              /* Coordinates of the start point of the rotation axis.
double   Axis[3];                  /* The direction vector of the rotation axis.
double   *Angle;                   /* Counter-clockwise rotation angle ( DEGREE ).
int      *Numu;                    /* Number of display curves in direction of the drive
                                     /* curve.
int      *Numv;                    /* Number of display curves in direction of the section
                                     /* curve.
                                     /*
                                     /* Output :
                                     /*
int      *Status;                  /* See General Concepts-Status on page 1-2.
                                     /*
                                     /* Returns :
                                     /* The ID of the created surface.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

USRFBD

Get the boundaries of a surface.

Syntax :

```

/*- - - - - */
void USRFBD ( Idsrf, Tol, Bufcon, Bufcrv, Numcon, Numcrv, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the surface.
/* The tolerance for approximation.
/*
/* Output :
/*
/* The number of curves on each contour.
/* IDs of the boundary curves.
/*
/* Input/Output :
/*
/* The maximum number of contours.
/* The number of closed contours found.
/* The maximum number of ID curves.
/* The number of curve IDs found.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/* 1 = The input value of NUMCON < the number of
/* contours with which it is possible to create
/* a surface boundary.
/* 2 = The input value of NUMCRV < the number of
/* curves with which it is possible to create
/* a surface boundary.
/* 3 = The input values of NUMCON and NUMCRV < the
/* number of contours and curves with which it
/* is possible to create a surface boundary.
/*
/*- - - - - */
Note: - If the input value of NUMCON and/or NUMCRV < number of
contours or curves with which it is possible to create a surface
boundary, the system builds the requested number of contours or
curves. However, the output value of these parameters equals the
number of contours and curves that can be created from this
surface.

```

USRFIN

Create splines along the intersection of two given surfaces.

Syntax :

```

/*- - - - - */
void USRFIN ( Ids1, Ids2, Maxs, Tol, Idsp, Ns, Status )
/*- - - - - */
/*
/* Input :
/*
/* The 1st surface ID.
/* The 2nd surface ID.
/* Maximal number of splines to create.
/* A given tolerance.
/*
/* Output :
/*
/* ID numbers of created splines.
/* Number of created splines. If NS < 0, either no
/* intersection was found or some other error occurred.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```

USRFPT

Find parameters of a surface/trimmed surface defining the point on the surface closest to the given point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void USRFPT ( Optpt, Idsrf, Ptm, Uc, Vc, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Optpt;
/*    1 = Parameters define point on the surface.
/*    2 = Parameters define point on the surface
/*        boundaries.
/*    3 = Parameters define one of the surface corners.
/*    4 = Parameters define one of the surface display
/*        curves intersection points.
int    *Idsrf;
/* ID of the surface/trimmed surface.
float  *Ptm;
/* The given point in MODEL.
/*
/* Output :
/*
float  *Uc;
float  *Vc;
/* Surface parameters defining the point on the surface
/* closest to the given point ( if STATUS = 0 ).
int    *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

USRNOR

Get the coordinates of a point which lies at a given distance from a point, that is normal to surface direction, on a given surface.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void USRNOR ( Ids, Idp, Dist, Pnt, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Ids;
/* ID number of the surface.
int    *Idp;
/* ID number of the point on the surface.
double *Dist;
/* Distance between points ( may be of any sign ).
/*
/* Output :
/*
double Pnt[3];
/* Coordinates of second point.
int    *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

USRSC

Scale a general surface.

Syntax :

```

/*- - - - - */
int USRSC ( Idsrf, Optkeep, Tol, Scale, Iducs, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the surface.
/* 1 = Keep original.
/* 2 = Delete original.
/* Tolerance for approximation.
/* Scales for X, Y and Z axes.
/* ID of the reference UCS.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created surface ( if STATUS = 0 ),
/* otherwise -1.
/*- - - - - */

```

USRSEC

Create splines along intersection of two given surfaces.

Syntax :

```

/*- - - - - */
void USRSEC ( Ids1, Ids2, Idp1, Idp2, Maxs, Tol, Idsp, Ns )
/*- - - - - */
/*
/* Input :
/*
/* ID numbers of the surfaces.
/* ID numbers of points on the surface, close to
/* intersection.
/* Maximal number of splines to create.
/* Given tolerance
/*
/* Output :
/*
/* ID numbers of created splines.
/* Number of created splines; if < 0, no intersection
/* was found or other error ( see General Notes ).
/*- - - - - */

```

UTRSCO

Trim a surface/trimmed surface by a contour lying on it.

Syntax :

```

/*- - - - - */
void UTRSCO ( Optdiv, Tol, Uvi, Idsrf, Bufcon, Idnew, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int    *Optdiv;    /* 1 = Trim.2 = Divide.
double *Tol;       /* Tolerance for approximation.
double  Uvi[2];    /* U and V parameters of the point on the remaining
/* side of the surface( only if OPTDIV = 1 ).
int    *Idsrf;     /* ID of the surface/trimmed surface.
int    *Bufcon;    /* Contour buffer.
/*
/* Output :
/*
int    Idnew[2];   /* ID of the new surface( s )/trimmed surface( s ).
int    *Status;    /* See General Concepts-Status on page 1-2.
/*      1 = Intersection was found, not all surfaces
/*      were created ( divide option ).
/*      -1 = Wrong input data.
/*      -3 = Trimming algorithm failed. No surface created.*/
/*- - - - - */

```

UTRSPA

Trim a surface/trimmed surface by a parameter.

Syntax :

```

/*- - - - - */
void UTRSPA ( Optdiv, Uvi, Idsrf, Optpar, Par, Idnew, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int    *Optdiv;    /* 1 = trim.
/*      2 = divide.
double  Uvi[2];    /* U and V parameters of the point on the remaining
/* side of the surface ( only if OPTDIV = 1 ).
int    *Idsrf;     /* ID of the surface/trimmed surface.
int    *Optpar;    /* 1 : U = const.
/*      2 : V = const.
double *Par;       /* Trimming parameter.
/*
/* Output :
/*
int    Idnew[2];   /* ID of the new surface( s )/trimmed surface( s ).
int    *Status;    /* See General Concepts-Status on page 1-2.
/*      1 = Intersection was found, not all surfaces
/*      were created ( divide option ).
/*      -1 = Wrong input data.
/*      -3 = Trimming algorithm failed. No surface created.*/
/*- - - - - */

```

UTRSPC

Trim a surface/trimmed surface by a projected contour.

Syntax :

```

/*- - - - - */
void UTRSPC ( Optdiv, Tol, Uvi, Idsrf, Bufcon, Dir, Angle, Idnew, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Optdiv;    /*    1 = trim.
/*    2 = divide.
double *Tol;       /* Tolerance for approximation.
double  Uvi[4];    /* U and V parameters of the point on the remaining
/* side of the surface (only if OPTDIV = 1).The U and V
/* parameters indicate the point of projection.
int    *Idsrf;     /* ID of the surface/trimmed surface.
int    *Bufcon;    /* Contour buffer.
double Dir[3];     /* Project direction.
double *Angle;     /* Project angle.
/*
/* Output :
/*
int    Idnew[2];   /* ID of the new surface( s )/trimmed surface( s ).
int    *Status;    /* See General Concepts-Status on page 1-2.
/*    1 = Intersection was found, not all surfaces
/*    were created ( divide option ).
/*    -1 = Wrong input data.
/*    -3 = Trimming algorithm failed. No surface created.
/*- - - - - */

```

UTRSPL

Trim a surface/trimmed surface by the work plane.

Syntax :

```

/*- - - - - */
void UTRSPL ( Optdiv, Tol, Uvi, Idsrf, Idnew, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Optdiv;    /*    1 = trim.
/*    2 = divide.
double *Tol;       /* Tolerance for approximation.
double  Uvi[2];    /* U and V parameters of the point on the remaining
/* side of the surface ( only if OPTDIV = 1 ).
int    *Idsrf;     /* ID of the surface/trimmed surface.
/*
/* Output :
/*
int    Idnew[2];   /* ID of the new surface( s )/trimmed surface( s ).
int    *Status;    /* See General Concepts-Status on page 1-2.
/*    1 = Intersection was found, not all surfaces
/*    were created ( divide option ).
/*    -1 = Wrong input data.
/*    -2 = Intersection was not found.
/*    -3 = Trimming algorithm failed. No surface created.
/*    -4 = Too many points of plane intersection - try
/*    to increase the tolerance.
/*    -5 = Only one intersection was taken although
/*    there are a few ( only if OPTDIV = 2 ).
/*- - - - - */

```

UTRSSR

Trim a surface/trimmed surface by a surface.

Syntax :

```

/*- - - - - */
void UTRSSR ( Optdiv, Opboth, Tol, Uvi, Idsrf, Idtrm, Idnew, Status )
/*- - - - - */
/*
/* Input :
/*
int *Optdiv; /* 1 = Trim.
/* 2 = Divide.
int *Opboth; /* 1 = Trim only the 1st surface.
/* 2 = Trim both surfaces.
double *Tol; /* Tolerance for approximation.
double Uvi[2]; /* U and V parameters of the point on the remaining
/* side of the surface ( only if OPTDIV = 1 ).
int *Idsrf; /* ID of the surface/trimmed surface.
int *Idtrm; /* ID of the trimming surface.
/*
/* Output :
/*
int Idnew[2]; /* ID of the new surface( s )/trimmed surface( s ).
int *Status; /* See General Concepts-Status on page 1-2.
/* 1 = Intersection was found, not all surfaces
/* were created ( divide option ).
/* -1 = Wrong input data.
/* -2 = Intersection was not found.
/* -3 = Trimming algorithm failed. No surface created.
/*- - - - - */

```

Note : - If OPTDIV=1 and OPBOTH=2, use the function CDK_SRF_TRM_SRF.

UVRBSR

Get/set control points of a Bezier Mesh surface.

Syntax :

```

/*- - - - - */
void UVRBSR ( Getset, Optpar, Idsrf, Iu, Iv, Npbez, Pts, Status )
/*- - - - - */
/*
/* Input :
/*
int *Getset; /* 1 = Get.
/* 2 = Set.
int *Optpar; /* 1 : U=Const. parametric curve.
/* 2 : V=Const. parametric curve.
int *Idsrf; /* ID of the surface.
int *Iu; /* U index ( cross-section & section numbers for the
/* first point to get/set ).
int *Iv; /* V index ( cross-section & section numbers for the
/* first point to get/set ).
int *Npbez; /* Number of control points to get/set.If NPBEZ=0: all
/* points on the chosen parametric curve.
/*
/* Input/Output :
/*
double *Pts; /* Coordinates of the control points ( if GETSET = 2 ).
/* Coordinates of the control points ( if GETSET = 1 ).
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```



SWEEP

CDK_SWEEP_ANGLE

To sweep entities angular.

Syntax :

```

/*- - - - - */
void CDK_SWEEP_ANGLE ( Origin, Axis, Angle, Mode, Numents, Instant, Arcs,
                      Numarcs, Status )
/*- - - - - */
/*
/* Input :
/*
int    Origin[];    /* Array containing origin entities to be swept
/* ( Size of array "NumEnts" ).
double Axis[6];    /* The coordinates of start [0...2] and end [3...5]
/* points of sweep axis ( MODEL ).
double *Angle;    /* The angle of sweep ( DEGREE ).
int    *Mode;    /* 0 = not create sweep arcs,
/* 1 = create.
/*
/* Input/Output :
/*
int    *Numents;    /* Number of entities to be swept.
/* Number of swept entities.
/*
/* Output :
/*
int    Instant[];    /* Array containing IDÆs of the swept entities
/* ( Size of array "NumEnts" ).
int    Arcs[];    /* Array containing IDÆs of the sweep arcs
/* ( Size of array "Mode * 2 * NumEnts" ).
int    *Numarcs;    /* Number of created sweep arcs.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_SWEEP_DELTA

To sweep entities linearly.

Syntax :

```

/*- - - - - */
void CDK_SWEEP_DELTA ( Origin, Vector, Mode, Numents, Instant, Lines, Numlines,
                      Status )
/*- - - - - */
/*
/* Input :
/*
int    Origin[];    /* Array containing origin entities to be swept
/* ( Size of array "NumEnts" ).
double Vector[3];    /* The sweep vector ( MODEL ).
int    *Mode;    /* 0 = not create sweep lines,
/* 1 = create.
/*
/* Input/Output :
/*
int    *Numents;    /* Number of entities to be swept.
/* Number of swept entities.
/*
/* Output :
/*
int    Instant[];    /* Array containing IDÆs of the swept entities
/* ( Size of array "NumEnts" ).
int    Lines[];    /* Array containing IDÆs of the sweep lines
/* ( Size of array "Mode * 2 * NumEnts" ).
int    *Numlines;    /* Number of created sweep lines.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

ENSWLR

Sweep entities linearly.

Syntax :

```

/*- - - - - */
void ENSWLR ( Count, Ids, Swvect, Linson, Max, Outar, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int *Count; /* Number of entities to sweep.
int Ids[Count]; /* Array containing COUNT IDs of the entities to sweep.
float Swvect[3]; /* Sweep vector ( in model system coordinates ).
int *Linson; /* 0 = Do not create sweep lines.
/* 1 = Create sweep lines.
int *Max; /* Maximum length of OUTAR.
/*
/* Output :
/*
int Outar[Max]; /* Array containing IDs of the entities on which sweep
/* was performed. If MAX is greater than COUNT, the IDs
/* of the sweep lines are appended to OUTAR.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - If LINSON = 1, the length of OUTAR must be sufficient to hold the IDs of the sweep lines.

ENSWRT

Sweep entities angularly.

Syntax :

```

/*- - - - - */
void ENSWRT ( Count, Ids, Axis, Angle, Linson, Max, Outar, Status )
/*- - - - - */
/*
/* Input :
/*
/*
int *Count; /* Number of entities to sweep.
int Ids[Count]; /* Array containing COUNT IDs of the entities to sweep.
float Axis[6]; /* 2 points on the rotation axis ( in model system
/* coordinates ).
float *Angle; /* Angle of rotation.
int *Linson; /* 0 = Do not create sweep lines.
/* 1 = Create sweep lines.
int *Max; /* Maximum length of OUTAR.
/*
/* Output :
/*
int Outar[Max]; /* Array containing IDs of the entities on which sweep
/* was performed. If MAX is greater than COUNT, the IDs
/* of the sweep lines are appended to OUTAR.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - If LINSON = 1, the length of OUTAR must be sufficient to hold the IDs of the sweep lines.



TRANSFORMATION

GTPLMA

Obtain a transformation matrix which is defined interactively either by indicating 3 points, or, by specifying a position point and a rotation angle (around the Z axis).

Syntax :

```
/*- - - - - */
void GTPLMA ( Transf, Respon )
/*- - - - - */
/*
/* Output :
/*
float   Transf[12]; /* Rotation ( first 9 elements of the array ) and
/* translation matrix ( last 3 elements ).
int     *Respon;    /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

MADM2W

Transform a point from the MODEL coordinate system to the current work coordinate system.

Syntax :

```
/*- - - - - */
void MADM2W ( Pointmodel, Pointwork, )
/*- - - - - */
/*
/* Input :
/*
double  Pointmodel[3]; /* The coordinates of the point in MODEL coordinate
/* system.
/*
/* Output :
/*
double  Pointwork[3]; /* The coordinates of the point in WORK coordinate
/* system.
/*- - - - - */
```

MADW2M

Transform a point from the current work coordinate system to the MODEL coordinate system.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void MADW2M ( Pointwork, Pointmodel )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
      /*
      /* Input :
      /*
double  Pointwork[3]; /* The coordinates of the point in WORK coordinate
      /* system.
      /*
      /* Output :
      /*
double  Pointmodel[3]; /* The coordinates of the point in MODEL coordinate
      /* system.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MAPM2W

Transform a point from the model coordinate system to the current work coordinate system.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void MAPM2W ( Pm, Pw )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
      /*
      /* Input :
      /*
float   Pm[3];        /* 3D point in model coordinate system.
      /*
      /* Output :
      /*
float   Pw[3];        /* 3D point in work coordinate system.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MAPW2M

Transform a point from the current work coordinate system to the model coordinate system.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void MAPW2M ( Pw, Pm )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
      /*
      /* Input :
      /*
float   Pw[3];        /* 3D point in work coordinate system.
      /*
      /* Output :
      /*
float   Pm[3];        /* 3D point in model coordinate system.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TR3PTS

Calculate a transformation matrix.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TR3PTS ( Base, Dir, Aux, Mat, Respon )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* The origin.
/* A point on the X axis.
/* A point to indicate Y.
/*
/* Output :
/*
/* Rotation ( first 9 elements ) and translation matrix
/* ( last 3 elements ).
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float Base[3];
float Dir[3];
float Aux[3];
float Mat[12];
int *Respon;

```

TRCONC

Calculate the transformation matrix (TRRESU) resulting from concatenation of transformation matrices TRA and TRB. TRRESU is equivalent to having first applied TRA and then TRB.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRCONC ( Tra, Trb, Trresu )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Transformation matrices to be concatenated.
/*
/* Output :
/*
/* Resulting concatenated transformation.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float Tra[12];
float Trb[12];
float Trresu[12];

```

Note : - Matrix TRA(1:9) must be transposed.

TRCONM

Calculate the transformation matrix (TRRESU) resulting from concatenation of transformation matrices TRA and TRB. TRRESU is equivalent to having first applied TRA and then TRB.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRCONM ( Tra, Trb, Trresu )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Transformation matrices to be concatenated.
/*
/* Output :
/*
/* Resulting concatenated transformation.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
float Tra[12];
float Trb[12];
float Trresu[12];

```

TRCONMD

Calculate the transformation resulting from concatenation of transformation matrices TransfA and TransfB. The resulting transformation matrix TransfRes is equivalent of having first applied TransfA and THEN TransfB.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRCONMD ( Transfa, Transfb, Transfres )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
    /*
    /* Input :
    /*
double  Transfa[12]; /* The 1st transformation matrix to be concatenated.
double  Transfb[12]; /* The 2nd transformation matrix to be concatenated.
    /*
    /* Output :
    /*
double  Transfres[12]; /* Resulting concatenated transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TRMILI

Calculate a transformation matrix which defines mirroring about the line.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRMILI ( Point1, Point2, Transf )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
    /*
    /* Input :
    /*
float   Point1[3]; /* A point on the mirror line.
float   Point2[3]; /* Direction point for the line.
    /*
    /* Output :
    /*
float   Transf[12]; /* Resulting transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TRMILID

Calculate the transformation which defines mirroring about the line.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRMILID ( Point, Vector, Transform, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
    /*
    /* Input :
    /*
double  Point[3]; /* A point on the mirror line.
double  Vector[3]; /* The direction vector of the mirror axis.
    /*
    /* Output :
    /*
double  Transform[12]; /* The resulting transformation matrix.
int      *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TRMIPL Calculate the transformation matrix which defines mirroring about the plane, PLANE.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRMIPL ( Plane, Transf )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float   Plane[4];    /* 4 coefficients defining the PLANE.
/*
/* Output :
/*
float   Transf[12];  /* Resulting transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

TRMIPLD Calculate the transformation which defines mirroring about the plane.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRMIPLD ( Plane, Transform, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double  Plane[4];    /* 4 coefficients, defining the plane.
/*
/* Output :
/*
double  Transform[12]; /* The resulting transformation matrix.
int      *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

TRMIPO Calculate the transformation matrix which defines mirroring through the point, POINT.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRMIPO ( Point, Transf )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float   Point[3];    /* Point through which mirroring will take place.
/*
/* Output :
/*
float   Transf[12];  /* Resulting transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

TRMIPOD

Calculate the transformation which defines mirroring about the point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRMIPOD ( Point, Transform )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double Point[3]; /* The coordinates of the point, through which mirroring
/* will take place.
/*
/* Output :
/*
double Transform[12]; /* The resulting transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TRROAXCalculate the transformation matrix which defines a rotation about the axis, **AXIS**, through the point, **POINT** by an angle, **ANGLE**.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRROAX ( Point, Axis, Angle, Transf )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float Point[3]; /* Point through which AXIS passes.
float Axis[3]; /* Direction point on the axis.
float *Angle; /* Rotation angle ( in degrees ).
/*
/* Output :
/*
float Transf[12]; /* Resulting transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TRROAXD

Calculate the transformation which defines a rotation about the axis.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRROAXD ( Point, Vector, Angle, Transform, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double Point[3]; /* The coordinates of the point, through which the axis
/* passes.
double Vector[3]; /* The direction vector of the axis.
double *Angle; /* The rotation angle ( DEGREE ).
/*
/* Output :
/*
double Transform[12]; /* The resulting transformation matrix.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TRSCA1

Build the transformation matrix required to change the scale of an entity by a given factor, relative to a specified position point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRSCA1 ( Point, Factor, Transf )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float   Point[3];    /* The coordinates of the fixed point.
float   *Factor;      /* Scaling factor.
/*
/* Output :
/*
float   Transf[12];   /* Resulting transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TRSCA1D

Build the transformation required to change the scale of an entity by a given factor, relative to a specified position point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRSCA1D ( Point, Scale, Transform )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
double  Point[3];    /* The coordinates of fixed point.
double  *Scale;       /* The scaling factor.
/*
/* Output :
/*
double  Transform[12]; /* The resulting transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TRSHIF

Calculate the transformation which defines a shift by the vector SHIFT.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRSHIF ( Shift, Transf )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float   Shift[3];    /* Point which, with the origin, defines the shift
/* vector.
/*
/* Output :
/*
float   Transf[12];   /* Resulting transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

TRSHIFD

Calculate the transformation which defines a shift.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void TRSHIFD ( Shift, Transform )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
double  Shift[3];              /* The vector of the shift.
                                /*
                                /* Output :
                                /*
double  Transform[12];        /* The resulting transformation matrix.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UTR3PNT

Calculate a transformation matrix.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void UTR3PNT ( Base, Xaxis, Ydirection, Transform, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
double  Base[3];              /* The coordinates of the origin point.
double  Xaxis[3];             /* The coordinates of the point on X-axis.
double  Ydirection[3];        /* The coordinates of the point to indicate Y-direction.
                                /*
                                /* Output :
                                /*
double  Transform[12];        /* The resulting transformation matrix.
int      *Status;             /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```



TRIM

CDK_TRS_BYPLN Trim a surface/trimmed surface by the work plane.
Syntax :

```

/*- - - - - */
void CDK_TRS_BYPLN (optdiv,tol,pln_coef,idsrf,idnew,numsrfl,ierflg)
/*- - - - - */
/* Input : */
/*
T_INT      optdiv      ; /* =1 : Trim - Keep part "above" according to normal */
/*                ; /* =2 : Divide */
/*
T_DOUBLE   tol         ; /* A given tolerance */
/*
T_DOUBLE   pln_coef[4]; /* Plane coefficients */
/*
T_INT      idsrf       ; /* The surface/trimmed surface ID */
/*
/* Output : */
/*
T_INT      idnew[]     ; /* The ID of the new surface/trimmed surface */
/*
T_INT      *numsrfl    ; /* Number of surfaces in idnew */
/*
T_INT      *ierflg     ; /* Error flag */
/*                ; /* = 0 : o.k. */
/*                ; /* In all other cases trimming had not been done: */
/*                ; /* ==-1 : error */
/*                ; /* ==-2 : bad trimming contour */
/*                ; /* ==-3 : number of trimming curves/contours is greater */
/*                ; /*                than maximum */
/*                ; /* ==-4 : too many points of plane intersection */
/*                ; /*                try to increase tolerance */
/*                ; /* ==-5 : intersection was not found */
/*- - - - - */

```

CVDELP Delete the part of a curve "Idolcv", which lies between the curves "Idtcv1" and "Idtcv2".
Syntax :

```

/*- - - - - */
void CVDELP ( Idtcv1, Idtcv2, Idolcv, Idnwcv, Status )
/*- - - - - */
/* Input : */
/*
int      *Idtcv1;      /*
int      *Idtcv2;      /* IDs of the two trimming curves.
int      *Idolcv;      /* ID of the curve whose part is to be deleted.
/*
/* Output : */
/*
int      *Idnwcv;      /* ID of the curve remaining after the deletion of a
/*                ; /* section. ( If IDNWCV=0, the deleted part was not a
/*                ; /* part of the curve IDOLCV. )
int      *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

DIVCRV

Divide a curve by its intersection with two other curves.

Syntax :

```

/*- - - - - */
void DIVCRV ( Idd1, Idd2, Iddc, Idp1, Idp2, Idnew, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idd1; /* ID of 1st dividing curve.
int *Idd2; /* ID of 2nd dividing curve.
int *Iddc; /* ID of curve to be divided.
int *Idp1; /* ID of point closest to the 1st trim point.
int *Idp2; /* ID of point closest to the 2nd trim point.
/*
/* Output :
/*
int *Idnew; /* ID of the curve remaining after the deletion of a
/* section.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - All curves must be coplanar.

GFDIVD

Divide a curve by 2 other curves.

Syntax :

```

/*- - - - - */
int GFDIVD ( Idd1, Idp1, Idd2, Idp2, Idc, Idcp, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idd1; /* ID of 1st dividing curve.
int *Idp1; /* ID of point on IDD1 closest to the trimming point.
int *Idd2; /* ID of 2nd dividing curve.
int *Idp2; /* ID of point on IDD2 closest to the trimming point.
/*
/* Input/Output :
/*
int *Idc; /* ID of curve to be divided.
/* ID of duplicate of IDC ( See Chapter 1, General
/* Concepts, Duplication of Entities ).
/*
/* Input :
/*
int *Idcp; /* ID of point on IDC.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created part of the curve.
/* Hint:
/* Use GFDIVD when one of the dividing curves
/* intersects the curve to be divided at more than one
/* point.
/*- - - - - */

```

GFDVPT

Divide an open curve by a point, or a closed curve by 2 points.

Syntax :

```

/*- - - - - */
int GFDVPT ( Idcrv, Idp1, Idp2, Status )
/*- - - - - */
/*
/* Input/Output :
/*
int *Idcrv; /* ID of curve to be divided.
/* ID of duplicate of IDCRV ( See Chapter 1, General
/* Concepts, Duplication of Entities ).
/*
/* Input :
/*
int *Idp1; /* ID of point on curve at which the curve will be
/* divided.
int *Idp2; /* ID of 2nd point on curve ( in the case of closed
/* curve, otherwise ignored ).
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created part of the curve.
/*- - - - - */

```

GFTRIM

Trim curve by another curve "Idtcrv".

Syntax :

```

/*- - - - - */
void GFTRIM ( Idtcrv, Idpt, Idcrv, Idpcrv, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idtcrv; /* ID of trimming curve.
int *Idpt; /* ID of point on trimming curve close to where
/* trimming will take place.
/*
/* Input/Output :
/*
int *Idcrv; /* ID of curve to be trimmed.
/* ID of curve after trimming ( See Chapter 1, General
/* Concepts, Duplication of Entities ).
/*
/* Input :
/*
int *Idpcrv; /* ID of point on curve to be trimmed.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

GFTRPL

Trim curve by work plane.

Syntax :

```

/*- - - - - */
void GFTRPL ( Idc, Idpr, Status )
/*- - - - - */
/*
/* Input/Output :
/*
int    *Idc;
/* ID of curve to be trimmed.
/* ID of trimmed curve ( See Chapter 1, General
/* Concepts, Duplication of Entities ).
/*
/* Input :
/*
int    *Idpr;
/* ID of point indicating side of the plane on which
/* curve will be kept.
/*
/* Output :
/*
int    *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - If, on output, IDC = -1 and STATUS = 0, the whole entity is deleted.

GFTRPT

Trim a curve by a point.

Syntax :

```

/*- - - - - */
void GFTRPT ( Idc, Idpc, Idtp1, Idtp2, Status )
/*- - - - - */
/*
/* Input/Output :
/*
int    *Idc;
/* ID of curve to be trimmed.
/* ID of trimmed curve ( See Chapter 1, General
/* Concepts, Duplication of Entities ).
/*
/* Input :
/*
int    *Idpc;
/* ID of a point on remaining section of IDC.
int    *Idtp1;
/* ID of trimming point.
int    *Idtp2;
/* ID of 2nd trimming point ( ignored in the case of an
/* opened curve ).
/*
/* Output :
/*
int    *Status;
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

TRMCRV

Trim a curve at its intersection with another curve.

Syntax :

```

/*- - - - - */
void TRMCRV ( Idt, Idc, Idpt, Idpc, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idt; /* ID of the trimming curve.
int *Idc; /* ID of the curve to be trimmed, and of the resulting
/* trimmed curve.
int *Idpt; /* ID of the point closest to the trimming point.
int *Idpc; /* ID of the point closest to the trimmed curve.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note : - Both curves must be coplanar.



General Description

The following routines are UCS associated routines.

Note: • The maximum length of a UCS name is 6 characters.

ACTUCS

Activate an existing UCS.

Syntax :

```
/*- - - - - */
void ACTUCS ( Id, Status )
/*- - - - - */
/*
/* Input :
/*
int *Id; /* ID of the UCS:
/* -100 = MODEL
/* -102 = VIEW
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

CDK_UCS_CREATE

Create a UCS by a transformation matrix.

Syntax :

```
/*- - - - - */
int CDK_UCS_CREATE ( name, scl, orig, trans )
/*- - - - - */
/*
/* Input :
/*
char name[]; /* UCS name (max len = 6).
double scl; /* The scale of the ucs display symbol.
double orig[3]; /* The ucs origin (MODEL).
double trans[9]; /* Rotation matrix (MODEL).
/* Return :
/* > 0 : UCS ID,
/* < 0 : See General Concepts-Status on page 1-2.
/* -2 : Name already exist.
/* -3 : Name is one of the reserved.
/*- - - - - */
```

CDK_UCS_CREATE_PTS Create a UCS by three points.
Syntax :

```

/*- - - - - */
int CDK_UCS_CREATE_PTS ( name, scl, orig, ptx, pty )
/*- - - - - */
/*
/* Input :
/*
char    name[];    /* UCS name (max len = 6).
double  scl;       /* The scale of the ucs display symbol.
double  orig[3];   /* The ucs origin (MODEL).
double  ptx[3];    /* The ucs X direction (MODEL).
double  pty[3];    /* The ucs Y direction (MODEL).
/* Return :
/* > 0 : UCS ID,
/* < 0 : See General Concepts-Status on page 1-2.
/* -2 : Name already exist.
/* -3 : Name is one of the reserved.
/*- - - - - */

```

CDK_UCS_DISPLAY Display UCS.
Syntax :

```

/*- - - - - */
int CDK_UCS_DISPLAY (iucsid, Mode )
/*- - - - - */
/*
/* Input :
/*
int    iucsid;    /* UCS ID.
int    Mode;      /* Display mode : = 0 : OFF, = 1 ON.
/* Return : See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_UCS_LIST Create a list of UCS names.
Syntax :

```

/*- - - - - */
int CDK_UCS_LIST( Max, Nucs, Names )
/*- - - - - */
/*
/* Input :
/*
int    Max;       /* Maximum number of names to retrieve
/*
/* Output :
/*
int    *Nucs;     /* Number of UCS's retrieved.
char    Names[][7]; /* UCS names.
/* Return :
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_UCS_MAP

Map a point from UCS to MODEL or WORK system.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_UCS_MAP ( mode, ucsid, ucspt, newpt )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int      mode;                  /* = 1 : To MODEL, = 2 : To WORK.
int      ucsid;                 /* The ID of the UCS.
double   ucspt;                 /* The 3D point in UCS coordinates.
double   newpt;                /* The transformed coordinates.
                                /*
                                /* Return : See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_UCS_MAP_FROM

Map a point from UCS to MODEL or WORK system.

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_UCS_MAP_FROM ( mode, ucsid, ucspt, syspt )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int      mode;                  /* = 1 : To MODEL, = 2 : To WORK.
int      ucsid;                 /* The ID of the UCS.
double   ucspt[3];             /* The 3D point in UCS coordinates.
                                /*
                                /* Output :
                                /*
double   *syspt;                /* The system coordinates.
                                /*
                                /* Return : Status.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_UCS_MAP_TO

Map a point from MODEL or WORK system to UCS.

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_UCS_MAP_TO ( mode, ucsid, ucspt, syspt )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int      mode;                  /* = 1 : From MODEL, = 2 : From WORK.
int      ucsid;                 /* The ID of the UCS.
double   syspt[3];             /* The system coordinates.
                                /*
                                /* Output :
                                /*
double   *ucspt;                /* The 3D point in UCS coordinates.
                                /*
                                /* Return : Status.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_UCS_TRANS

Tranform three points into a rotation matrix.

Syntax:

```

/*- - - - - */
int CDK_UCS_TRANS ( orig, ptx, pty, trans )
/*- - - - - */
/*
/* Input :
/*
double orig[3]; /* The ucs origin (MODEL).
double ptx[3]; /* The ucs X direction (MODEL).
double pty[3]; /* The ucs Y direction (MODEL).
/*
/* Output :
/*
double trans[9]; /* Rotation matrix (MODEL).
/* Return : See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_UCS_ZAXIS

Create a UCS from the Z axis only.

Syntax:

```

/*- - - - - */
int cdk_ucs_zaxis ( origin, zaxis, ucsname)
/*- - - - - */
/*
/* Input :
/*
double origin[3]; /* Origin point of UCS
double zaxis[3]; /* Point on z axis
char *ucsname; /* UCS name (new)
/*
/* Return value : new UCS ID
/*- - - - - */

```

CDK_WORKPLANE

Get / set active work plane parameters (double precision).

Syntax:

```

/*- - - - - */
int CDK_WORKPLANE (getSet, wrkptr, wrkdep)
/*- - - - - */
/*
/* Input:
/*
int getSet; /* 1: Get, 2: Set
/*
/* In/Out:
/*
int wrkptr; /* The work plane's id. (rotation pointer)
double wrkdep; /* The work plane depth
/*
/* Return value:
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CREUCS

Create a UCS, without making it the active coordinate system.

Syntax :

```

/*- - - - - */
int CREUCS ( Ucsnam, Lennam, Mat, Orig, Scl, Status )
/*- - - - - */
/*
/* Input :
/*
/* Name of the UCS.
/* Number of characters in UCSNAM.
/* Transformation matrix defining the new UCS.
/* Origin point of the new UCS.
/* Display size of the UCS icon.
/*
/* Output :
/*
/* 0 = O.K.
/* -1 = Error in program.
/* -2 = Name already exists.
/* -3 = Reserved name.
/*
/* Returns :
/* ID of the new UCS entity.
/*
/*- - - - - */

```

CRUCS3

Create a UCS entity as in the 3-PTS option.

Syntax :

```

/*- - - - - */
int CRUCS3 ( Ucsnam, Lennam, P1, P2, P3, Scl, Status )
/*- - - - - */
/*
/* Input :
/*
/* Name of the UCS.
/* Length of relevant information in UCSNAM.
/* Origin point in UCS coordinates.
/* X direction in UCS coordinates.
/* Y direction in UCS coordinates.
/* Display size of UCS icon.
/*
/* Output :
/*
/* 0 = O.K.
/* -1 = Error in program.
/* -2 = Name already exists.
/* -3 = Reserved name.
/* -4 = Error in coordinates.
/*
/* Returns :
/* ID of the new UCS entity.
/*
/*- - - - - */

```

DEFPLN Define one of the planes of a specified UCS as the active work plane.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DEFPLN ( Uid, Optpln, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int      *Uid;      /* ID of the UCS.
int      *Optpln;   /* Plane code:1= XY2= XZ3= YZ
/*
/* Output :
/*
int      *Status;   /* 0 = O.K.-1 = Error in program or ID is not a UCS
/* ID.-2 = Wrong OPTPLN input.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DELUCS Delete an existing UCS. (The active UCS cannot be deleted).

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DELUCS ( Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int      *Id;       /* ID of the UCS.
/*
/* Output :
/*
int      *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

GACUCS Retrieve the ID of the active UCS.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GACUCS ( Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int      *Id;       /* ID of the active UCS.
int      *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

ID2NAM

Given a UCS ID, retrieve the UCS name.

Syntax :

```

/*- - - - - */
void ID2NAM ( Id, Ucsnam, Status )
/*- - - - - */
          /*
          /* Input :
          /*
int      *Id;      /* ID of the UCS.
          /*
          /* Output :
          /*
char      Ucsnam[6]; /* Name of the UCS.
int      *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

MADU2M

Map a point from the UCS to the MODEL system.

Syntax:

```

/*- - - - - */
void madu2m ( ucsid, pu, pm, res)
/*- - - - - */
          /*
          /* Input :
          /*
int      *ucsid;   /* The ID of the UCS.
double   pu[];     /* 3D point in UCS coordinates
          /*
          /* Output :
          /*
double   pm[];     /* 3D point in MODEL system
int      *res;     /* 0 = O.K. otherwise: ERROR
/*- - - - - */

```

MADUCS

Map a point from the UCS to the MODEL or WORK system.

Syntax:

```

/*- - - - - */
void maducs ( mode, ucsid, pu, pnew, res)
/*- - - - - */
          /*
          /* Input :
          /*
int      *mode;    /* 1 = to mode, 2 = to work
int      *ucsid;   /* The ID of the UCS.
double   pu[];     /* 3D point in UCS coordinates
          /*
          /* Output :
          /*
double   pnew[];   /* 3D point in MODEL system
int      *res;     /* 0 = O.K. otherwise: ERROR
/*- - - - - */

```

MAPM2U

Map a point from the Model coordinate system to a UCS.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MAPM2U ( Ucsid, Pm, Pu, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of the UCS.
/* The coordinates of the point to be mapped, in the
/* Model coordinate system.
/*
/* Output :
/*
/* Mapped point coordinates in the specified UCS.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MAPU2M

Map a point from a specified UCS to the Model system.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MAPU2M ( Ucsid, Pu, Pm, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of the UCS.
/* The coordinates of the point to be mapped, in the
/* specified UCS.
/*
/* Output :
/*
/* Mapped point coordinates in the Model system.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MAPU2W

Map a point from a selected UCS to the Work system.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void MAPU2W ( Ucsid, Pu, Pw, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of the UCS.
/* Coordinates of point to map, in the specified UCS.
/*
/* Output :
/*
/* Coordinates of the point in the Work system.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

MAPW2U

Map a point from the Work system to a specified UCS.

Syntax :

```

/*- - - - - */
void MAPW2U ( Ucsid, Pw, Pu, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the UCS.
/* Coordinates of point to map, in the work system.
/*
/* Output :
/*
/* Coordinates of the mapped point in the UCS.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```

NAM2ID

Given a UCS name, retrieve the UCS ID.

Syntax :

```

/*- - - - - */
void NAM2ID ( Ucsnam, Id, Status )
/*- - - - - */
/*
/* Input :
/*
/* Name of the UCS.
/*
/* Output :
/*
/* ID of the UCS.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```

UCSMAT

Create a matrix given 3 points for UCS purposes.

Syntax :

```

/*- - - - - */
void UCSMAT ( P1, P2, P3, Mat, Status )
/*- - - - - */
/*
/* Input :
/*
/* Origin point.
/* X direction.
/* Y direction.
/*
/* Output :
/*
/* Transformation matrix.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```



CDK_VERIFY_ENTITY

Obtain the class and type of an entity.

Syntax :

```
/*- - - - - */
int CDK_VERIFY_ENTITY( Id, Cl, Ty )
/*- - - - - */
/*
/* Input :
/*
int    Id;      /* ID of an entity.
/*
/* Output :
/*
int    *Cl;     /* The Class of the entity.
int    *Ty;     /* The Type of the entity.
/* Return :
/* See General Concepts-Status on page 1-2.
/*- - - - - */
```

Note : - See Classes and Types on page 1-1.

OBCRCL

Obtain arc/circle data.

Syntax :

```
/*- - - - - */
void OBCRCL ( Idc, Mode, Radius, Center, Start, Delta, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idc;    /* ID of the arc/circle.
int    *Mode;   /* 1 = Model.
/*           /* 2 = Work.
/*
/* Output :
/*
float  *Radius; /* The radius of the arc/circle.
float  Center[3]; /* The coordinates of the center point:
/*           /* coordinate.CENTER( 1 ) = X
/*           /* coordinate.CENTER( 2 ) = Y coordinate.CENTER( 3 ) = Z
/*           /* coordinate.
float  *Start;  /* Start angle in degrees.
float  *Delta;  /* Size of the arc/circle in degrees, angle relative to
/*           /* START.
int    *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

Note : - START angle is given with reference to an X axis on the plane of the circle, and may therefore be irrelevant in some applications.

OBGROU

Obtain data about a master.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBGROU ( Idm, Max, Ids, Count, St, Tm, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of master.
/* Maximum number of IDs to put in IDS.
/*
/* Output :
/*
/* Array containing the IDs of the entities that make
/* up the master.
/* Number of entities in the master.
/* Master Status:
/* 0 = 3D Master.
/* 1 = 2D Master.
/* 10 = Group.
/* Transformation matrix of the master.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int *Idm;
int *Max;

int Ids[Max];
int *Count;
int *St;

float Tm[12];
int *Status;

```

OBINST

Obtain an instance data by its ID.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBINST ( Insid, Masid, Rotat, Base, Scale, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of the instance.
/*
/* Output :
/*
/* ID of its master.
/* Rotation matrix.
/* Base point.
/* Scaling factor.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int *Insid;

int *Masid;
float Rotat[9];
float Base[3];
float *Scale;
int *Status;

```

OBLINE

Obtain line data.

Syntax

```
/*- - - - - */
void OBLINE ( Idln, Mode, P1, P2, Length, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of line.
/* 1 = Model.
/* 2 = Work.
/*
/* Output :
/*
/* Coordinates of 1st endpoint.
/* Coordinates of 2nd endpoint.
/* Length of line ( true length or length of projection
/* on the work plane, according to MODE ).
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

int *Idln;
int *Mode;

float P1[3];
float P2[3];
float *Length;

int *Status;
```

OBPNT

Obtain point data.

Syntax :

```
/*- - - - - */
void OBPNT ( Idpt, Mode, P, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of point.
/* 1 = Model.
/* 2 = Work.
/*
/* Output :
/*
/* Coordinates of point.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

int *Idpt;
int *Mode;

float P[3];
int *Status;
```

OBTUCS

Obtain information on the UCS.

Syntax :

```
/*- - - - - */
void OBTUCS ( Id, Mat, Orig, Scl, Ucsnam, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the UCS.
/*
/* Output :
/*
/* Transformation matrix.
/* Origin point of the new UCS.
/* Scale.
/* The UCS name.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

int *Id;

float Mat[9];
float Orig[3];
float *Scl;
char *Ucsnam;
int *Status;
```

UGTMNM

Obtain the name of a master.

Syntax :

```

/*- - - - - */
void UGTMMN ( Mstrid, Mtype, Fnlen, Fname, Mnlen, Mname, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Mstrid;    /* ID of the master.
/*
/* Output :
/*
int    *Mtype;    /* Type of master:
/*      1 = Internal master.
/*      2 = External master.
/*      3 = External sub-view.
/*      4 = Sub-assembly.
/*      5 = Internal sub-view.
/*
/* Input/Output :
/*
int    *Fnlen;    /* Maximum number of characters in the file name.
/*      Length of FNAME.
/*
/* Output :
/*
char    *Fname;    /* File name.
/*
/* Input/Output :
/*
int    *Mnlen;    /* Maximum number of characters in the master name.
/*      Length of MNAME.
/*
/* Output :
/*
char    *Mname;    /* Master name.
int    *Status;    /*      0 = O.K.
/*      -1 = Error.
/*      -2 = Master table not found.
/*- - - - - */

```

UODCIR

Obtain circle data in double.

Syntax :

```

/*- - - - - */
void UODCIR ( Id, Radius, Center, Start, Delta, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Id;        /* ID of entity.
/*
/* Output :
/*
double *Radius;    /* Radius of entity.
double Center[3];  /* Center coordinates of entity.
double *Start;     /* Start degree of entity.
double *Delta;     /* Delta degree of entity.
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UODCON

Obtain conic data in double.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UODCON ( Idc, Rotang, Aaxis, Baxis, Start, Delta, Subtyp, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Idc; /* ID of entity
/*
/* Output :
/*
double *Rotang; /* Rotation angle of major symmetry axis.
double *Aaxis; /* Half major axis.
double *Baxis; /* Half minor axis.
double *Start; /* Start degree of entity.
double *Delta; /* Delta degree of entity.
int *Subtyp; /* 0 = Circle.
/* 1 = Ellipse.
/* 2 = X-axis hyperbola.
/* 3 = Y-axis hyperbola.
/* 4 = X-axis parabola.
/* 5 = Y-axis parabola.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

UODHLX

Obtain helix data in double.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UODHLX ( Id, Heltyp, Bp, Thread, Pitch, Srad, Erad, Step, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Id; /* ID of entity.
/*
/* Output :
/*
int *Heltyp; /* Subtype of helix:
/* 1 = Fixed radius.
/* 2 = Linear varied radius.
/* 3 = Exponential varied radius.
double Bp[3]; /* Base point ( in model coordinates ).
double *Thread; /* Number of complete revolutions of the thread ( e.g.
/* 3, 5.5 etc. ).
double *Pitch; /* Distance between each thread.
double *Srad; /* Radius. If HELTYP = 3, this variable signifies the
/* start radius.
double *Erad; /* End radius. Only valid if HELTYP = 3.
double *Step; /* Difference in radius between each thread. Only valid
/* if HELTYP = 2.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

UODLIN

Obtain line model coordinates in double.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UODLIN ( Id, Point1, Point2, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of entity.
/*
/* Output :
/*
/* Model endpoint coordinates.
double Point1[3];
/* Model endpoint coordinates.
double Point2[3];
/* See General Concepts-Status on page 1-2.
int *Status;
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UODPNT

Obtain point model coordinates in double.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UODPNT ( Id, Point, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of entity.
/*
/* Output :
/*
/* Error point coordinates.
double Point[3];
/* See General Concepts-Status on page 1-2.
int *Status;
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UODPRJ

Obtain the projected matrix of an entity.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UODPRJ ( Id, Mat, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of entity.
/*
/* Output :
/*
/* Projected matrix.
double Mat[9];
/* See General Concepts-Status on page 1-2.
int *Status;
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UODSPL

Obtain spline data in double.

Syntax :

```

/*- - - - - */
void UODSPL ( Id, From, Num, Buf, Spltyp, Dim, Status )
/*- - - - - */
/*
/* Input :
/*
int *Id; /* ID of entity.
int *From; /* The index of a point from which to retrieve
/* information.
int *Num; /* The number of points to retrieve.
/*
/* Output :
/*
double *Buf; /* The coordinates of the control points.
int *Spltyp; /* Type:
/* 1 = CUBIC.
/* 2 = BEZIER.
/* 3 = B SPLINE.
/* 4 = VDA.
int *Dim; /* Spline dimension:
/* 2 = 2D.
/* 3 = 3D.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```



CDK_WPGETD

Obtain the coefficients of the work plane in the equation $Ax+By+Cz+D=0$. Type: Double.

Syntax :

```

/*- - - - - */
void CDK_WPGETD( A, B, C, D )
/*- - - - - */
        /*
        /* Output:
        /*
double *A;      /*
double *B;      /*
double *C;      /*
double *D;      /*
/*- - - - - */

```

CDK_WRKPLN_3PNT

Set a new work plane, defining as passing through 3 points.

Syntax :

```

/*- - - - - */
void CDK_WRKPLN_3PNT ( Point1, Point2, Point3, Status )
/*- - - - - */
        /*
        /* Input :
        /*
double Point1[3]; /* The coordinates of the 1 point ( MODEL ).
double Point2[3]; /* The coordinates of the 2 point ( MODEL ).
double Point3[3]; /* The coordinates of the 3 point ( MODEL ).
        /*
        /* Output :
        /*
int *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UWP2CRV

Define a new working plane by planar curve or 2 lines.

Syntax :

```

/*- - - - - */
void UWP2CRV ( Id1, Id2, Status )
/*- - - - - */
        /*
        /* Input :
        /*
int *Id1;         /* ID of the planar curve or of the 1st line.
int *Id2;         /* ID of the 2nd line, ignored when Id1 is planar curve.
        /*
        /* Output :
        /*
int *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UWP3PTID

Set a new work plane, defining as passing through 3 points.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UWP3PTID ( Idpoint1, Idpoint2, Idpoint3, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Idpoint1;    /* The ID of the 1 point.
int    *Idpoint2;    /* The ID of the 2 point.
int    *Idpoint3;    /* The ID of the 3 point.
/*
/* Output :
/*
/*
int    *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UWPCOEF

Define a new working plane by 4 coefficients.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UWPCOEF ( Coefa, Coefb, Coefc, Coefd )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
double *Coefa;
double *Coefb;
double *Coefc;
double *Coefd;    /* The working plane coefficients.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UWPDSPL

Define a new working plane as parallel to the screen and passing through a given point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UWPDSPL ( Idpoint, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Idpoint;    /* ID of the given point.
/*
/* Output :
/*
/*
int    *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UWPLNPT Set a new work plane, defining as perpendicular to a line and passing through a point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void UWPLNPT ( Idline, Idpoint, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Idline;    /* ID of the given line.
int    *Idpoint;   /* ID of the given point.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

WORKDF Define a work plane interactively.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void WORKDF ( Opt, Wvptr, Mat, Res )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Opt;       /* Method to be used to define the work plane:
/*      1 - By curve( s )
/*      2 - By normal vector and point
/*      3 - By three points
/*      4 - By 4 coefficient
/*           in the equation Ax + By + Cz + D = 0
/*
/* Output :
/*
int    *Wvptr;     /* The pointer to the transformation matrix of the new
/*                  defined work coordinate system.
float   Mat[9];    /* The transformation matrix ( 3x3 ).
int     *Res;      /*      = 0: Work coordinate system was defined.
/*                  /*      = -1: Work coordinate system was not defined.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

WPCRVS

Set a new work plane defined by a planar curve or two lines.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void WPCRVS ( Id1, Id2, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Id1;    /* ID of planar curve or first line.
int    *Id2;    /* ID of second line ( ignored when ID1 is a planar
/* curve ).
/*
/* Output :
/*
/*
int    *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

WPGETObtain the coefficients of the work plane in the equation $Ax + By + Cz + D = 0$. Type: Float.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void WPGET ( A, B, C, D )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/*
float  *A;
float  *B;
float  *C;
float  *D;
/* The work plane coefficients.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

WPLNPT

Set a new work plane, defined as perpendicular to a line and passing through a point.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void WPLNPT ( Id1, Id2, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Id1;    /* ID of line.
int    *Id2;    /* ID of point.
/*
/* Output :
/*
/*
int    *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```





Chapter 5

Drafting Routines

Introduction

This chapter contains user routines associated with drafting activities within Cimatron^{it}. Each routine appears under an appropriate subsection that logically corresponds to the interactive functions of the system.

The subsections contained in this chapter are as follows:

ANGULAR DIMENSION	Dimension entities for angles.
CENTER LINES	Create center lines of an arc/circle.
CHAMFER DIMENSION	Chamfer dimension entities.
DIMENSION PARAMETERS	Set global parameter values for dimensions.
REFERENCE GEOMETRY	Geometric entities/points.
GEOTOL	Create and access Geometrical Tolerance data.
HATCH	Hatching.
IDNUM	Create and access ID numbers.
LABEL	Create labels.
LINEAR DIMENSION	Creation of linear dimensions.
NOTE	Create and position text.
ORDINATE DIMENSION	Creation of ordinate dimensions.
RADIAL DIMENSION	Creation of radial dimensions.
STYLE	Manipulation of styles that are applied to text entities (NOTE, LABEL, etc.)
VIEWS	Routines associated with views.

- Note:**
- Some of the routines in the **GEOTOL** and **LABEL** subsections, refer to file **cdk_type.h**.
This file defines the types used in CimaDEK and can be found in **<root_cad>linc**.

ANGULAR DIMENSION

General Description

Before using any function from the Drafting package, the function **DIGLMD** must be called. This function initializes all global drafting parameters.

A number of functions refer to quadrants. Quadrants are arranged according to the figure below.

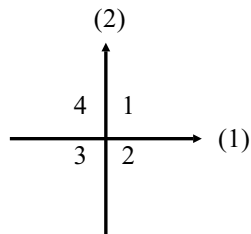


Figure 5-1: Quadrants

A number of functions refer to drafting symbols. See the table below:

1	α
2	β
3	+
5	\diamond
11	I
16	\emptyset
20	\square
22	$\sqrt{\quad}$
25	V
27	X
29	\angle
30	\pm
127	\circ

Table 5-1: Drafting Symbols

DIAN1A

Create a dimension entity for an angle defined by a line and an axis.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIAN1A ( Idline, Vertex, Axis, QudrntIdptex, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Idline;    /* ID of the line.
char    *Vertex;    /* Choose one of the following ways to indicate which
/* endpoint of the line is the vertex of the angle
/* being measured:
/* XLARGE
/*    The point where X is largest.
/* XSMALL
/*    The point where X is smallest.
/* YLARGE
/*    The point where Y is largest.
/* YSMALL
/*    The point where Y is smallest.
int    *Axis;    /* 1 : X axis.
/* 2 : Y axis.
int    *Qudrnt;    /* Quadrant number in which the angle is found and in
/* which the cross product of vectors IDLINE and AXIS
/* is greater than zero.
/* 1 : The first quadrant.
/* The quadrant between the line and the selected axis
/* in the direction indicated by the line.
/* 2, 3, or 4 : The remaining quadrants.
/* The quadrants are numbered in a clockwise direction,
/* see Quadrant Figure on page 5-2.
int    *Idptex;    /* ID of point at which the text will be positioned.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created dimension entity for an angle
/* ( if STATUS=0 ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Keeps associativity with IDLINE.

DIAN2A

Create a dimension entity for an angle defined by two lines.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIAN2A ( Idlin1, Idlin2, Qudrnt, IdptexStatus )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
int    *Idlin1;    /* ID of 1st line.
int    *Idlin2;    /* ID of 2nd line.
int    *Qudrnt;    /* Quadrant number that agrees with the condition that
/* the cross product of vectors IDLIN1 and IDLIN2 is
/* greater than zero.
/*      1 : The first quadrant.
/* The quadrant between the line and the selected axis
/* in the direction indicated by the line.
/*      2, 3, or 4 : The remaining quadrants.
/* The quadrants are numbered in a clockwise direction,
/* see Quadrant Figure on page 5-2.
int    *Idptex;    /* ID of point at which the text will be positioned.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created dimension entity for an angle ( if
/* STATUS=0 ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Keeps associativity with IDLINE1 and IDLIN2.

DIAN3A

Create a dimension entity for an angle defined by three points.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIAN3A ( Idvtx, Ptvtx, Idpt1, Pt1, Idpt2, Pt2Qudrnt, Idptex, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Idvtx; /* ID of the geometric point associated with the vertex */
/* point.
float Ptvtx[2]; /* Coordinates of the vertex point in the WORK */
/* coordinate system.
int *Idpt1; /* ID of the geometric point associated with the 1st */
/* edge point.
float Pt1[2]; /* Coordinates of the 1st point in the WORK coordinate */
/* system.
int *Idpt2; /* ID of the geometric point associated with the 2nd */
/* edge point.
float Pt2[2]; /* Coordinates of the 2nd point in the WORK coordinate */
/* system.
int *Qudrnt; /* Quadrant number that agrees with the condition that */
/* the cross product of vectors VECTOR1 and VECTOR2 is */
/* greater than zero (see diagram). VECTOR1 is from IDCT */
/* to IDP1. VECTOR2 is defined from IDCT to IDP2. When */
/* VECTOR1 * VECTOR2 > 0. ( cross product )
/* 1 : The first quadrant.
/* The quadrant between the line and the selected axis */
/* in the direction indicated by the line.
/* 2, 3, or 4 : The remaining quadrants.
/* The quadrants are numbered in a clockwise */
/* direction. When VECTOR1 * VECTOR2 > 0. ( cross */
/* product ). See Quadrant Figure on page 5-2.
int *Idptex; /* ID of point at which the text will be positioned. */
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2. */
/*
/* Returns :
/* ID of the created dimension entity for an angle ( if */
/* STATUS=0 ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - Keeps associativity with IDVTX, IDPT1 and IDPT2.

DIANEX

Extract all specific modals for angular dimensions.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DIANEX ( Id, Modnam, Type, Modval, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int      *Id;                   /* ID of the dimension entity.
char     *Modnam;               /* Parameter name.
char     *Type;                /* Parameter type:
                                /* C - character, I - integer, R - real.
                                /*
                                /* Output :
                                /*
int      *Modval;               /* Parameter value.
int      *Status;              /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
Note:      - The following MODNAMs and TYPEs are available:

```

MODNAM	TYPE
"CENTERED"	"I"
"HORIZONTAL"	"I"
"LEADER"	"I"
"SECTOR"	"I"
"WITHIN "	"I"

DIANG1

Create a dimension entity for an angle defined by a line and an axis.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIANG1 ( Idline, Vertex, Axis, QudrntIdptex, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Idline;    /* ID of the line.
char    *Vertex;    /* Choose one of the following ways to indicate which
/* endpoint of the line is the vertex of the angle
/* being measured:
/* XLARGE
/*     The point where X is largest.
/* XSMALL
/*     The point where X is smallest.
/* YLARGE
/*     The point where Y is largest.
/* YSMALL
/*     The point where Y is smallest.
int    *Axis;    /* 1 : X axis.
/*              /* 2 : Y axis.
int    *Qudrnt;    /* Quadrant number in which the angle is found and in
/* which the cross product of vectors IDLINE and AXIS
/* is greater than zero.
/* 1 : The first quadrant.
/* The quadrant between the line and the selected axis
/* in the direction indicated by the line.
/* 2, 3, or 4 : The remaining quadrants.
/* The quadrants are numbered in a clockwise direction,
/* see Quadrant Figure on page 5-2.
int    *Idptex;    /* ID of point at which the text will be positioned.
/*
/* Output :
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created dimension entity for an angle
/* ( if STATUS=0 ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DIANG2

Create a dimension entity for an angle defined by two lines.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIANG2 ( Idlin1, Idlin2, Qudrnt, Idptex, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of 1st line.
/* ID of 2nd line.
/* Quadrant number that agrees with the condition that
/* the cross product of vectors IDLIN1 and IDLIN2 is
/* greater than zero.
/* 1 : The first quadrant.
/* The quadrant between the line and the selected axis
/* in the direction indicated by the line.
/* 2, 3, or 4 : The remaining quadrants.
/* The quadrants are numbered in a clockwise direction,
/* see Quadrant Figure on page 5-2.
/* ID of point at which the text will be positioned.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created dimension entity for an angle ( if
/* STATUS=0 ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DIANG3

Create a dimension entity for an angle defined by three points.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIANG3 ( Idct, Idp1, Idp2, Qudrnt, Idptex, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of vertex point of angle.
/* ID of 1st edge point.
/* ID of 2nd edge point.
/* Quadrant number that agrees with the condition that
/* the cross product of vectors VECTOR1 and VECTOR2 is
/* greater than zero ( see diagram ).
/* VECTOR1 is from IDCT to IDP1.
/* VECTOR2 is defined from IDCT to IDP2.
/* When VECTOR1 * VECTOR2 > 0. ( cross product )
/* 1 : The first quadrant.
/* The quadrant between the line and the selected axis
/* in the direction indicated by the line.
/* 2, 3, or 4 : The remaining quadrants.
/* The quadrants are numbered in a clockwise direction,
/* see Quadrant Figure on page 5-2.
/* When VECTOR1 * VECTOR2 > 0. ( cross product )
/* ID of point at which the text will be positioned.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created dimension entity for an angle.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DIANPR

Set specific modal values for dimension entities of angles. Refer to the Cimatron^{it} **Drafting Manual**, for explanations of the modal parameters.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DIANPR ( Stt, Ut, Lt, Witn, Inout, SmlbgHorinc, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Stt;      /* Status of tolerance:
/*      1 : NO-T.
/*      2 : UNI-T.
/*      3 : BI-T.
/*      4 : SNG-T.
float  *Ut;      /* Upper tolerance.
float  *Lt;      /* Lower tolerance.
int    *Witn;    /* Witness lines:
/*      1 : no.
/*      2 : 1st.
/*      3 : 2nd.
/*      4 : both.
int    *Inout;   /* 1 : <> Arrows inside witness lines.
/*      2 : >< Arrows outside witness lines.
int    *Smlbg;   /* 1 : Quadrant angle.
/*      2 : Complement to 360.
int    *Horinc;  /* 1 : Horizontal.
/*      2 : Inclined text.
/*
/* Output :
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DIARPA

Set all specific modals for angular dimensions.

Syntax :

```

/*- - - - - */
void DIARPA ( Modnam, Type, Modval, Status )
/*- - - - - */
/*
/* Input :
/*
char *Modnam; /* Parameter name.
char *Type; /* Parameter type:
/* C - character, I - integer, R - float.
int *Modval; /* Parameter value.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note: - The following MODNAMs and TYPEs are available:

MODNAM	TYPE	available values		default values
“CENTERED”	“I”	1-INDICATE	2-CENTERED	1
“HORIZONTAL”	“I”	0-INCLINED	1-HORIZONTAL	1
“LEADER ”	“I”	0-no	1-yes(if not horizont)	0
“SECTOR ”	“I”	0-complement	1-sector	1
“WITHIN ”	“I”	0-no	1-within	depends on drafting standard



CENTER LINES

General Description

The following routine is used to create the center lines of an arc or circle at a specified angle to the horizontal axis.

CEN2LN

Create center lines.

Syntax :

```

/*- - - - - */
void CEN2LN ( Idc, Angle, Id, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the circle/arc.
/* Angle between a center line and the horizontal axis.
/*
/* Output :
/*
/* Array containing IDs of the newly created center
/* lines.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```



CHAMFER DIMENSION

DICHEX

Extract all the specific modals for chamfer dimension entities.

Syntax :

```

/*- - - - - */
void DICHEX ( Id, Modnam, Type, Modval, Status )
/*- - - - - */
/*
/* Input :
/*
int *Id; /* ID of dimension entity.
char *Modnam; /* Parameter name.
char *Type; /* Parameter type.
/*
/* Output :
/*
int *Modval; /* Parameter value.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
Note: - The following MODNAMs and TYPEs are available:

```

MODNAM	TYPE
"MIDDLE "	"I"
"TEXTHOR "	"I"
"TORH "	"I"
"WITHIN "	"I"

DICHMA

Create a chamfer dimension entity.

Syntax :

```

/*- - - - - */
int DICHMA ( Idchm, Idptex, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idchm; /* ID of the chamfer line.
int *Idptex; /* ID of the text positioning point.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the newly created dimension entity ( if STATUS
/* = 0 ).
/*- - - - - */
Note : - Keeps associativity with IDVCHM.

```

DICHPA

Set all the specific modals for chamfer dimension entities.

Syntax :

```

/*- - - - - */
void DICHPA ( Modnam, Type, Modval, Status )
/*- - - - - */
/*
/* Input :
/*
char *Modnam; /* parameter name
char *Type; /* parameter type
int *Modval; /* parameter value
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
Note: - The following MODNAMs and TYPEs are available:

```

MODNAM	TYPE	available values	default values
“MIDDLE ”	“I”	0 .. 1 0 - free 1 - middle	0
“TEXTHOR ”	“I”	0 .. 1 0 - no 1 - horizontal	0
“TORH ”	“I”	1 .. 2 1 - to LEFT 2 - to RIGHT	2
“WITHIN ”	“I”	0 .. 1 0 - no 1 - within	depends on drafting standard



DIMENSION PARAMETERS

General Description

The following functions are used to set new global parameter values for dimensions. This setting of new values must be performed at least once in every program using angular, linear, ordinate or radial dimensions.

When using the Cimatron^{it} **Drafting Application**, the function DRAF_PAR sets the global parameters. However, when using **CimaDEK**, these values are ignored and new values must be set using one of the following functions. This procedure must be completed before attempting to use any subroutine that deals with dimension values of any kind.

Calling the function DIGLMD (that takes no parameter values) will produce the same values defined in the function DRAF_PAR by the interactive system. Calling DIGLPR allows you to input new parameter values via its arguments.

DIEXTR

Extract the non-geometrical data of a dimension from the database to the memory structures.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DIEXTR ( Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of dimension entity.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

DIGLEX

Extract all the global modals for dimension entities of all kinds.

Syntax :

```

/*
void DIGLEX ( Id, Modnam, Type, Nch, Modval, Status )
/*
/*
/* Input :
/*
int *Id; /* ID of dimension entity.
char *Modnam; /* Parameter name.
char *Type; /* Parameter type.
/*
/* Output :
/*
/*
int *Nch; /* number of chars if extract value of character type
char *Modval; /* parameter value
int *Status; /* See General Concepts-Status on page 1-2.
/*
Note: - The following MODNAMs and TYPEs are available:

```

MODNAM	TYPE
"ARROW1 "	I
"ARROW2 "	I
"CHARSIZE"	R
"DIMPRA "	I
"DIMPRL "	I
"DIMVALUE"	R
"HOLECODE"	C
"INSIDE "	I
"MANUAL "	I
"NAMETXT "	C
"NUMERIC "	I
"POSTFIX "	C
"POSTSYM "	C
"PREFIX "	C
"PREFSYM "	C
"SHAFTCOD"	C
"SINGLEA "	I
"SINGLEL "	I
"STANDARD"	I
"TOLA "	I
"TOLL "	I
"TOLLOA "	R

"TOLLLOL "	R
"TOLNTYPE"	I
"TOLPRA "	I
"TOLPRL "	I
"TOLUPA "	R
"TOLUPL "	R
"UNDERFIX"	C
"UNITA "	I
"UNITL "	I
"WITGAP "	R
"WITNESS1"	I
"WITNESS2"	I
"WITOVERS"	R

DIGLMD

Accept the modal values of the interactive system set by the DRAF_PAR function as the global parameters for the Drafting functions in **CimaDEK**.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - */
void DIGLMD ( void )
/*- - - - - - - - - - - - - - - - - - - - */
```

DIGLPA

Set all the global modals for dimension entities of all kinds.

Syntax :

```

/*_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ */
void DIGLPA ( Modnam, Type, Nch, Modval, Status )
/*_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ */
                                /*
                                /* Input :
                                /*
char    *Modnam;                /* Parameter name.
char    *Type;                 /* Parameter type.
int     *Nch;                  /* Number of chars ( in use only if value of character
                                /* type ).
char    *Modval;               /* Parameter value.
                                /*
                                /*
                                /* Output :
                                /*
int     *Status;               /* See General Concepts-Status on page 1-2.
/*_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ */
Note:   - The following MODNAMs and TYPEs are available:

```

MODNAM	TYPE	available values	
“ARROW1 ”	“T”	-1 .. 6	TERMINATOR (left or down)
		NO_TERM	-1
		STD_TERM	0
		ARROW	1
		SLASH	2
		CIRCLE	3
		OPEN_ARROW	4
		FILLED_ARROW	5
		FILLED_CIRCLE	6
“ARROW2 ”	“T”	-1 .. 6	TERMINATOR (right or up)
		NO_TERM	-1
		STD_TERM	0
		ARROW	1
		SLASH	2
		CIRCLE	3
		OPEN_ARROW	4
		FILLED_ARROW	5
		FILLED_CIRCLE	6

MODNAM	TYPE	available values	
"DIMPRA "	"I"	0 .. 2(4) 0 - DIM DEG 1 - DIM. DEG.MIN 2 - DIM. DEG.MIN.SEC 0 - DIM 1 - DIM.# .. 4 - DIM.####	DECIMAL PLACES DEGREES(DEC) DECIMAL PLACES DEG_MIN
"DIMPRL "	"I"	0 .. 4 (7) 0 - DIM 1 - DIM.# .. 4 - DIM.#### 0 - DIM #/1 DECIMAL PLACES 1 - DIM #/2 .. 7 - DIM #/128	DECIMAL PLACES Case fraction of foot/inch
"DIMVALUE"	"R"		
"HOLECODE"	"C"	NCH=4	Shaft code lower case letters followed by a digit
"INSIDE "	"I"	0 .. 1 0 - OUT 1 - IN	-><- <->
"MANUAL "	"I"	0 .. 1 0 - AUTOMATIC 1 - MANUAL	
"NAMETXT "	"C"	NCH<=20	Dimension text case textual dimension
"NUMERIC "	"I"	0 .. 2 0 - NUMERIC 2 - TEXTUAL	
"POSTFIX "	"C"	NCH<=10	Symbol follows the dimension

MODNAM	TYPE	available values	
"POSTSYM "	"C"	NCH<=20	Text follows the dimension
"PREFIX "	"C"	NCH<=20	Text precedes the dimension
"PREFSYM "	"C"	NCH<=10	Symbol precedes the dimension
"SINGLEA "	"I"	0 .. 2	Second unit for angular double units
		0 - No second unit	
		1 - Degree	
		2 - angular units	
"SINGLEL "	"I"	0 .. 7	Second unit for length double units
		0 - No second unit	
		1 - Millimeter	
		2 - Centimeter	
		3 - Inch	
		4 - Fractional Inch	
		5 - Feet	
		6 - Fractional Feet	
		7 - Meter	
"STANDARD"	"I"	No of active standard (no=STDNID())	
"TOLA "	"I"		Angular tolerance
		0=No toler	
		1=Numeric	
		6=exact dim.	
"TOLL "	"I"	0=No toler	Linear tolerance
		1=Numeric	
		2=Coded	
		3=Both	
		6=exact dim.	
"TOLLOA "	"R"	-45 +45	Lower tolerance for ANGULAROL_LOA, TOL_UPA,
"TOLLOL "	"R"	-1000. +1000.	Lower tolerance for LINEAR

MODNAM	TYPE	available values	
"TOLNTYPE"	"I"	1 .. 4	Type of tolerance for NUMERIC only
		1 - Uni	
		2 - Bi	
		3 - Single	
"TOLPRA "	"I"	4 - Limits	Precision decimal places for tolerance decimal places DEG_MIN
		0 .. 2(4)	
		0 - DIM DEG	
		1 - DIM. DEG.MIN	
		2 - DIM. DEG.MIN.SEC	
		0 - DIM	
		1 - DIM.#	
		..	
		4 - DIM.####	
"TOLPRL "	"I"		DECIMAL PLACES
		0 - DIM,	
		1 - DIM.#	
		..	
		4 - DIM.####	
"TOLUPA "	"R"	-45 .. +45	Upper tolerance for ANGULAR (cases of BI, SINGLE, LIMITED)
		0.0 .. +45	Implicit upper tolerance for ANGULAR (case of UNI)
"TOLUPL "	"R"	-1000. .. +1000.	Upper tolerance for LINEAR (cases of BI, SINGLE, LIMITED)
		0.0 .. +1000.	Implicit upper tolerance for ANGULAR (case of UNI)

MODNAM	TYPE	available values	
"UNDERFIX"	"C"	NCH<=20	Text under the dimension
"UNITA "	"I"	1 .. 2	Units for ANGULAR DIMENSIONS
		1 - Degree	
		2 - angular units	
"UNITL "	"I"	1 .. 7	Units for LINEAR DIMENSIONS
		1 - Milimeter	
		2 - Centimeter	
		3 - Inch	
		4 - Fractional Inch	
		5 - Feet	
		6 - Fractional Feet	
		7 - Meter	
"WITGAP "	"R"	0 .. 1000	DIMENSION GAP
"WITNESS1"	"I"	0 .. 1	WITNESS LINE (left or down)
		0 - no	
		1 - yes	
"WITNESS2"	"I"	0 .. 1	Witness Line (right or up)
		0 - no	
		1 - yes	
"WITOVERS"	"R"	0 .. 1000	Dimension overshoot

DIGLPR

Set global parameters for the Drafting USER functions. Refer to the Cimatron^{it} **Drafting Manual**, for explanations of the modal parameters.

Syntax :

```

/*- - - - - */
void DIGLPR ( Arr, Un, Stndrd, Dpr, Tpr, Texth, Gap, OsStatus )
/*- - - - - */
/*
/* Input :
/*
int    *Arr;    /* Arrow type:
/*      1 : arrow.
/*      2 : slash.
/*      3 : circle.
int    *Un;    /* Unit of measure to be used:
/*      1 : MM.
/*      2 : CM.
/*      3 : INCH ( decimal notation ).
/*      4 : INCH ( fraction notation ).
/*      5 : FEET ( decimal notation ).
/*      6 : FEET ( fraction notation ).7 = METER.
int    *Stndrd; /* The standard obtained from a previous call to STDNID.
int    *Dpr;    /* Dimension precision:
/* Decimal precision = 0-4 precision digits.
/* Fraction notation = 1/2*n ( 0 n 7 ).
int    *Tpr;    /* Tolerance precision, same as for DPR, dimension
/* precision.
float  *Texth;  /* Height of the text ( TEXTH 0.0001 )
float  *Gap;    /* Gap of the dimension ( GAP 0. )
float  *Os;     /* Overshoot of the dimension ( OS 0. )
/*
/* Output :
/*
int    *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

DIGSNA

To get names of existing standards.

Syntax :

```

/*- - - - - */
void DIGSNA ( First, Num, Names, Status )
/*- - - - - */
/*
/* Input :
/*
int    *First;  /* The number of first standard for output.
/*
/* Input/Output :
/*
int    *Num;    /* The max number of standards for output.
/* The number of standards.
/*
/* Input :
/*
char    Names[NUM]; /* Array of standard names.
/*
/* Output :
/*
int    *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

DIMODI

Modify the non-geometrical data of a dimension with the data stored in the memory structures (the result of previous calls to DIEXTR and DIGLPA).

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIMODI ( Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Id;      /* ID of dimension entity.
/*
/* Output :
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* The ID of the modified dimension, if STATUS = 0.
/* See Chapter 1, General Concepts, Duplication of
/* Entities.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

STDNID

Given the name of a drafting standard, get its internal number in the standard table.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void STDNID ( Sname, Number )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char    *Sname;  /* Name of the drafting standard.
/*
/* Output :
/*
int     *Number; /* Internal number in the standard table.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



GEOMETRY

General Description

Geometric entities, associativity and geometric points.

The purpose of the geometric point entity is to store a record of how a certain point was obtained. So any entity storing the ID of a geometric point entity will be able to re-evaluate the current position of the geometric point.

Any entity storing the reference of a geometric point entity, stores the following data:

- The ID of the geometric point entity.
- The X, Y, (Z) coordinates returned by the last evaluation of this geometric point.

Associativity is supported *only* if the point is derived from one of the following options:

END
MID
CENTER
INTERSECTION
PICK

To obtain geometric point entities, the following routines are used:

PTASCE	Creates a point associated with the CENTER of an ARC/CIRCLE .
PTASEN	Creates a point associated with the END of a curve.
PTASID	Creates a point through interaction.
PTASIN	Creates a point associated with the point of INTERSECTION of two curves.
PTASMI	Creates a point associated with the MIDDLE of a curve.
PTASPI	Creates a point associated with the PICK option.

To obtain geometric entities for curves, the following routines are used:

CRVASO	Creates a geometric curve by its ID.
CRVASP	Creates a geometric curve interactively.

CRVASO

Create a user defined geometrical entity for a curve.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int CRVASO ( Nesting, Idc, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Nesting;    /* Depth of nesting ( currently not in use; should be
/* set to 1 ).
int    *Idc;        /* ID of the curve.
/*
/* Output :
/*
int    *Status;     /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created geometrical curve (if STATUS = 0).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CRVASP

Create a geometrical entity for a curve interactively.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int CRVASP ( Res )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int    *Res;        /* Users response.
/*      0 : PICK
/*     -1 : EXIT
/*     -2 : REJECT
/*    < -2 : Data Base error.
/*
/* Returns :
/* ID of the created geometrical curve ( if RES = 0 ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

PTASCE

Create a geometric point for the CENTER option.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int PTASCE ( Idarc, Center, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Idarc;      /* ID of the arc/circle.
/*
/* Output :
/*
float  *Center;     /* The center point in work coordinates.
int    *Status;     /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created geometric point ( if STATUS ==
/* OK ).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

PTASEN

Create a geometric point for the END option.

Syntax :

```

/*- - - - - */
int PTASEN ( Idcrv, SelectPoint, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the curve.
/* Choose one of the following ways to indicate which
/* endpoint should be chosen:
/* XLARGE
/* The point where X is largest.
/* XSMALL
/* The point where X is smallest.
/* YLARGE
/* The point where Y is largest.
/* YSMALL
/* The point where Y is smallest.
/*
/* Output :
/*
/* The point in work coordinates.
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created geometric point ( if STATUS ==
/* OK ).
/*- - - - - */

```

PTASID

Create a geometric point interactively.

Syntax :

```

/*- - - - - */
int PTASID ( Point, Option, Respon, Status )
/*- - - - - */
/*
/* Output :
/*
/* The point in work coordinates.
/*
/* Input/Output :
/*
/* Method to be used to define the point. See PTGET.
/* Option chosen (if RESPON = 0). May differ from the
/* input value if changed via the POINT submenu.
/*
/* Output :
/*
/* See Chapter 1, General Concepts, Mouse Response.
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created geometric point
/* ( if STATUS == OK ).
/* 0 : if STATUS == OK and RESPON==EXIT or REJECT
/* 0 : if STATUS == 3 - Cannot create a geometric
/* point
/* -1 : if STATUS == -2 /Database error/
/*- - - - - */

```

PTASIN

Create a geometric point for the INTERSECTION option.

Syntax :

```

/*- - - - - */
int PTASIN ( Idcrv1, Idcrv2, Select, Point, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the 1st curve.
/* ID of the 2nd curve.
/* Choose one of the following ways to indicate which
/* point should be chosen if more than one
/* intersection point exists:
/* XLARGE
/* The point where X is largest.
/* XSMALL
/* The point where X is smallest.
/* YLARGE
/* The point where Y is largest.
/* YSMALL
/* The point where Y is smallest.
/*
/* Output :
/*
/* The point in work coordinates.
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created geometric point
/* ( if STATUS == OK ).
/*- - - - - */

```

PTASMI

Create a geometric point for the MIDDLE option.

Syntax :

```

/*- - - - - */
int PTASMI ( Idcrv, Point, Status )
/*- - - - - */
/*
/* Input :
/*
/* ID of the curve.
/*
/* Output :
/*
/* The point in work coordinates.
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created geometric point
/* ( if STATUS == OK ).
/*- - - - - */

```

PTASPI

Create a geometric point for the PICK option.

Syntax :

```

/*- - - - - */
int PTASPI ( Idpnt, Point, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Idpnt;    /* ID of the picked point.
/*
/* Output :
/*
float   Point[3]; /* The point in work coordinates.
int     *Status;  /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created geometric point
/* ( if STATUS == OK ).
/*- - - - - */

```

□

GEOTOL

General Description

The following routines allow the user to create and access Geometrical Tolerance data.

CDK_GEOTOL_CREATE Create a geometrical tolerance.**Syntax :**

```

/*- - - - - */
int  CDK_GEOTOL_CREATE( SymbolSize, FramePos, HorAlignment, VerAlignment,
                        Connection, NumLeaderPts, Leader, Terminator,
                        AllAround, AssocType, RefGeometry, NumFrames, Frames )
/*- - - - - */

/*
/*  Input:
/*
/*  The text height.
/*  The frames position ( WORK ).
/*  1 = Left; 2 = Center; 3 = Right.
/*  1 = Top; 2 = Center; 3 = Bottom.
/*  The index of the connection point :
/*
/*      4          5          6
/*      \          |          /
/*      *---+ *          *          */
/*      | FRAME No 1 |
/*      |---+-----+
/*      3 - * FRAME No 2          * - 7
/*      |-----+---+
/*      | FRAME No 3          |
/*      *---*---+          *          */
/*      /          |          \
/*      2          1          0
/*
/*  The number of points
/*  in the leader definition sequence,
/*  including the end point ( <= 33 ).
/*
/*  The leader definition sequence:
/*  ( See type definition of CDKT_LEADER_NODE
/*  in cdk_type.h ).
/*
/*  The leader terminator type:
/*  -1 - without terminator
/*  1 - ARROW
/*  2 - SLASH
/*  3 - CIRCLE
/*  4 - OPEN_ARROW
/*  5 - FULL_ARROW
/*  6 - FULL_CIRCLE
/*  7 - TRIANGLE
/*  8 - FULL_TRIANGLE
/*
/*  1 = Set "all around" symbol,
/*  0 = Do not set it.
/*  0 = None, 1 = Line with witness,
/*  2 = Point, 3 = Curve without witness.
/*  if AssocType = 0 : 0;
/*  if AssocType = 1 : The ID of associated line.
/*  if AssocType = 2 : The ID of associated point.
/*  if AssocType = 3 : The ID of associated curve.
/*  The number of frames of the creating entity
/*  ( <= 10 ).
/*
/*  ( See type definition of CDKT_GEOTOL_FRAME
/*  in cdk_type.h ).
/*
/*  Returns:      > 0 = ID of the created entity,
/*               < 0 = See Chapter 1, General
/*               Concept, Status.
/*
/*- - - - - */

```

CDK GEOTOL GET

Obtain the geometrical tolerance data.

Syntax :

```

/*----- */
int  CDK_GEOTOL_GET( IdGeotol, SymbolSize, TextStart, NumLeaderPts, Leader,
                    Terminator, AllAround, AssocType, RefGeometry,
                    NumFrames, Frames )

/*----- */
/*
/* Input:
/* The ID of the geotol entity.
/*
/* Output:
/*
/* The text height.
/* The text start position ( WORK ).
/* The number of points.
/*
/* The leader definition sequence:
/* ( See type definition of CDKT_LEADER_NODE
/* in cdk_type.h ).
/* The leader terminator symbol:
/* -1 - without terminator
/* 1 - ARROW
/* 2 - SLASH
/* 3 - CIRCLE
/* 4 - OPEN_ARROW
/* 5 - FULL_ARROW
/* 6 - FULL_CIRCLE
/* 7 - TRIANGLE
/* 8 - FULL_TRIANGLE
/*
/* 1 = "All around" symbol is set,
/* 0 = "All around" symbol is not set.
/* 0 = None, 1 = Line with witness,
/* 2 = Point, 3 = Curve without witness.
/*
/* if AssocType = 0 : 0;
/* if AssocType = 1 : The ID of associated line.
/* if AssocType = 2 : The ID of associated point.
/* if AssocType = 3 : The ID of associated curve.
/* The number of frames in creating entity.
/* Array of frame types:
/* 1 = geotol, 2 = datum.
/* Array of tolerance type - see tolerance type.
/*
/* ( See type definition of CDKT_GEOTOL_FRAME
/* in cdk_type.h ).
/*
/* Returns:      = 0 : OK.
/*              < 0 = See Chapter 1, General
/*                  Concept, Status.
/*----- */

```

CDK_GEOTOL_SET

Sets variables for the geometrical tolerance data.

Syntax :

```

/*- - - - - */
int    CDK_GEOTOL_SET ( Frames, SetFrames, FrameType, TolType, TolPrec,
                      Prefix, TolModifier, Postfix, TolValue, NumDatums,
                      Datums, DatModifiers, Identifier, Dashes )
/*- - - - - */
/*
/*  Input:
/*
/*  Number of frames ( must be < 0 ).
/*
/*  frame buffer :
/*  ( see definition of CDKT_GEOTOL_FRAME
/*  in cdk_type.h ).
/*
/*  Array of frame types:
/*  1: geotol      2: datum.
/*
/*  Array of tolerance type - see tolerance type.
/*
/*  Array of tolerance precision values ( 0 - 4 ).
/*
/*  Array of prefix : 0: no prefix 1: radial.
/*
/*  Array of tolerance modifiers :
/*  0: non; 1: 'M'; 2: 'L'; 3: 'P'; 4: 'S'
/*
/*  Array of tolerance postfix strings.
/*  Each string max 20 characters.
/*
/*  Array of tolerance values.
/*
/*  Array with numbers of datums:
/*  no_datum, 1 datum, 2 datums, 3 datums.
/*
/*  Array of datum names. Frames * 3.
/*
/*  Array of modifiers. Frames * 3.
/*
/*  Datum symbol.
/*
/*  Dashes flag :
/*  1 = datum with dashes, 0 = without.
/*
/*
/*
/*  Returns:      > 0 = ID of the created entity,
/*               < 0 = See Chapter 1, General
/*               Concept, Status.
/*
/*- - - - - */

```

Note :

- This function must be used, at least once, before using "CDK_GEOTOL_CREATE".
- All parameters, except "Frames" and "SetFrames", can have a "NULL" value to set default values, and -100 to keep values unchanged.



HATCH

General Description

The following integer function performs “hatching”, and corresponds to the RANDOM option of the HATCH function in the interactive system.

DHATCH

Create hatching. Refer to the Cimatron^{it} **Drafting Manual**, for explanations of the parameters.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DHATCH ( N, Ids, Ang, Space, Matrl, Forp, IdpStatus )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *N;          /* Number of curves in IDS.
int Ids[N];       /* Array of N IDs of curves defining the contours. (The
/* curves may be in any order and may include islands.)
/* The curves defining the contours must meet at their
/* endpoints, and no overlapping of lines is
/* permitted.
float *Ang;       /* Angle between a HATCH line and the X axis of the
/* work plane.
float *Space;     /* Spacing between HATCH lines.
int *Matrl;       /* Material to be represented by HATCH lines:
/*      1 : general.
/*      2 : steel.
/*      3 : brass.
/*      4 : plastic.
/*      5 : glass.
int *Forp;        /*      1 : free.
/*      2 : through a point.
int *Idp;         /* ID of the point if FORP = 2.
/*
/* Output :
/*
int *Status;      /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created HATCH entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



General Description

The following routines allow the user to create and access ID numbers.

CDK_IDNUM_GSC

Get/Set/Create the graphical part of ID_NUM.

Syntax :

```

/*- - - - - */
void CDK_IDNUM_GSC ( Mode, IdNum, TextPos, Text, TextSize, TargetPoint,
                    TerminatorType, FigureType, RadiusFactor, Pen, Status )
/*- - - - - */
/*
/* Input :
/*
int      *Mode;      /* 1 = GET, 2 = SET, 3 = CREATE
/*
/* Input/Output :
/*
int      *IdNum;      /* idnum ID
float     TextPos[3];  /* text position
char      Text;        /* text
float     *TextSize;   /* text height
float     TargetPoint[3]; /* target point
int       *TerminatorType; /* leader terminator type :
/*      -1 - without terminator
/*      1 - ARROW
/*      2 - SLASH
/*      3 - CIRCLE
/*      4 - OPEN_ARROW
/*      5 - FULL_ARROW
/*      6 - FULL_CIRCLE
/*      7 - TRIANGLE
/*      8 - FULL_TRIANGLE
int       *FigureType; /* figure type :
/*      0 - without figure
/*      3 - CIRCLE
/*      4 - TRIANGLE
/*      5 - RECTANGLE
/*      6 - HEXAGON
/*      7 - UNDERLINE
/*
float     *RadiusFactor; /* radius factor.
int       *Pen;          /* text pen ( 1 - 8 ).
/*
/* Output :
/*
int       *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - */

```



General Description

The following routines allow the user to create labels. The routines are written in the C language.

Using Label Routines

Carry out the following process when using these routines:

1. To change the active font, use function CDK_FONT_SET_ACTN. (See NOTE routines).
2. To create a label, use function CDK_LABEL_CREATE.
Default values are defined (scheme, symbol size, text direction).
3. To change the default values, use function CDK_LABEL_SET.
4. To obtain the size of the text box, use function CDK_NOTE_DIMENSIONS.

SCHEME

Use SCHEME to change the default characters that are produced from the keyboard. Each scheme file defines a different set of characters, (Japanese, Hebrew, etc.). The scheme files are found in the directory **font**

The default scheme is **tright**. The scheme file extension is **.cdt**.

CDK_LABEL_CONNECTION

Edit connection between the leader and label frame.

Syntax :

```
/*- - - - - */
int    CDK_LABEL_CONNECTION( GetSet, Label, LeaderConnect )
/*- - - - - */
/*
/*   Input:
/*
int    GetSet;    /* 1 = Get, 2 = SET.
int    Label;     /* The ID of the label to get/set.
/*
/*   Input / Output:
int    *LeaderConnect; /* The point number to connect the leader:
/*                      4-5-6
/*                      |
/*                      3 TEXT 7
/*                      |
/*                      2-1-0
/*
/* Returns:          0 = OK.
/*                  < 0 = See Chapter 1, General
/*                  Concepts, Status.
/*- - - - - */
```

CDK_LABEL_CREATE

Create a label.

Syntax :

```

/*- - - - - */
int CDK_LABEL_CREATE( Text, LabelPos, TextAngle, HorAlignment, VerAlignment,
                     LeaderConnect, NumLeaderPoints, LeaderTree )
/*- - - - - */

/*
/* Input :
/*
/* The text of the label.
/* The coordinates of label position ( WORK ).
/* The angle(radians) between text direction and X-axis*/
char *Text;
double LabelPos[];
double TextAngle;
int HorAlignment;
int VerAlignment;
int LeaderConnect;
/* 0 = Left 1 = Center 2 = Right.
/* 0 = Normal 1 = Top 2 = Center 3 = Bottom.
/* The number of point to connect the leader:
/* 4-5-6
/* |
/* 3 TEXT 7
/* |
/* 2-1-0
/*
CDKT_LEADER_NODE
LeaderTree[];
/* The array of structures CDKT_TREE NODE,
/* that defines the leader of the label.
/* ( LeaderTree[ NumLeaderPoints] ).
/* ( See type definition of CDKT_LEADER_NODE
/* in cdk_type.h ).
/*
/*
/* Returns: > 0 = ID of the created entity,
/* < 0 = See Chapter 1, General
/* Concept, Status.
/*
/*- - - - - */

```

CDK_LABEL_LEADER

Edit the leader of the label.

Syntax :

```

/*- - - - - */
int    CDK_LABEL_LEADER( GetSet, Label,
                        Terminator, NumLeaderPoints, LeaderTree )
/*- - - - - */
/*
/*   Input:
/*
int      GetSet      /* 1 = GET, 2 = SET.
int      Label       /* The ID of the label to get/set.
/*
/*   Input / Output:
/*
int      *Terminator; /* The leader terminator symbol:
/*          -1 - without terminator
/*          1 - ARROW
/*          2 - SLASH
/*          3 - CIRCLE
/*          4 - OPEN_ARROW
/*          5 - FULL_ARROW
/*          6 - FULL_CIRCLE
/*          7 - TRIANGLE
/*          8 - FULL_TRIANGLE
int      *NumLeaderPoints /* The number of points, defines the label
CDKT_LEADER_NODE
      LeaderTree[]; /* The array of structures LEADER_NODE,
/* that defines the leader of the label.
/* ( See type definition of CDKT_TREE_NODE
/*   in cdk_type.h ).
/*
/* Returns:      > 0 = real number of points in
/*                  leader, if NumLeaderPoints
/*                  < real number of points.
/*                  < 0 = See Chapter 1,
/*                  General Concepts, Status.
/*- - - - - */

```

CDK_LABEL_LEADER2 Get a label leader and plane.
Syntax :

```

/*- - - - - */
int CDK_LABEL_LEADER2( Label, Terminator, NumLeaderPoints, Plane, LeaderTree )
/*- - - - - */
/*
/* Input :
/*
int   Label;          /* The ID of the label to get/set.
/*
/* Output :
/*
int   *Terminator;    /* The leader terminator symbol:
/* -1 = without terminator
/*  1 = ARROW
/*  2 = SLASH
/*  3 = CIRCLE
/*  4 = OPEN_ARROW
/*  5 = FULL_ARROW
/*  6 = FULL_CIRCLE
/*  7 = TRIANGLE
/*  8 = FULL_TRIANGLE
int   *NumLeaderPoints; /* The number of points, defines the label.
/* If GetSet = GET then:
/*   Input - Defines the size of LeaderTree array.
/*   Output - The real number of leader points.
double Plane[4];      /* Coefficients of the labels plane.
CDKT_LEADER_NODE
    LeaderTree[];     /* The array of structures LEADER_NODE, that defines
/* the leader of the label.
/* If a node is defined by a geometrical point, obtain
/* the coordinates of the geometrical point, even if
/* it is out of the labels plane.
/* Returns :
/* <= 0 : See General Concepts-Status on page 1-2.
/* > 0 : The Number of leader points.
/*- - - - - */

```

CDK_LABEL_NUMLEADERPOINTS

Obtain the number of leader points.

Syntax :

```

/*- - - - - */
int CDK_LABEL_NUMLEADERPOINTS( Label )
/*- - - - - */
/*
/* Input :
/*
int   Label;          /* The ID of the label.
/* Returns :
/* <= 0 : See General Concepts-Status on page 1-2.
/* > 0 : The Number of leader points.
/*- - - - - */

```

CDK_LABEL_POSITION Edit the label frame position.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int   CDK_LABEL_POSITION( GetSet, Label, LabelPos, TextAngle,
                          Box, HorAlignment, VerAlignment )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/*   Input:
/*
/*   1 = Get, 2 = Set.
/*   The ID of the label to get/set.
/*
/*   Input / Output:
/*
/*   The coordinates of label position ( WORK ).
/*   The angle between text direction and X-axis.
/*   0 = Without box, 1 = Box, 2 = Underline.
/*   0 = Normal, 1 = Left, 2 = Center, 3 = Right.
/*   0 = Normal, 1 = Top, 2 = Center, 3 = Bottom.
/*
/*
/* Returns:      0 = OK.
/*               < 0 = See Chapter 1, General
/*               Concepts, Status.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_LABEL_SET Set label parameters.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void CDK_LABEL_SET( TextDirection, SymbolSize, Box, Terminator, Scheme )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/*   Input:
/*
/*   0 = ->    1 = <-
/*   The height of the symbols.
/*   0: Without box; 1: Box; 2: Underline.
/*   The leader terminator symbol:
/*   -1 - without terminator
/*   1 - ARROW
/*   2 - SLASH
/*   3 - CIRCLE
/*   4 - OPEN_ARROW
/*   5 - FULL_ARROW
/*   6 - FULL_CIRCLE
/*   7 - TRIANGLE
/*   8 - FULL_TRIANGLE
/*
/*   Scheme file name ( without extension ).
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note :

- If this function is not used at least once, before using "CDK_LABEL_CREATE", then default values are taken.
- Any change in the parameters of CDK_LABEL_SET, cause a change in the appropriate parameters of function CDK_NOTE_SET, and vice versa.

CDK_LABEL_TEXT

Edit label text.

Syntax :

```

/*-----*/
int CDK_LABEL_TEXT( GetSet, Label, TextDirection, FontName, Scheme, SymbolSize,
                   TextLen, Text )
/*-----*/
/*
/*  Input:
/*
/*  1 = Get, 2 = Set.
/*  The ID of the label to get/set.
/*
/*  Input/OutPut:
/*
/*  0 = ->, 1 = <-.
/*  The name of the font.
/*  Scheme file name ( without extension ).
/*  The height of the symbols.
/*
/*  Input:
/*
/*  Size of buffer "Text", to resave the labels
/*  text ( Get Option only ).
/*
/*  Input/OutPut:
/*
/*  The text of the label.
/*
/*
/* Returns:
/*      0 = OK.
/*      < 0 = See Chapter 1, General
/*            Concept, Status.
/*      > 0 = Size of Text if it is >
/*            TextLen ( Get Option only ).
/*
/*-----*/

```

CDK_LABEL_UPDATE

Update a label with a plane definition.

Syntax :

```

/*- - - - - */
int CDK_LABEL_UPDATE( Label, Plane )
/*- - - - - */
/*
/*  Input :
/*
/*  The ID of the label to update.
/*  Coefficients of a new labels plane ( use NULL if it
/*  not needed ).
/*
/*
/* Returns :
/*      See General Concepts-Status on page 1-2.
/*      -2 : entity cannot be regenerated.
/*
/*- - - - - */

```



LINEAR DIMENSION

General Description

The following routines are used to create linear dimensions. The dimensions may appear horizontally, vertically or parallel to an existing line. Carry out the following process when using these routines:

1. Set global parameter values by using the subroutine DIGLMD or DIGLPR, if they have not been previously set. For details of these routines and their use, see DIMENSION PARAMETERS in Chapter 5, Drafting Routines.

Create linear dimensions by calling DIMLIN and indicating the point at which the text is to be positioned.

DILINA

Create a dimension entity for a linear.

Syntax :

```
/*- - - - - */
int DILINA ( Idpt1, Pt1, Idpt2, Pt2, Dir, Iddir, IdptexStatus )
/*- - - - - */
/*
/* Input :
/*
int *Idpt1; /* ID of 1st geometric point.
float Pt1[3]; /* The 1st point in work coordinates.
int *Idpt2; /* ID of 2nd geometric point.
float Pt2[3]; /* The 2nd point in work coordinates.
int *Dir; /* 1 : horizontal,
/* 2 : vertical,
/* 3 : parallel to line.
int *Iddir; /* ID of the direction LINE.
int *Idptex; /* ID of text positioning point.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created dimension entity
/* ( if STATUS = OK )
/*- - - - - */
```

Note : - Keeps associativity with IDPT1, IDPT2, IDDIR.

DILNEX

Extract all the specific modals for dimension entities of linear.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DILNEX ( Id, Modnam, Type, Modval, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of dimension entity.
/* Parameter name.
/* Parameter type.
/*
/* Output :
/*
/* Parameter value.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note: - The following MODNAMs and TYPEs are available:

MODNAM	TYPE
"CENTERED"	"I"
"TORH "	"I"
"WITHIN "	"I"

DILNPA

Set all the specific modals for dimension entities of linear.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DILNPA ( Modnam, Type, Modval, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Parameter name.
/* Parameter type.
/* Pointer to parameter value.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note: - The following MODNAMs and TYPEs are available:

MODNAM	TYPE	Available Values		Default Values
"CENTERED"	"I"	1-INDICATE	2-CENTERED	1-CENTERED 19-1-9
"TORH "	"I"	1 -to LEFT	2-to RIGHT	1
"WITHIN "	"I"	0 -NO	1-within	depends on drafting standard

DILNPN

Set modal values of linear dimensions.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DILNPN ( Stt, Ut, Lt, Witn, Inout, TorhPre, Sym, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Stt;      /* Status of tolerance:
/*      1 : NO-T.
/*      2 : UNI-T.
/*      3 : BI-T.
/*      4 : SNG-T.
float  *Ut;      /* Upper tolerance.
float  *Lt;      /* Lower tolerance.
int    *Witn;    /* Witness lines:
/*      1 : No.
/*      2 : 1st.
/*      3 : 2nd.
/*      4 : Both.
int    *Inout;   /* 1 : <- -> Arrows inside witness lines.
/*      2 : -> <- Arrows outside witness lines.
int    *Torh;    /* Extension:
/*      1 : Begins at tail of text.
/*      2 : Begins at head of text.
int    *Pre;     /* 1 : No prefix.
/*      2 : Prefix of SYM.
/*      3 : M.
int    *Sym;     /* Number of symbol (see Symbol Table on page 5-2).
/*
/* Output :
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DILNPR

Set modal values of linear dimensions.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DILNPR ( Stt, Ut, Lt, Witn, Inout, TorhPre, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Status of tolerance:
/*   1 = NO-T.
/*   2 = UNI-T.
/*   3 = BI-T.
/*   4 = SNG-T.
/* Upper tolerance.
/* Lower tolerance.
/* Witness lines:
/*   1 : No.
/*   2 : 1st.
/*   3 : 2nd.
/*   4 : Both.
/*   1 : <> Arrows inside witness lines.
/*   2 : >< Arrows outside witness lines.
/* Extension:
/*   1 : Begins at tail of text.
/*   2 : Begins at head of text.
/*   1 : No prefix.
/*   2 : Prefix of PHI.
/*   3 : M.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DIMLIN

Create a dimension entity for a line.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIMLIN ( Idpt1, Idpt2, Dir, Iddir, IdptexStatus )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of 1st point.
/* ID of 2nd point.
/* Orientation of linear dimension:
/*   1 : Horizontal.
/*   2 : Vertical.
/*   3 : Parallel to a specified line.
/* ID of direction line to which dimension will be
/* parallel, if DIR = 3.
/* ID of point at which text will be positioned.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of the created linear dimension.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



NOTE

General Description

The following routines allow the user to create text and position it at various locations on the screen.

- Important:** • Two new FONT functions have been added, CDK_FONT_INIT and CDK_FONT_CLOSE. CDK_FONT_INIT must be called to initialize a FONT list, and CDK_FONT_CLOSE must be called after CDK_FONT_INIT.

The only FONT functions that are exceptions to the above are CDK_FONT_LIST and CDK_FONT_GET_ACT.

- **cset** is an unused parameter. **standard.ttf** is used to create notes from versions prior to version 8.0.

Using Note Routines

Carry out the following process when using these routines:

How To:

1. To change the active font, use function CDK_FONT_SET_ACTVN.
2. To create a note, use function CDK_NOTE_CREATE.
Default values are defined (scheme, symbol size, text direction).
3. To change the default values, use function CDK_NOTE_SET.

SCHEME

Use SCHEME to change the default characters that are produced from the keyboard. Each scheme file defines a different set of characters, (Japanese, Hebrew, etc.). The scheme files are found in the directory **font**

The default scheme is **tright**. The scheme file extension is **.cdt**.

CDK_EXPNOTE

Explode a note and give all the entities (line & arc) from which this note consists. The note must have been created using the options HOR-NORMAL and VER-NORMAL.

Syntax :

```

/*- - - - - */
void CDK_EXPNOTE ( Idnote, Idbuf, Maxnum, NumidsDelopt, Status )
/*- - - - - */
/*
/* Input :
/*
int *Idnote; /* ID of the note.
/*
/* Output :
/*
int Idbuf[Maxnum]; /* The array of the entities after the explode.
/*
/* Input :
/*
int *Maxnum; /* The maximum number of entities.
/*
/* Output :
/*
int *Numids; /* The number of entities which were created.
/*
/* Input :
/*
int *Delopt; /* 0 : Do not delete the note after exploding it.
/* 1 : Delete the note after exploding it.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_FONT_CLOSE

Close the font list.

Syntax :

```

/*-----*/
int CDK_FONT_CLOSE( )
/*-----*/

```

Note : This function must be used after using CDK_FONT_INIT.

CDK_FONT_GET_ACT

Get the active font name.

Syntax :

```

/*- - - - - */
void CDK_FONT_GET_ACT( MaxChar, FontName )
/*- - - - - */
/*
/* Input:
/*
int *MaxChar; /* The number of allocated characters.
/*
/* Output:
/*
char *FontName; /* The active font's name.
/*- - - - - */

```

CDK_FONT_GET_ACT_UNI Get active font name in UNICODE.
Syntax:

```

/*- - - - - */
void CDK_FONT_GET_ACT_UNI (MaxChar, FontName)
/*- - - - - */
/*
/* Input:
/*
int    *MaxChar;    /* The number of allocated characters.
/*
/* Output:
/*
unsigned short
    FontName[];    /* The active font's name.
/*- - - - - */

```

CDK_FONT_GETNUM Get the number of fonts.
Syntax :

```

/*- - - - - */
void CDK_FONT_GETNUM( NumFonts, Status )
/*- - - - - */
/*
/* Output:
/*
int    *NumFonts;    /* The number of fonts.
int    *Status;    /* = 0: O.k.
/* < 0: Error
/*- - - - - */

```

CDK_FONT_INIT Initialize the font list.
Syntax :

```

/*- - - - - */
int CDK_FONT_INIT( )
/*- - - - - */

```

Note : This function must be used before using any other font function.
Warning : CDK_FONT_CLOSE must be called after using this function.

CDK_FONT_LIST Get the list of fonts.
Syntax :

```

/*- - - - - */
void CDK_FONT_LIST( NumFonts, MaxChar, FontList, Status )
/*- - - - - */
/*
/* Input:
/*
/* The number of fonts.
/* The number of allocated characters for each font
/* name.
/*
/* Output:
/*
/* The list of font names.
/* Status.
/*
/*- - - - - */

```

Note : This function does not need a call to CDK_FONT_INIT before being used.

CDK_FONT_IND2NAME Get the font name by index.
Syntax :

```

/*- - - - - */
void CDK_FONT_IND2NAME( Index, MaxChar, FontName, Status )
/*- - - - - */
/*
/* Input:
/*
/* The index of the font.
/* The number of allocated characters.
/*
/* Output:
/*
/* The font's name.
/* = 0: O.k.
/* < 0: Error
/*
/*- - - - - */

```

CDK_FONT_SET_ACTI Set the active font by index.
Syntax :

```

/*- - - - - */
void CDK_FONT_SET_ACTI( Index, Status )
/*- - - - - */
/*
/* Input:
/*
/* The index of the font to set.
/*
/* Output:
/*
/* = 0: O.k.
/* < 0: Error
/*
/*- - - - - */

```

CDK_FONT_SET_ACTN Set the active font by name.
Syntax :

```

/*- - - - - */
void CDK_FONT_SET_ACTN( FontName, Status )
/*- - - - - */
/*
/* Input:
/*
char *FontName; /* The name of the font to set.
/*
/* Output:
/*
int *Status; /* = 0: O.k.
/* < 0: Error
/*- - - - - */

```

CDK_NOTE_CREATE Create a note.
Syntax :

```

/*- - - - - */
int CDK_NOTE_CREATE( Text, NotePos, TextAngle, HorAlignment, VerAlignment )
/*- - - - - */
/*
/* Input :
/*
char *Text; /* The text of the note.
double NotePos[3]; /* The coordinates of note position ( WORK ).
double TextAngle; /* The angle between text direction and X-axis.
int HorAlignment; /* 0 = Left 1 = Center 2 = Right.
int VerAlignment; /* 0 = Normal 1 = Top 2 = Center 3 = Bottom.
/*
/*
/* Returns:
/* > 0 = ID of the created entity,
/* < 0 = See Chapter 1, General Concept, Status.
/*- - - - - */

```

CDK_NOTE_DIMENSIONS Get box dimensions for a given text, with the current active font, size and scheme.
Syntax :

```

/*- - - - - */
int CDK_NOTE_DIMENSIONS( Text, text_width, text_height )
/*- - - - - */
/*
/* Input :
/*
char *Text; /* Input string.
/*
/* Output :
/*
double *text_width; /* Text width.
double *text_height; /* Text height.
/*
/*
/* Returns:
/* See Chapter 1, General Concept, Status.
/*- - - - - */

```

Note : - This function may also be used for Labels.

CDK_NOTE_POSITION

Edit the frame position of notes.

Syntax :

```

/*- - - - - */
int CDK_NOTE_POSITION( GetSet, IdNote,
                      Pos, TextAngle, HorAlignment, VerAlignment )
/*- - - - - */
/*
/* Input :
/*
/* 1 = GET, 2 = SET.
/* The ID of the note to get/set.
/*
/* Input/Output:
/*
/* The coordinates of note position ( WORK ).
/* The angle (in Radians) between the text direction
/* and X-axis.
/* 0 = Left    1 = Center  2 = Right.
/* 0 = Normal  1 = Top    2 = Center  3 = Bottom.
/*
/*
/* Returns:      See Chapter 1, General Concept,
/*              Status.
/*- - - - - */

```

CDK_NOTE_SET

Set the parameters to be used when creating notes.

Syntax :

```

/*- - - - - */
int CDK_NOTE_SET( TextDirection, SymbolSize, scheme )
/*- - - - - */
/*
/* Input:
/*
/* 0 = ->    1 = <- .
/* The height of the symbols.
/* Scheme file name ( without extension ).
/*
/*
/* Returns:      See Chapter 1, General Concept,
/*              Status.
/*- - - - - */

```

Notes :

- If this function is not used, at least once, before using CDK_NOTE_CREATE then default values are taken.
- Any change in the parameters of CDK_NOTE_SET, cause a change in the appropriate parameters of function CDK_LABEL_SET, and vice versa.

CDK_NOTE_TEXT

Get / set note text.

Syntax:

```

/*- - - - - */
int CDK_NOTE_TEXT( GetSet, IdNote, TextDirection, FontName,
                  scheme, SymbolSize, TextLen, Text )
/*- - - - - */
/*
/* Input :
/*
/*
int      GetSet;      /* 1 = GET, 2 = SET.
int      IdNote;      /* The Id of the note to get/set.
/*
/* Input/Output:
/*
int      *TextDirection; /* 0 = ->; 1 = <-.
char      *FontName;    /* Font name.
/*
/* Input :
/*
char      *scheme;      /* Scheme file name(without extension).
/*
/* Input/Output:
/*
double *SimbolSize;     /* The height of the symbols.
/*
/* Input :
/*
int      TextLen;      /* Size of buffer "Text",to resave the text
/* of the note (GetSet = GET only )
/*
/* Input/Output:
/*
char      *Text;        /* The text of the note.
/* Return:
/* Set -> status.
/* Get -> < 0 = Status.
/* > 0 = Real number of text characters
/*- - - - - */

```

CDK_TEXT_EDIT_PARAMS

Get/Set the text parameters of a text entity
(Note/Label).

Syntax :

```

/*- - - - - */
int CDK_TEXT_EDIT_PARAMS ( GetSet, Id, Char_Ratio, Space_Factor, Line_Space,
                           Slant, Bold, Underline, Backwards, Upsideown, Vertical )
/*- - - - - */

/*
/* Input :
/*
int    GetSet;    /* 1 = GET, 2 = SET.
int    Id;        /* The ID of the note to get/set.
/*
/* Input/Output:
/*
double *Char_Ratio; /* Ratio between the width and the height of the
/* characters.
double *Space_Factor; /* Defines the space between characters.
double *Line_Space;   /* Defines the distance between lines.
double *Slant;        /* Defines the slant angle ( radians ).
int    *Bold;         /* Bold style font Off/On.
int    *Underline;    /* Underline Off/On.
int    *Backwards;    /* Backwards Off/On.
int    *Upsideown;    /* Upsideown Off/On.
int    *Vertical;     /* Vertical Off/On.
/* Returns :
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_TEXT_EDITOR

Open the text editor window and wait for the user to edit text.

Syntax :

```

/*- - - - - */
int CDK_TEXT_EDITOR ( Mode, Text, Status )
/*- - - - - */

/*
/* Input :
/*
int    Mode;      /* = 0 : Create text, = 1 : Edit text.
/*
/* Input/Output :
/*
char    text[];   /* Text Buffer.
int    *Status;   /* = -1 : ERROR,
/*              /* = -2 : Editing was not caried out,
/*              /* > 0 : Number of characters.
/*- - - - - */

```

CDK_TEXT_PARAMS

Set text parameters.

Syntax :

```

/* - - - - - */
void CDK_TEXT_PARAMS ( Char_Ratio, Space_Factor, Line_Space, Slant, Bold,
                        Underline, Backwards, UpsideDown, Vertical )
/* - - - - - */

/*
/* Input:
/*
double    Char_Ratio; /* Ratio between the width and the height of the
/* characters.
/*
double    Space_Factor; /* Defines the space between characters.
/*
double    Line_Space; /* Defines the distance between lines.
/*
double    Slant; /* Defines the slant angle ( radians ).
/*
int       Bold; /* Bold style font Off/On.
/*
int       Underline; /* Underline Off/On.
/*
int       Backwards; /* Backwards Off/On.
/*
int       UpsideDown; /* UpsideDown Off/On.
/*
int       Vertical; /* Vertical Off/On.
/*
/* - - - - - */

```



General Description

1. Set global parameter values by using the subroutine DIGLMD or DIGLPR, if they have not been previously set. For details of these routines and their use, see DIMENSION PARAMETERS in Chapter 5, Drafting Routines.

Create ordinate dimension entities by calling **DIMORD** and indicating the point at which the text is to be positioned.

Create an ordinate dimension for a point entity.

```
/*_ _ _ _ _ */  
int DIMORD ( Idref, Idp, Idptex, Axis, RefStatus ) /*_  
/*_ _ _ _ _ */  
        /* Input : */  
        /* ID of text position point. */  
int      *Idref; /* ID of point to be dimensioned. */  
int      *Idp;   /* ID of coordinate for the text position on the */  
int      *Idptex; /* specified axis, ( i.e. X or Y coordinate, depending */  
           /* on the value of AXIS ). */  
int      *Axis;  /*       1 : X axis. */  
           /*       2 : Y axis. */  
float     *Ref;  /* Reference value to be added to the actual value of */  
           /* the dimension. ( Default = 0. ) */  
           /* Output : */  
int      *Status; /* See General Concepts-Status on page 1-2. */  
           /* Returns : */  
           /* ID of the created ordinate dimension entity. */  
/*_ _ _ _ _ *
```

Note : - This function does not work in a drafting standard having a dimension line. Refer to the DRAF PAR function, in the Cimatron Drafting Manual, to see how to set the drafting standard.

DIORDA

Create an ordina dimension.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIORDA ( Idref, Refpt, Idpnt, Pnt, Idpdl, Ptdl, Idptex, Axis, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Idref; /* ID of geometric point corresponding to the reference
/* point.
float Refpt[3]; /* The reference point in work coordinates.
int *Idpnt; /* ID of geometric point corresponding to the
/* dimensioned point.
float Pnt[3]; /* The point to be dimensioned in work coordinates.
int *Idpdl; /* ID of geometric point corresponding to the dimension
/* line indicating point.
float Ptdl[3]; /* The dimension line indicating point in work
/* coordinates.
int *Idptex; /* ID of text positioning point.
int *Axis; /* 1 : X axis,
/* 2 : Y axis.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*
/* Returns :
/* ID of newly created dimension entity
/* ( if STATUS = OK )
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
Note : - Keep associativity by IDREF, IDPNT, IDPDL.

```

DIOREX

Extract all the specific modals for dimension entities of ordina.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DIOREX ( Id, Modnam, Type, Modval, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int *Id; /* ID of dimension entity.
char *Modnam; /* Parameter name.
char *Type; /* Parameter type.
/*
/* Output :
/*
int *Modval; /* Parameter value.
int *Status; /* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
Note: - The following MODNAMs and TYPEs are available:

```

MODNAM	TYPE
"BRKANG "	"R"
"BRKPNT_X"	"R"
"BRKPNT_Y"	"R"
"CROOKED "	"I"
"DIMLINE "	"I"
"REFVALUE"	"R"
"TEXTHOR "	"I"

DIORPA

Set all the specific modals for dimension entities of ordina.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DIORPA ( Modnam, Type, Modval, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
char    *Modnam;                /* Parameter name.
char    *Type;                 /* Parameter type.
int     *Modval;               /* Parameter value.
                                /*
                                /* Output :
                                /*
int     *Status;               /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
Note:    - The following MODNAMs and TYPEs are available:

```

MODNAM	TYPE	Available Values	Default
BREAK POINT			
"BRKANG "	"R"		45
"BRKPNT_X"	"R"		0
"BRKPNT_Y"	"R"		0
"CROOKED "	"I"	0 .. 1 0 - no 1 - crooked	0
"DIMLINE"	"I"	0 .. 1 0 - no 1 - exists	depends on drafting standard
"REFVALUE"	"R"		0
"TEXTHOR "	"I"	0 .. 1 0 - no 1 - horizontal	0

DIORPR

Set modal values for ordinate dimensions. Refer to the Cimatron^{it} **Drafting Manual**, for explanations of the modal parameters.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DIORPR ( Stt, Ut, Lt, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Stt;    /* Status of tolerance:
/*      1 : NO-T.
/*      2 : UNI-T.
/*      3 : BI-T.
/*      4 : SNG-T.
float  *Ut;    /* Upper tolerance.
float  *Lt;    /* Lower tolerance.
/*
/* Output :
/*
int    *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

RADIAL DIMENSION

General Description

Carry out the following process when using this group of routines:

1. Set global parameter values by using the subroutine DIGLMD or DIGLPR, if they have not been previously set. For details of these routines and their use, see DIMENSION PARAMETERS in Chapter 5, Drafting Routines.

Create radial dimension entities by calling DIMRAD and indicating the point at which the text is to be positioned.

DIMRAD

Create dimension entities for arcs or circles.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int DIMRAD ( Idarc, Angle, Raddia, IdptexStatus )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                     /*
                                     /* Input :
                                     /*
int      *Idarc;                    /* ID of arc/circle.
float    *Angle;                   /* Angle of dimension line.
int      *Raddia;                  /* 1 = radius.2 = diameter.
int      *Idptex;                 /* ID of point for text positioning.
                                     /*
                                     /* Output :
                                     /*
int      *Status;                 /* See General Concepts-Status on page 1-2.
                                     /*
                                     /* Returns :
                                     /* ID of the created radial dimension entity.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

DIRDEX

Extract all the specific modals for dimension entities of linear.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DIRDEX ( Id, Modnam, Type, Modval, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of dimension entity.
/* Parameter name.
/* Parameter type.
/*
/* Output :
/*
/* Parameter value.
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note: - The following MODNAMs and TYPEs are available:

MODNAM	TYPE
"CENTERED"	"I"
"RADIAL "	"I"
"TOCENTER"	"I"
"WITHIN "	"I"

DIRDPA

Set all the specific modals for dimension entities of circular.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DIRDPA ( Modnam, Type, Modval, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* Parameter name.
/* Parameter type.
/* Pointer to parameter value.
/*
/* Output :
/*
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note: - The following MODNAMs and TYPEs are available:

MODNAM	TYPE	Available Values		Default Values
"CENTERED"	"I"	1-INDICATE	2-CENTERED	1 CENTERED 19-1 -9
"RADIAL "	"I"	0-DIAMETER	1-RADIAL	0
"TOCENTER"	"I"	0-to text	1-to center	1
"WITHIN "	"I"	0-NO	1-within	Depends on drafting standard.

DIRDPR

Set modal values for radial dimensions. Refer to the Cimatron^{it} **Drafting Manual**, for explanations of the modal parameters.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void DIRDPR ( Stt, Ut, Lt, Inout, Pre, Centex, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Stt;      /* Status of tolerance:
/*      1 : NO-T.
/*      2 : UNI-T.
/*      3 : BI-T.
/*      4 : SNG-T.
float  *Ut;      /* Upper tolerance.
float  *Lt;      /* Lower tolerance.
int    *Inout;   /*      1 : <> Arrows inside witness lines.
/*      2 : >< Arrows outside witness lines.
int    *Pre;     /*      1 : No prefix/suffix.
/*      2 : Prefix of diameter.
/*      3 : Prefix of radius.
int    *Centex;  /*      1 : The arrow marking the radius will be from
/*                  the center to the arc/circle.
/*      2 : If a radius is being dimensioned in ISO mode,
/*                  the arrow will be from the text to the
/*                  arc/circle.
/*
/* Output :
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



STYLE

General Description

Manipulation of styles that are applied to text entities (NOTE, LABEL, etc.)

CDK_FONT_ASCII_TO_UNICODE

Convert a string written in ASCII to Unicode.

Syntax:

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
unsigned short *CDK_FONT_ASCII_TO_UNICODE (fontAscii)
/*
/* Purpose :
/*
/* Input:
/*
char *fontAscii; /* the font name in Unicode.
/*
/* Return value
/* a pointer to a static buffer which holds the font
/* name in Unicode.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
Note: no allocation needed.
```

CDK_FONT_UNICODE_TO_ASCII

Convert a string written in Unicode to ASCII.

Syntax:

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
char *CDK_FONT_UNICODE_TO_ASCII (fontUni)
/*
/* Input:
/*
unsigned short
/*
/* the font name in Unicode.
/*
/* Return value
/* a pointer to a static buffer which holds the font
/* name in ASCII.
/*
/*-----*/
Note: no allocation needed.
```

CDK_STYLE_DELETE

Delete a style from the styles list.

Syntax:

```

/*- - - - - */
int    CDK_STYLE_DELETE (styleName, active)
/*- - - - - */
/*
/* Input:
/*
/* the Style's name we want to delete.
/* flag, 1 -> delete even if style is active
/*      (put standard as active style.)
/*      0 -> if style is active return error code.
/*      (do not delete.)
/*
/*
/* Return value:
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_STYLE_EDIT

Edit style attributes.

Syntax:

```

/*- - - - - */
int    CDK_EDIT_STYLE (cdkStyle)
/*- - - - - */
/*
/* Input:
/*
/* structure holding Style's attributes.
/*
/*
/* Return value:
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Notes:

- CDK_STYLE structure appears in file cdk_type.h
- All CDK_STYLE_ERRORS types are detailed in cdk_type.h.
- There is an option to change only a few of the fields, and in the rest of the fields to leave the previously defined values. In order to do this, you need to fill in those fields with -100, or in field fontName: 'default'.

CDK_STYLE_GET_ACTIVE

Get the name of the active style.

Syntax:

```

/*- - - - - */
int    CDK_STYLE_GET_ACTIVE (activeStyle)
/*- - - - - */
/*
/* Output:
/*
/* the active style's name.
/*
/* Return value:
/* See General Concepts-Status on page 1-2.
/*- - - - - */

```

Note: Active Style's array should be 20 cells!

CDK_STYLE_GET_BY_NAME Get style attributes by the StyleName.
Syntax:

```

/*- - - - - */
T_INT CDK_STYLE_GET_BY_NAME (style)
/*- - - - - */
/*
/* Input/Output:
/*
/* Input:
/* style->styleName
/*
/* Output:
/* all other fields of 'CDK_STYLE'.
/*
/* Return value:
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```

CDK_STYLE_GET_DEFAULT Return the default style values.
Syntax:

```

/*- - - - - */
T_INT CDK_STYLE_GET_DEFAULT (style)
/*- - - - - */
/*
/* Output:
/*
/* All style's fields beside style->styleName which is
/* left untouched .
/*
/* Return value:
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */

```

CDK_STYLE_LIST Get styles list.
Syntax:

```

/*- - - - - */
int CDK_STYLE_LIST (list)
/*- - - - - */
/*
/* Output:
/*
/* list of all styles (seperated with '@').
/* Return value:
/* See General Concepts-Status on page 1-2.
/*
/*-----*/
Note: allocate sufficient buffer for list!!

```

CDK_STYLE_MODIFY_ENTITIES

Modify the style of an existing entity.

Syntax:

```
/*- - - - - */
int    CDK_STYLE_MODIFY_ENTITIES (entityID, styleName)
/*- - - - - */
/*
/* Input:
/*
int    entityID; /* ID of entity whose style is to be changed.
char    styleName[]; /* the new style's name that we want to set.
/*
/* Return value
/* See General Concepts-Status on page 1-2.
/*- - - - - */
```

CDK_STYLE_MODIFY_ENTITIES_IA

Modify styles of entities interactively.

Syntax:

```
/*- - - - - */
int    CDK_STYLE_MODIFY_ENTITIES_IA ()
/*- - - - - */
/*
/* Return value:
/* See General Concepts-Status on page 1-2.
/*- - - - - */
```

CDK_STYLE_NEW

Create a new style.

Syntax:

```
/*- - - - - */
int    CDK_STYLE_NEW (cdkStyle, active)
/*- - - - - */
/*
/* Input:
/*
CDK_STYLE /* structure holding Style's attributes.
*cdkStyle;
int    active; /* flag, 1: set the new style as the active style
/* 0: do not handle with the active issue.
/* (the previous active style remains active.)
/*
/* Return value:
/* See General Concepts-Status on page 1-2.
/*
/*- - - - - */
```

Notes:

- CDK_STYLE structure appears in file cdk_type.h
- All CDK_STYLE_ERRORS types are detailed in 'cdk_type.h'.
- In order to have in a field the default parameter, you need to fill -100, or in field 'fontName', fill in "default". Any other parameter will be filled as the parameter is filled.

CDK_STYLE_SET_ACTIVE

Set the active style.

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int   CDK_STYLE_SET_ACTIVE (styleName)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
          /*
          /* Purpose : set the active style.
          /*
          /* Input:
          /*
char   *styleName; /* the Style's name.
          /*
          /* Return value:
          /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_STYLE_TEXT_INIT

Set style attributes for TEXT and NOTE functions.

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int   CDK_STYLE_TEXT_INIT (cdkStyle)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
          /*
          /* Input:
          /*
CDK_STYLE *cdkStyle; /* a ptr from structe CDK_STYLE. there are 2 options:
          /* NULL -> use active style's attributes.
          /* a new style's attributes, that is not neccessary
          /* an existing style.
          /*
          /* Return value:
          /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
Note: CDK_STYLE_TEXT_INIT should be called before using CDK_NOTE_CREATE.

```

General Description

The following routines may be used to define a view, using various options, to display pre-defined views, and to close an open view. Refer to the Cimatron **Drafting Manual**, for detailed descriptions of the VIEWS definition options.

- Note:**
- The length of the name of the view must be less than or equal to 6 characters.

CDK_DRAW_NEW

Define a new drawing.

Syntax :

```
/*- - - - - */
void CDK_DRAW_NEW( DrawName, DrawNum, Status )
/*- - - - - */
/*
/* Input:
/*
char DrawName[7]; /* The drawing name.
/*
/* Output:
/*
int *DrawNum; /* The new drawing number.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

CDK_SUBS_DAT

Get a subsystem's name, type and ID by its number.

Syntax :

```
/*- - - - - */
void CDK_SUBS_DAT( SubsNum, SubType, SubsName, SubsId, Status )
/*- - - - - */
/*
/* Input:
/*
int *SubsNum; /* The subsystem's number in the table.
/*
/* Output:
/*
int *SubType; /* The subsystem type:
/* 1 = VIEW
/* 2 = DRAWING
/* 3 = MACSYS
/* 4 = FEMSYS
char SubsName[7]; /* The subsystem's name.
int *SubsId; /* The subsystem's ID.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

CDK_SUBS_TOT

Get numbers of subsystems for the given type.

Syntax :

```

/*- - - - - */
void CDK_SUBS_TOT( Type, BufSize, BufSubsNum, Number, Status )
/*- - - - - */
/*
/* Input:
/*
int *Type; /* The type:
/* 1 = VIEW
/* 2 = DRAWING
/* 3 = MACSYS
/* 4 = FEMSYS
/*
int *BufSize; /* The size of the BufSubsNum[].
/*
/* Output:
/*
int *BufSubsNum; /* The buffer of subsystems' numbers.
int *Number; /* The number of subsystems that were found.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_VIEW_ACT

Activate a pre-defined view/drawing.

Syntax :

```

/*- - - - - */
void CDK_VIEW_ACT( ViewNum, Status )
/*- - - - - */
/*
/* Input:
/*
int *ViewNum; /* The number of the view.
/*
/* Output:
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_VIEW_DEL

Delete a view.

Syntax :

```

/*- - - - - */
void CDK_VIEW_DEL( ViewNum, Status )
/*- - - - - */
/*
/* Input:
/*
int *ViewNum; /* The view's number.
/*
/* Output:
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_VIEW_NEW

Define a new view.

Syntax :

```

/*- - - - - */
void CDK_VIEW_NEW( ViewName, Indep, IdOrg, IdX, IdY, BufEntId, NumId, ViewNum,
                  Status )
/*- - - - - */
/*
/* Input:
/*
char ViewName[7]; /* The view name.
int *Indep; /* 1 = Independent, 0 = Dependent.
int *IdOrg; /* The ID of the origin point.
int *IdX; /* The ID of the point on +X axis.
int *IdY; /* The ID of the point indicating the +Y axis.
int *BufEntId; /* The buffer of entities to include to macsys.
int *NumId; /* 0 = the size of BufEntId[];
/* < 0 = entire Model.
/*
/* Output:
/*
int *ViewNum; /* The new view number.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

CDK_VIEW_SVR

Create a reference subview for the current PFM view.

Syntax :

```

/*- - - - - */
void CDK_VIEW_SVR( Length, PartFileName, ViewNum, IdMast, Status )
/*- - - - - */
/*
/* Input:
/*
int *Length; /* The length of the PFN ( =0 - current PFM )
char *PartFileName; /* The part file name.
int *ViewNum; /* The view's number.
/*
/* Output:
/*
int *IdMast; /* The master ID.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

OBDRVW

Get the drawing internal views and their reference points and scales.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBDRVW ( Ndr, Views, Nviews, Tran, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Ndr;      /* Drawing number.
/*
/* Output :
/*
int    *Views;    /* Array of view numbers belonging to the drawing.
float  *Nviews;   /* Number of views.
float  *Tran;     /* Array of transformation:
/* Each view has 4 words: ( real )
/* 1 : x-coordinate of view reference point.
/* 2 : y-coordinate of view reference point.
/* 3 : z-coordinate of view reference point.
/* 4 : scale.
float  *Status;   /* 0 : O.K.
/* -1 : Drawing number is out of range.
/* -2 : Data base inconsistency.
/* -99 : More than 27 views found.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OBDVID

Given a drawing/view number, retrieve the ID of its entity.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBDVID ( Op, Num, Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char    *Op;      /* DR : Drawing.
/* VW : View.
int     *Num;     /* Number of drawing/view/level.
/*
/* Output :
/*
int     *Id;      /* ID of its entity.
int     *Status;  /* 0 : O.K.
/* 1 : Number is out of range.
/* 2 : Not found.
/* -2 : Data base inconsistency.
/* -3 : Illegal option code.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OBDVLC

Obtain the name of the drawing/view/level, given its number.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBDVLC ( Op, Num, Name, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
char *Op;          /* DR : drawing.
/* VW : view.
/* LE : level.
int *Num;          /* Number of drawing/view/level.
/*
/* Output :
/*
/* Drawing/view/level name.
char *Name;        /*
int *Status;       /* 0 : O.K.
/* 1 : number is out of range.
/* 2 : not found.-2= data base inconsistency.
/* -3 : illegal option code.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OBDVLL

List all defined drawings/views/levels.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBDVLL ( Op, List, Nlist, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
char *Op;          /* DR : drawing.
/* VW : view.
/* LE : level.
/*
/* Output :
/*
/* Array of drawing/view/level numbers
int List[500];     /* ( max.view/drawing =47; max.levels =500 ).
/* Number of elements in the list.
int *Nlist;        /*
int *Status;       /* 0 : O.K.
/* 1 : Not found.
/* -2 : Data base inconsistency.
/* -3 : Illegal option code.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OBDVLN

Obtain the number of the drawing/view/level, given its name.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBDVLN ( Op, Name, Num, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
char *Op;          /* DR : drawing.
/* VW : view.
/* LE : level.
char *Name;        /* Drawing/view/level name.
/*
/* Output :
/*
/* Number of drawing/view/level.
int *Num;          /* 0 : O.K.
int *Status;       /* 1 : Not found.
/* -2 : Data base inconsistency.
/* -3 : Illegal option code.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OBVWLS

Given the ID of an entity, list all the views in which it is seen.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void OBVWLS ( Id, List, Nlist, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ID of entity.
int *Id;
/*
/* Output :
/*
/* Array of views numbers in which the entity is seen.
int List[47];
/* Number of entries in the array LIST.
int *Nlist;
/* 0 : O.K.
int *Status;
/* 1 : Entity is visible in model only.
/* -1 : Illegal ID number.
/* -2 : Data base inconsistency.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

OBVWTR

Given a view number, return its transformation.

Syntax :

```

/*- - - - - */
void OBVWTR ( Vnum, Trmat, Base, ScaleMode23, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Vnum;    /* View number.
/*
/* Output :
/*
float   Trmat[9]; /* Transformation matrix.
float   Base[3];  /* Base point of view.
float   *Scale;    /* View scale.
int     *Mode23;   /* 2D / 3D mode.
int     *Status;   /*      0 : O.K.
/*      -1 : View number is out of range.
/*      -2 : Data base inconsistency.
/*- - - - - */

```

VWACTV

Activate and display a pre-defined view.

Syntax :

```

/*- - - - - */
void VWACTV ( Name, Status )
/*- - - - - */
/*
/* Input :
/*
char    *Name;    /* Name of view to activate and display ( up to 6
/* characters long. )
/*
/* Output :
/*
int     *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

VWDFCD

Define a view using the CURRENT DISPLAY option.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VWDFCD ( Name, Idor, Indepn, N, Ids, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
char *Name; /* Name of new view ( up to 6 characters long ).
int *Idor; /* ID of origin point.
int *Indepn; /* 1 : Independent view.
/* 0 : Dependent view.
int *N; /* Number of entities to be included in the new view,
/* ( if N < 0, all entities ).
int Ids[N]; /* Array of N ID numbers of entities to be included in
/* the new view.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - This routine works in the Drafting Application only. Before using this routine exit from the Modeling Application using the routine VWEXIT, with EXMODE=1.

VWDFMP

Define a view using the MODEL PROJ. option.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VWDFMP ( Name, Idor, Idx, Idy, Indepn, N, IdsStatus )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/*
char *Name; /* Name of new view ( up to 6 characters long ).
int *Idor; /* ID of origin point of new view.
int *Idx; /* ID of a point on the +X axis.
int *Idy; /* ID of a point indicating the +Y axis:
/* 1 : Independent view.
/* 0 : Dependent view.
/*
/*
/*
int *Indepn; /*
/*
/* Input :
/*
/*
int *N; /* Number of ID numbers of entities to be included in
/* the new view ( if N < 0, all entities ).
int Ids[N]; /* Vector of N ID numbers to be included in the new
/* view.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Note : - This routine works in the Drafting Application only. Before using this routine exit from the Modeling Application using the routine VWEXIT, with EXMODE=1.

VWEXIT

Close current view with an optional exit to the Modeling application.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void VWEXIT ( Exmode, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                     /*                                     */
                                     /* Input :                             */
                                     /*                                     */
int      *Exmode;                   /* Mode of exit:                     */
                                     /*   1 : Exit current view.          */
                                     /*   2 : Exit application.           */
                                     /*                                     */
                                     /* Output :                           */
                                     /*                                     */
int      *Status;                   /* See General Concepts-Status on page 1-2. */
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



Section III

NC User Package



Chapter 6

NC Routines

Introduction

This chapter contains user routines associated with the NC application of Cimatron. Each routine appears under an appropriate subsection that logically corresponds to the interactive functions of the system.

This package allows the user to create a new MACSYS, tool path, tools, GO procedures and to receive relevant data on the above items.

START PROGRAM	Start an NC user program.
OPEN TOOLPATH	Obtain data on the current open tool path.
GLOBAL PARAMS	Obtain current global parameters.
TOOL POSITION	Obtain data on the current tool position.
MACHINE PARAMS	Obtain current machine parameters.
CURRENT TOOL	Obtain the parameters of the current tool or create/update a tool.
TOOL MOTIONS	Create tool motions.
NEW MACHINE PARAMS	Set new machine parameters.
MARKERS	Create markers.
MESSAGE	Enter a message.
TOOL	Obtain the tool tangent point from the tool tip point or vice versa.
TOOLPATH	Create a tool path and receive information on existing tool paths.
MACSYS	Perform operations on a MACSYS.
PROCEDURE MANAGEMENT	Perform operations on procedures.
TEMPLATES	Read and write parameters from template files: PCT - Procedure template and TPT - ToolPath template.

Closed Tool Paths

CLOSED TOOLPATH	Obtain information about closed tool paths.
------------------------	---

Commands

The following commands are used:

ncumake - to compile and link NC User Package programs.

ncumake -c or **ncucomp** - to compile NC User Package programs.

ncumake -l or **nculink** - to link NC User Package programs.

For additional information on commands, see Operating Instructions in Chapter 2.

Development Authorization

The following codes are required:

user_dev and **nc_user_dev** to develop and run NC User Package programs.

user_run - to run User Package programs.

To run the NC User Package, the Cimatron^{it} NC application is required.

START PROGRAM

General Description

The following routine is used to start a USER NC program.

UNC_STR

Start a USER NC program. Must be called before all the other USER NC subroutines, and may only be called once in a program.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_STR ( Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int *Status; /* 0= OK.
/* -1 : <EXIT> from interaction.
/* -2 : <REJECT> from interaction.
/* -3 : Not allowed to start program.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

Notes:

- The USER NC program may only be invoked when a toolpath is open. It is advised to check if the toolpath type is valid for your program using **UNC_MTyp** (see the OPEN TOOLPATH section of this chapter).
- All USER NC subroutines work correctly only when the MACSYS is set to be the active work system. This setting is done by **UNC_STR**. It is advised not to change the active work plane, but if you must, remember to change it back before calling USER NC routines.

UNC_STR activates an interaction which is consistent with the general interaction in all the Cimatron^{it} NC segments. The following modal table is shown: (Refer to the **NC Manual**)

- MILL_USR:num	COMMENT	MACHINE PARAMS	SERVICE
- LATH_USR:num			
- WIRE_USR:num			
- PUNC_USR:num			

By this interaction the USER running the USER NC program may set or verify the tool, machine parameters, global parameters and comment for the created NC procedure.



OPEN TOOLPATH

General Description

The following routines may be used to obtain data of the currently open toolpath.

UNC_HOME

Get the toolpath home position.

Syntax :

```
/*- - - - - */
void UNC_HOME ( Hom, Status )
/*- - - - - */
/*
/* Output :
/*
float Hom[3]; /* Home position (x,y,z), in MM, relative to the MACSYS.*/
int *Status; /* 0 : OK.
/* -3 : Not valid.
/*- - - - - */
```

UNC_MTYP

Get the current machine type.

Syntax :

```
/*- - - - - */
void UNC_MTYP ( Type, Status )
/*- - - - - */
/*
/* Output :
/*
int *Type; /* Machine type:
/* 125 : Mill, 2.5 axis
/* 130 : Mill, 3 axis
/* 141 : Mill, 4 axis, rotation-x
/* 142 : Mill, 4 axis, rotation-y
/* 143 : Mill, 4 axis, rotation-z
/* 150 : Mill, 5 axis
/* 200 : Lathe
/* 310 : Wiredm, Agie
/* 320 : Wiredm, Makino
/* 330 : Wiredm, Charmille
/* 400 : Punch
int *Status; /* 0 : OK.
/* -3 : Not valid.
/*- - - - - */
```



GLOBAL PARAMS

General Description

The following routines may be used to obtain current global parameters. These parameters were set by interaction in **UNC_STR** (see the START PROGRAM section of this chapter).

- Note:**
- Global Parameter operations may only be performed on an open procedure. This means that the routine **UNC_PRC_OPEN** (see **Procedure Management** in this chapter) must be called *before* using any of the Global Parameter routines below.

UNC_CLER

Get the current clear value.

Syntax :

```
/*- - - - - */
void UNC_CLER ( Cler, Status )
/*- - - - - */
/*
/* Output :
/*
float *Cler; /* Clear value, in MM, relative to the MACSYS.
int *Status; /* 0 : OK.
/* -3 : Not valid.
/*- - - - - */
```

UNC_CMPNS

Send the compensation block to the machine.

Syntax :

```
/*- - - - - */
void UNC_CMPNS(cmpval, status)
/*- - - - - */
T_INT *cmpval; /* I - -1 = Left, 1 = Right 0 = Off
T_INT *status; /* O - 0 = OK <>0 = ERROR
/*- - - - - */
```

- Note:**
- This function will cause the cutter compensation command to be output in the next tool motion. If the cutter compensation is switched off this function should have no effect.

UNC_MORG

Get the current origin.

Syntax :

```
/*- - - - - */
void UNC_MORG ( Org, Status )
/*- - - - - */
/*
/* Output :
/*
float Org[3]; /* Origin ( x, y, z ), in MM, relative to the MACSYS.
int *Status; /* 0 : OK.
/* -3 : Not valid.
/*- - - - - */
```

UNC_PRC_CHORG Change origin.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_CHORG( UcsId, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
                                /*
int   *UcsId;                  /* The UCS ID number.
                                /*
                                /* Output:
                                /*
int   *Status;                /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_CLON Get/Set the clear ON/OFF value of the procedure.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_CLON( Mode, Clon, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
int   *Mode;                  /* 1 = Get, 2 = Set.
                                /*
                                /* Input/Output:
                                /*
int   *Clon;                  /* 0 = Off, 1 = On.
                                /*
                                /* Output:
                                /*
int   *Status;                /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_DIA_COMP Get/Set the Diameter Compensation ON/OFF value.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_DIA_COMP(mode,dia_comp,status)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
T_INT  *mode;                 /* 1 = Get, 2 = Set.
                                /*
                                /* Input/Output:
                                /*
T_INT  *dia_comp;             /* 1 = ON, 2 = OFF.
                                /*
                                /* Output:
                                /*
T_INT  *status;               /* 0 = OK, <>0 = Error.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

- Note:**
- This function allows the user dll to determine if the cutter diameter compensation will be used at all (switches compensation on/off).

UNC_PRC_DSTL

Get/Set the tool display flag.

Syntax :

```

/*- - - - - */
void UNC_PRC_DSTL( Mode, DispTool, Status )
/*- - - - - */
/*
/* Input:
/*
int    *Mode;    /* 1 = Get, 2 = Set.
/*
/* Input/Output:
/*
int    *DispTool; /* The tool display flag:
/* 0 = regular tool display
/* 1 = multi tool display
/* 2 = display tool off
/* 3 = tool & movement display off
/* 4 = no "fast" display
/*
/* Output:
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_GCLA

Get/Set the absolute internal clear value.

Syntax :

```

/*- - - - - */
void UNC_PRC_GCLA( Mode, AbClear, Status )
/*- - - - - */
/*
/* Input:
/*
int    *Mode;    /* 1 = Get, 2 = Set.
/*
/* Input/Output:
/*
float *AbClear;  /* The absolute internal clear value.
/*
/* Output:
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_GCLI Get/Set the incremental internal clear value.
Syntax :

```

/*- - - - - */
void UNC_PRC_GCLI( Mode, InClear, Status )
/*- - - - - */
/*
/* Input:
/*
int    *Mode;    /* 1 = Get, 2 = Set.
/*
/* Input/Output:
/*
float *InClear;  /* The incremental internal clear value.
/*
/* Output:
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_GSCL Get/Set the clear value.
Syntax :

```

/*- - - - - */
void UNC_PRC_GSCL( Mode, Clear, Status )
/*- - - - - */
/*
/* Input:
/*
int    *Mode;    /* 1 = Get, 2 = Set.
/*
/* Input/Output:
/*
float *Clear;    /* The clear value.
/*
/* Output:
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_INAB Get/Set the internal clear mode of the procedure.
Syntax :

```

/*- - - - - */
void UNC_PRC_INAB( Mode, InAb, Status )
/*- - - - - */
/*
/* Input:
/*
int    *Mode;    /* 1 = Get, 2 = Set.
/*
/* Input/Output:
/*
int    *InAb;    /* 0 = absolute, 1 = incremental.
/*
/* Output:
/*
int    *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

□

TOOL POSITION

General Description

The following routines may be used to obtain information on the current tool position.

UNC_TPOS

Get the current tool position.

Syntax :

```

/*- - - - - */
void UNC_TPOS ( Tpos, Status )
/*- - - - - */
/*
/* Output :
/*
float   Tpos[3];    /* Current tool position (x, y, z), in MM, relative to
/* the MACSYS.
int     *Status;    /*      0 : OK.
/*      -3 : Not valid.
/*- - - - - */

```

UNC_TVEC

Get the current tool axis.

Syntax :

```

/*- - - - - */
void UNC_TVEC ( Tvec, Status )
/*- - - - - */
/*
/* Output :
/*
float   Tvec[3];    /* Current tool axis ( i, j, k ).
int     *Status;    /*      0 : OK.
/*      -3 : Not valid.
/*- - - - - */

```



MACHINE PARAMS

General Description

The following routines may be used to obtain the current machine parameters. These parameters were initialized by interaction in **UNC_STR** and may be changed by subroutines in the NEW MACHINE PARAMS section of this chapter.

UNC_LATH

Get the current machine parameters for Lathe tool paths.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_LATH ( Byvc, Vc, Spn, Spnd, Fdfst, Feed, Cool, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int    *Byvc;    /* 1 : by Vc
/*           2 : by spin
float  *Vc;      /* Vc - in Meter/minute
int    *Spn;     /* Spin
int    *Spnd;    /* Spindle:
/*           1 : CW Spindle
/*           2 : CCW Spindle
/*           3 : Spindle Off
int    *Fdfst;   /* Feed type:
/*           1 : Regular Feed
/*           2 : Fast
float  *Feed;    /* in MM/minute. Valid for regular feed.
int    *Cool;    /* 1 : Off
/*           2 : On
int    *Status;  /* 0 : OK.
/*           -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_MILL

Get the current machine parameters for Mill tool paths.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_MILL ( Vc, Spn, Spnd, Fdfst, Feed, Cool, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
float  *Vc;      /* in Meter/minute
int    *Spn;     /* Spin
int    *Spnd;    /* Spindle:
/*           1 : CW Spindle
/*           2 : CCW Spindle
/*           3 : Spindle Off
int    *Fdfst;   /* Feed type:
/*           1 : Regular Feed
/*           2 : Fast
float  *Feed;    /* in MM/minute. Valid for regular feed.
int    *Cool;    /* 1 : Off
/*           2 : Flood
/*           3 : Mist
/*           4 : Through
/*           5 : Air
int    *Status;  /* 0 : OK.
/*           -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PUNC

Get the current machine parameters for Punch tool paths.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PUNC ( Punc, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Output :
                                /*
                                /*
int    *Punc;                  /*    1 : Punch Off
                                /*    2 : Punch On
int    *Status;               /*    0 : OK.
                                /*   -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_BOTT

Update the bottom menu.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_BOTT ( )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_CFD

Get/Set the corner feed value for Mill or Lathe tool paths.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_CFD( Mode, CornFeed, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
int    *Mode;                  /* 1 = Get,  2 = Set.
                                /*
                                /* Input/Output:
                                /*
int    *CornFeed;             /* The corner feed value.
                                /*
                                /* Output:
                                /*
int    *Status;               /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_COOL

Get/Set the cool value for Mill or Lathe tool paths.

Syntax :

```

/*- - - - - */
void UNC_PRC_COOL( Mode, Cool, Status )
/*- - - - - */
/*
/* Input:
/*
int  *Mode;    /* 1 = Get,  2 = Set.
/*
/* Input/Output:
/*
int  *Cool;    /* The cool value:
/* for mill tool path:
/*               1-off
/*               2-flood
/*               3-mist
/*               4-through
/*               5-air
/* for lathe tool path:
/*               1-off
/*               2-on
/*
/* Output:
/*
int  *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_DWFD

Get/Set the down feed value for Mill or Lathe tool paths.

Syntax :

```

/*- - - - - */
void UNC_PRC_DWFD( Mode, DownFeed, Status )
/*- - - - - */
/*
/* Input:
/*
int  *Mode;    /* 1 = Get,  2 = Set.
/*
/* Input/Output:
/*
int  *DownFeed; /* The down feed value.
/*
/* Output:
/*
int  *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_FST

Get/Set the feed type value for Mill or Lathe tool paths.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_FST( Mode, FdFst, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
int  *Mode;      /* 1 = Get,  2 = Set.
/*
/* Input/Output:
/*
int  *FdFst;     /* The feed type:
/* 1 = regular feed
/* 2 = fast
/* 3 = corner feed
/* 4 = plunge feed
/* 5 = side feed
/* 6 = down feed
/*
/* Output:
/*
int  *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_NIB

Get/Set the NIBBLING ON/OFF value for Punch tool paths.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_NIB( Mode, Nibb, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
int  *Mode;      /* 1 = Get,  2 = Set.
/*
/* Input/Output:
/*
int  *Nibb;      /* 1 = nibbling ON, 2 = nibbling OFF.
/*
/* Output:
/*
int  *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_PFD

Get/Set the plunge feed value for Mill or Lathe tool paths.

Syntax :

```

/*- - - - - */
void UNC_PRC_PFD( Mode, PlngFeed, Status )
/*- - - - - */
/*
/* Input:
/*
int    *Mode;    /* 1 = Get,  2 = Set.
/*
/* Input/Output:
/*
int    *PlngFeed; /* The plunge feed value.
/*
/* Output:
/*
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_PUN

Get/Set the PUNCH ON/OFF value for Punch tool paths.

Syntax :

```

/*- - - - - */
void UNC_PRC_PUN( Mode, Punch, Status )
/*- - - - - */
/*
/* Input:
/*
int    *Mode;    /* 1 = Get,  2 = Set.
/*
/* Input/Output:
/*
int    *Punch;   /* 1 = punch ON, 2 = punch OFF.
/*
/* Output:
/*
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_RFD

Get/Set the feed value for Mill or Lathe tool paths.

Syntax :

```

/*- - - - - */
void UNC_PRC_RFD( Mode, Feed, Status )
/*- - - - - */
/*
/* Input:
/*
int    *Mode;    /* 1 = Get,  2 = Set.
/*
/* Input/Output:
/*
float  *Feed;     /* The feed value.
/*
/* Output:
/*
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_SDFD

Get/Set the side feed value for Mill or Lathe tool paths.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_SDFD( Mode, SideFeed, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
int   *Mode;                    /* 1 = Get, 2 = Set.
                                /*
                                /* Input/Output:
                                /*
int   *SideFeed;                /* The side feed value.
                                /*
                                /* Output:
                                /*
int   *Status;                  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_SPD

Get/Set the spindle value.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_SPD( Mode, Spindle, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
int   *Mode;                    /* 1 = Get, 2 = Set.
                                /*
                                /* Input/Output:
                                /*
int   *Spindle;                 /* The spindle value: 1 = CW, 2 = CCW, 3 = OFF.
                                /*
                                /* Output:
                                /*
int   *Status;                  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_SPN

Get/Set the spin value.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PRC_SPN( Mode, Spin, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
int   *Mode;                    /* 1 = Get, 2 = Set.
                                /*
                                /* Input/Output:
                                /*
int   *Spin;                    /* The spin value.
                                /*
                                /* Output:
                                /*
int   *Status;                  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PRC_VCS

Get/Set the Vc or spin value for Lathe tool paths.

Syntax :

```

/*- - - - - */
void UNC_PRC_VCS( Mode, VcSpin, Status )
/*- - - - - */
    /*
    /* Input:
    /*
    int  *Mode;    /* 1 = Get, 2 = Set.
    /*
    /* Input/Output:
    /*
    int  *VcSpin;  /* 1 = by Vc, 2 = by spin.
    /*
    /* Output:
    /*
    int  *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_PRC_VCV

Get/Set the Vc value.

Syntax :

```

/*- - - - - */
void UNC_PRC_VCV( Mode, VcVal, Status )
/*- - - - - */
    /*
    /* Input:
    /*
    int  *Mode;    /* 1 = Get, 2 = Set.
    /*
    /* Input/Output:
    /*
    float *VcVal;  /* The Vc value.
    /*
    /* Output:
    /*
    int  *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - */

```



CURRENT TOOL

General Description

The following routines may be used to obtain the parameters of the current tool.

UNC_TAGI

Get the parameters of the current tool.

Note: • For an Agie tool.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - */
void UNC_TAGI ( Gnrt, Flsh, Offs, Angl, Regs, Status )
/*- - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int  *Gnrt;      /* Generator ( T )
int  *Flsh;      /* Flush ( S )
int  *Offs;      /* Offset ( D )
int  *Angl;      /* Angle ( P )
int  *Regs;      /* Register
int  *Status;    /* 0 : OK.
/*              /* -3=: Not valid.
/*- - - - - - - - - - - - - - - - - - - - */
```

UNC_TCHR

Get the parameters of the current tool.

Note: • For a Charmille tool.**Syntax :**

```

/*- - - - - */
void UNC_TCHR ( Gcd1, Gcd2, Gcd3, Gcd4, Gcd5, CregKreg, Xreg, Yreg, Rreg, Areg,
               OffsStatus )
/*- - - - - */
/*
/* Output :
/*
int    *Gcd1;    /* 1 : G27
/*              /* 2 : G28
/*              /* 3 : G29
/*              /* 4 : G30
/*              /* 5 : G28G29
/*              /* 6 : 5G29G30
/*              /* 7 : G32
int    *Gcd2;    /* 1 : G38
/*              /* 2 : G39
/*              /* 3 : None
int    *Gcd3;    /* 1 : G45
/*              /* 2 : G46
/*              /* 3 : None
int    *Gcd4;    /* 1 : G60
/*              /* 2 : G61
/*              /* 3 : None
int    *Gcd5;    /* 1 : G62
/*              /* 2 : G63
/*              /* 3 : None
float  *Creg;    /* C Register value
float  *Kreg;    /* K Register value
float  *Xreg;    /* X Register value, in MM
float  *Yreg;    /* Y Register value, in MM
float  *Rreg;    /* R Register value, in MM
float  *Areg;    /* A Register value
int    *Offs;    /* Offset
int    *Status;  /* 0 : OK.
/*              /* -3 : Not valid.
/*- - - - - */

```

UNC_TDRL

Get the parameters of the current tool.

Note: • For a Drill or Lathe-Drill tool.**Syntax :**

```

/*- - - - - */
void UNC_TDRL ( Diam, Tipa, Clrl, Cutl, Gugl, Status )
/*- - - - - */
/*
/* Output :
/*
float  *Diam;    /* Tool diameter, in MM.
float  *Tipa;    /* Tool tip angle, in degrees.
float  *Clrl;    /* Tool clear length, in MM
float  *Cutl;    /* Tool cut length, in MM.
float  *Gugl;    /* Tool gauge length, in MM
int    *Status;  /* 0 : OK.
/*              /* -3 : Not valid.
/*- - - - - */

```

UNC_TGRV

Get the parameters of the current tool.

Note: • For a Groove tool.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TGRV ( Lang, Crnr, Axan, Cutl, Cutw, HlenHwid, Ctrp, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
float *Lang;      /* Tool lead angle, in degrees.
float *Crnr;      /* Tool corner radius, in MM.
float *Axan;      /* Tool axis angle, in degrees.
float *Cutl;      /* Tool cut length, in MM.
float *Cutw;      /* Tool cut width, in MM
float *Hlen;      /* Holder length, in MM.
float *Hwid;      /* Holder width, in MM.
int *Ctrp;        /* Control point indication:
/*      -1 : Left
/*      0 : Center
/*      1 : Right
int *Status;      /*      0 : OK.
/*      -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_THLD

Get the holder number of the current tool.

Note: • For Mill, Drill, Lathe, Lathe-Drill, Thread and Groove tools.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_THLD ( Hold, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int *Hold;        /* Holder number.
int *Status;      /*      0 : OK.
/*      -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TLAT

Get current tool parameters for lathe tool.

Syntax :

```

/*- - - - - */
void UNC_TLAT( EdgRad, FacAng, BacAng, TlWid, HLen, HWid, CtrPnt, Status )
/*- - - - - */
/*
/*  Output:
/*
/*  float *EdgRad;    /* The edge radius, in degrees.
/*  float *FacAng;    /* The face angle, in degrees.
/*  float *BacAng;    /* The back angle, in degrees.
/*  float *TlWid;     /* The tool width, in MM.
/*  float *HLen;      /* The holder length, in MM.
/*  float *HWid;      /* The holder width, in MM.
/*  int *CtrPnt;      /* The control point indication: 0 = center, 1 = tip.
/*  int *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TL_ATT

Return tool attributes for given ID.

Syntax :

```

/*- - - - - */
void UNC_TL_ATT( id, type, tlname, tmtral, tltext, intext, used, status )
/*- - - - - */
/*
/*  Input:
/*
/*  int *id;          /* The Tool's ID.
/*
/*  Output:
/*
/*  int *type;        /* The Tool's type.
/*  110 = MILL
/*  120 = DRILL
/*  210 = LATHE
/*  220 = LDRIL
/*  230 = THREA
/*  240 = GROOV
/*  310 = WAGIE
/*  320 = CHARML
/*  330 = MAKINO
/*  410 = PUNCH
/*
/*  char tlname[21];  /* The Tool's name.
/*  char tmtral[21];  /* The Tool's material;
/*  char tltext[21];  /* The Tool's text.
/*  int *intext;       /* The Tool's internal/external flag.( if 0: lib.tool )
/*  int *used;         /* The usage flag.
/*  int *Status;      /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TL_CAGI

Create/Update AGIE tool.

Syntax :

```

/*- - - - - */
void UNC_TL_CAGI( TlName, TlText, Gnrt, Flsh, Offs, Ang, Regs, Id, Status )
/*- - - - - */
/*
/*  Input:
/*
/*
char  TlName[21]; /* The tool name.
char  TlText[21]; /* The tool comment.
int   *Gnrt;      /* Generator ( T ).
int   *Flsh;      /* Flush      ( S ).
int   *Offs;      /* Offset      ( D ).
int   *Ang;       /* Angle       ( P ).
int   *Regs;      /* Register.
/*
/*  Output:
/*
/*
int   *Id;        /* The tool ID.
int   *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TL_CDRL

Create/Update drill tool.

Syntax :

```

/*- - - - - */
void UNC_TL_CDRL( TlName, TlText, Diam, TipAng, ClrLen, CutLen, GugLen, Id, Status )
/*- - - - - */
/*
/*  Input:
/*
/*
char  TlName[21]; /* The tool name.
char  TlText[21]; /* The tool comment.
float *Diam;      /* The diameter.
float *TapAng;    /* The taper angle.
float *ClrLen;    /* The tool clear length, in MM.
float *CutLen;    /* The tool cut length, in MM.
float *GugLen;    /* The tool gauge length, in MM.
/*
/*  Output:
/*
/*
int   *Id;        /* The tool ID.
int   *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TL_CGRV

Create/Update groove tool.

Syntax :

```

/*- - - - - */
void UNC_TL_CGRV( TlName, TlText, LdAng, CrnRad, AxAng, CutLen, CutWid,
                  HLen, HWid, CtrPnt, Id, Status )
/*- - - - - */
/*
/*   Input:
/*
char  TlName[21]; /* The tool name.
char  TlText[21]; /* The tool comment.
float *LdAng;      /* The tool lead angle, in degrees.
float *CrnRad;     /* The corner radius.
float *AxAng;      /* The tool axis angle, in degrees.
float *CutLen;     /* The tool cut length, in MM.
float *CutWid;     /* The tool cut width, in MM.
float *HLen;       /* The holder length, in MM.
float *HWid;       /* The holder width, in MM.
int   *CtrPnt;     /* The control point indication: 0 = center, 1 = tip.
/*
/*   Output:
/*
int   *Id;         /* The tool ID.
int   *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TL_CLAT

Create/Update lathe tool.

Syntax :

```

/*- - - - - */
void UNC_TL_CLAT( TlName, TlText, EdgRad, FacAng, BacAng, TlWid, HLen, HWid, CtrPnt,
                  Id, Status )
/*- - - - - */
/*
/*   Input:
/*
char  TlName[21]; /* The tool name.
char  TlText[21]; /* The tool comment.
float *EdgRad;     /* The edge radius, in degrees.
float *FacAng;     /* The face angle, in degrees.
float *BacAng;     /* The back angle, in degrees.
float *TlWid;      /* The tool width, in MM.
float *HLen;       /* The holder length, in MM.
float *HWid;       /* The holder width, in MM.
int   *CtrPnt;     /* The control point indication: 0 = center, 1 = tip.
/*
/*   Input/Output :
/*
int   *Id;         /* Tool ID
/*   -1 : create tool
/*   >0 : update tool
/*
/*   Output:
/*
int   *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TL_CLDR

Create/Update lathe drill tool.

Syntax :

```

/*- - - - - */
void UNC_TL_CLDR( TlName, TlText, Diam, TipAng, ClrLen, CutLen, GugLen, Id,
Status )
/*- - - - - */
/*
/*   Input:
/*
char   TlName[21]; /* The tool name.
char   TlText[21]; /* The tool comment.
float  *Diam;      /* The diameter.
float  *TapAng;    /* The taper angle.
float  *ClrLen;    /* The tool clear length, in MM.
float  *CutLen;    /* The tool cut length, in MM.
float  *GugLen;    /* The tool gauge length, in MM.
/*
/*   Input/Output :
/*
int    *Id;        /* Tool ID
/*      -1 : create tool
/*      >0 : update tool
/*
/*   Output:
/*
int    *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TL_CMIL

Create/Update mill tool.

Syntax :

```

/*- - - - - */
void UNC_TL_CMIL ( TlName, TlText, Ball, Diam, CrnRad, TapAng, ClrLen, CutLen,
GugLen, TethNum, DimCmp, LenCmp, FixCmp, Hdiam, Id, Status )
/*- - - - - */
/*
/*   Input :
/*
char   *TlName;    /* tool name
char   *TlText;    /* tool comment
int    *Ball;      /* 1:ball tool 2:non ball tool
float  *Diam;      /* diameter
float  *CrnRad;    /* corner radius
float  *TapAng;    /* taper angle
float  *ClrLen;    /* tool clear length, in MM
float  *CutLen;    /* tool cut length, in MM
float  *GugLen;    /* tool gauge length, in MM
int    *TethNum;   /* number of teeth
int    *DimCmp;    /* diameter compensation
int    *LenCmp;    /* length compensation
int    *FixCmp;    /* fixture compensation
int    *Hdiam;     /* holder diameter
/*
/*   Input/Output :
/*
int    *Id;        /* Tool ID
/*      -1 : create tool
/*      >0 : update tool
/*
/*   Output :
/*
int    *Status;    /* 0 : OK
/*      -4 : tool name is illegal
/*      -5 : tool name already exists
/*      -6 : tool holder is in use
/*      -7 : tool not found
/*- - - - - */

```

UNC_TL_CTHR

Create/Update thread tool.

Syntax :

```

/*- - - - - */
void UNC_TL_CTHR( TlName, TlText, TlTyp, TlAng, TlBase, TlRad, AxAng,
                  CutLen, CutWid, HLen, HWid, Id, Status )
/*- - - - - */
/*
/*  Input:
/*
char  TlName[21]; /* The tool name.
char  TlText[21]; /* The tool comment.
int   *TlTyp;      /* The tool type:
/* 1 = witworth
/* 2 = metric
/* 3 = inch
/* 4 = triangle
/* 5 = trapezoid
float *TlAng;      /* The tool angle, in degrees.
float *TlBase;     /* The tool base, in MM. Only for trapezoid tool.
float *TlRad;      /* The tool radius, in MM. Only for non trapezoid tool.
float *AxAng;      /* The tool axis angle, in degrees.
float *CutLen;     /* The tool cut length, in MM.
float *CutWid;     /* The tool cut width, in MM.
float *HLen;       /* The holder length, in MM.
float *HWid;       /* The holder width, in MM.
/*
/* Input/Output :
/*
int   *Id;         /* Tool ID
/* -1 : create tool
/* >0 : update tool
/*
/* Output:
/*
int   *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TL_CUR

Set the current tool.

Syntax :

```

/*- - - - - */
void UNC_TL_CUR( Id, Status )
/*- - - - - */
/*
/* Input:
/*
int   *Id;         /* The Tool's ID.
/*
/* Output:
/*
int   *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TL_DEL

Delete the tool entity.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TL_DEL( Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
int  *Id;      /* The Tool's ID.
/*
/* Output:
/*
int  *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TL_ID

Return tool ID for given type and index.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TL_ID( Type, Index, Id, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
int  *Type;    /* The Tool's type.
/* 110 = MILL
/* 120 = DRILL
/* 210 = LATHE
/* 220 = LDRIL
/* 230 = THREA
/* 240 = GROOV
/* 310 = WAGIE
/* 320 = CHARML
/* 330 = MAKINO
/* 410 = PUNCH
/*
int  *Index;   /* The Tool's index.
/*
/* Output:
/*
int  *Id;      /* The tool ID.
int  *Status;  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TL_NUM Return number of tools of a specific type.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TL_NUM( Type, NumAll, NumUnus, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
                                /*
int   *Type;                    /* The Tool's type.
                                /* 110 = MILL
                                /* 120 = DRILL
                                /* 210 = LATHE
                                /* 220 = LDRIL
                                /* 230 = THREA
                                /* 240 = GROOV
                                /* 310 = WAGIE
                                /* 320 = CHARML
                                /* 330 = MAKINO
                                /* 410 = PUNCH
                                /*
                                /* Output:
                                /*
int   *NumAll;                  /* The number of tools.
int   *NumUnus;                 /* The number of unused tools.
int   *Status;                  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TL_SYM Set/Unset user defined symbol.
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TL_SYM( Mode, Id, SymId, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
                                /*
int   *Mode;                    /* 1 = set, 2 = unset.
int   *Id;                      /* The Tool's ID.
int   *SymId;                   /* The Symbol's ID.
                                /*
                                /* Output:
                                /*
int   *Status;                  /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TMIL

Get the parameters of the current tool.

Note: • For a Mill tool.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TMIL ( Diam, Hdiam, Crnr, Tapa, Clrl, Cutl, Gugl, Teth, Dimc, Lenc,
Fixc, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/* Tool diameter, in MM
/* Holder diameter, in MM
/* Tool corner radius, in MM
/* Tool taper angle, in degrees
/* Tool clear length, in MM
/* Tool cut length, in MM
/* Tool gauge length, in MM
/* Number of teeth
/* Diameter compensation
/* Length compensation
/* Fixture compensation
/*      0 : OK.
/*     -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TMKN

Get the parameters of the current tool.

Note: • For a Makino tool.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TMKN ( Cutc, Offs, Regs, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
/* Cut cont.( D )
/* Offset( E )
/* Register
/*      0 : OK.
/*     -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TPUN

Get the parameters of the current tool.

Note: • For a Punch tool.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TPUN ( Shap, Diam, Leng, Widt, Angl, Ovlp, SclpStatus )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int    *Shap;    /* Tool shape:1 = Rectangular2 = Oval3 = Round
float  *Diam;    /* Tool diameter, in MM.
float  *Leng;    /* Valid only for a round tool.
float  *Widt;    /* Tool length, in MM.
float  *Angl;    /* Valid only for a rectangular or oval tool.
float  *Widt;    /* Tool width, in MM.
float  *Angl;    /* Valid only for a rectangular or oval tool.
float  *Ovlp;    /* Tool angle, in degrees.
float  *Ovlp;    /* Overlap, in MM.
float  *Sclp;    /* Valid only for a rectangular or oval tool.
float  *Sclp;    /* Scallop, in MM.
int    *Status;  /* Valid only for a round or oval tool.
/*          0 : OK.
/*          -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TTHR

Get the parameters of the current tool.

Note: • For a Thread tool.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TTHR ( Ttyp, Angl, Base, Radu, Axan, CutlCutw, Hlen, Hwid, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int    *Ttyp;    /* Tool type:
/*          1 : Witworth
/*          2 : Metric
/*          3 : Inch
/*          4 : Triangle
/*          5 : Trapezoid
float  *Angl;    /* Tool angle, in degrees.
float  *Base;    /* Tool base, in MM.
float  *Radu;    /* Valid only for a trapezoid tool.
float  *Radu;    /* Tool radius, in MM.
float  *Axan;    /* Valid only for a non trapezoid tool.
float  *Axan;    /* Tool axis angle, in degrees.
float  *Cutl;    /* Tool cut length, in MM.
float  *Cutw;    /* Tool cut width, in MM.
float  *Hlen;    /* Holder length, in MM.
float  *Hwid;    /* Holder width, in MM.
int    *Status;  /*          0 : OK.
/*          -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TTYP

Get the type of the current tool.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TTYP ( Type, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Output :
                                /*
int      *Type;                /* Tool type:
                                /* 110 : Mill
                                /* 120 : Drill
                                /* 210 : Lathe
                                /* 220 : Lathe-drill
                                /* 230 : Thread
                                /* 240 : Groove
                                /* 310 : Agie
                                /* 320 : Charmille
                                /* 330 : Makino
int      *Status;             /* 410 : Punch
                                /* 0 : OK.
                                /* -3 : Not valid.o
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



TOOL MOTIONS

General Description

The following routines may be used to create tool motions. The motion is created for the current tool tip point, using the current machine parameters. Each motion is from the current tool position to the specified new position.

UNC_CIR5

Create a circular motion to end point. 5 axis circular motion. The motion is on the MACSYS XY plane with a Z value of the current tool position.

Note: • For Mill 4-axis, Mill 5-axis or Wireedm toolpaths.

Syntax :

```

/*- - - - - */
void UNC_CIR5 ( Arc_cen, End_pnt, Arc_dir, End_axisStatus )
/*- - - - - */
/*
/* Input :
/*
float   Arc_cen[2]; /* Arc center ( x, y ), in MM, relative to the MACSYS */
float   End_pnt[2]; /* End point ( x, y ), in MM, relative to the MACSYS */
int     *Arc_dir;   /*      1 : CCW arc */
/*      -1 : CW arc */
float   End_axis[3]; /* Tool axis at the end point ( i, j, k ) */
/*
/* Output :
/*
int     *Status;    /*      0 : OK. */
/*      -3 : Not valid. */
/*      -1 : Creation failed. */
/*      -1 : ARC_CEN not valid. */
/*      -12 : ARC_END not valid. */
/*      -13 : ARC_DIR out of bounds. */
/*      -14 : Zeroed END_AXIS vector. */
/*- - - - - */

```

UNC_CIRC

Create circular motion to end point. The motion is on the MACSYS XY plane with a Z value of the current tool position.

Syntax :

```

/*- - - - - */
void UNC_CIRC ( Arc_cen, End_pnt, Arc_dir, Status )
/*- - - - - */
/*
/* Input :
/*
float   Arc_cen[2]; /* Arc center ( x, y ), in MM, relative to the MACSYS
float   End_pnt[2]; /* End point ( x, y ), in MM, relative to the MACSYS
int     *Arc_dir;    /*      1 : CCW arc
/*      -1 : CW arc
/*
/* Output :
/*
int     *Status;     /*      0 : OK.
/*      -3 : Not valid.
/*      -1 : Creation failed.
/*      -11 : ARC_CEN not valid.
/*      -12 : ARC_END not valid.
/*      -13 : ARC_DIR out of bounds
/*- - - - - */

```

UNC_CIRS

Create circular motion to end point. 3 axis motion, with 3D compensation vector. The motion is on the MACSYS XY plane with a Z value of the current tool position.

Note: • For Mill 3-axis, Mill 4-axis, Mill 5-axis toolpaths.

Syntax :

```

/*- - - - - */
void UNC_CIRS ( Arc_cen, End_pnt, Arc_dir, Srf_nrmStatus )
/*- - - - - */
/*
/* Input :
/*
float   Arc_cen[2]; /* Arc center ( x, y ), in MM, relative to the MACSYS.
float   End_pnt[2]; /* End point ( x, y, z ), in MM, relative to the MACSYS.
int     *Arc_dir;    /*      1 : CCW arc
/*      -1 : CW arc
float   Srf_nrm[3]; /* Normal to surface at end point ( i, j, k ).
/*
/* Output :
/*
int     *Status;     /*      0 : OK.
/*      -3 : Not allowed to create.
/*      -1 : Creation failed.
/*      -11 : Error in arc angle.
/*      -12 : Error in arc center.
/*      -14 : Zeroed SRF_NRM vector.
/*- - - - - */

```

UNC_LIN5

Create linear motion to end point. 5 axis linear motion.

Note: • For Mill 4-axis, Mill 5-axis or Wiredm toolpaths.**Syntax :**

```

/*- - - - - */
void UNC_LIN5 ( End_pntEnd_axisStatus )
/*- - - - - */
/*
/* Input :
/*
float End_pnt[3]; /* End point ( x, y, z ), in MM, relative to the MACSYS.
float End_axis[3]; /* Tool axis at end point ( i, j, k ).
/*
/* Output :
/*
int *Status; /* 0 : OK.
/* -3 : Not valid.
/* -1 : Creation failed.
/* -12 : Zeroed END_AXIS vector.
/*- - - - - */

```

UNC_LINE

Create linear motion to end point.

Syntax :

```

/*- - - - - */
void UNC_LINE ( End_pnt, Status )
/*- - - - - */
/*
/* Input :
/*
float End_pnt[3]; /* End point ( x, y, z ), in MM, relative to the MACSYS.
/*
/* Output :
/*
int *Status; /* 0 : OK.
/* -3 : Not valid.
/* -1 : Creation failed.
/*- - - - - */

```

UNC_LINS

Create linear motion to end point. 3 axis motion, with 3D compensation vector.

Note: • For Mill 3-axis, Mill 4-axis, Mill 5-axis toolpaths.**Syntax :**

```

/*- - - - - */
void UNC_LINS ( End_pnt, Srf_nrm, Status )
/*- - - - - */
/*
/* Input :
/*
float End_pnt[3]; /* End point ( x, y, z ), in MM, relative to the MACSYS.
float Srf_nrm[3]; /* Normal to surface at end point ( i, j, k ).
/*
/* Output :
/*
int *Status; /* 0 : OK.
/* -3 : Not valid.
/* -1 : Creation failed.
/* -12 : Zeroed SRF_NRM vector. o
/*- - - - - */

```



NEW MACHINE PARAMS

General Description

The following routines may be used to set new machine parameters. These parameters were initialized by interaction in **UNC_STR**. After a new value is set, all following tool motions will be performed with it.

UNC_BYVC

Set by Vc or by SPIN.

Note: • For Lathe toolpaths.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_BYVC ( Byvc, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Byvc;
/*    1 : By Vc.
/*    2 : By spin.
/*
/* Output :
/*
int    *Status;
/*    0 : OK.
/*   -3 : Not valid.
/*   -11 : Value out of bounds.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

UNC_COOL

Set to new cool value.

Note: • For Mill or Lathe toolpaths.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_COOL ( Cool, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Cool;
/* New cool value
/* For Mill toolpath:
/*    1 : Off
/*    2 : Flood
/*    3 : Mist
/*    4 : Through
/*    5 : Air
/* For Lathe toolpath:
/*    1 : Off
/*    2 : On
/*
/* Output :
/*
int    *Status;
/*    0 : OK.
/*   -3 : Not valid.
/*   -11 : Cool value out of bounds.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

UNC_FAST

Set to fast motion.

Note: • For Mill or Lathe toolpaths.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_FAST ( Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int      *Status;
/*      0 : OK.
/*     -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_FEED

Set to new feed value.

Note: • For Mill or Lathe toolpaths.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_FEED ( Feed, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
float  *Feed;
/* New feed value, in MM.
/*   .0001 <= value <= 9999.
/*
/* Output :
/*
int      *Status;
/*      0 : OK.
/*     -3 : Not valid.
/*    -11 : Feed value out of bounds.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_PONF

Set to punch off or punch on.

Note: • For Punch toolpaths.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_PONF ( Punc, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int      *Punc;
/*      1 : Punch off
/*      2 : Punch on
/*
/* Output :
/*
int      *Status;
/*      0 : OK.
/*     -3 : Not valid.
/*    -11 : Value out of bounds.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_RFED Set to feed motion.

Note: • For Mill or Lathe toolpaths.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_RFED ( Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
int      *Status;      /*      0 : OK.
/*      -3 : Not valid.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

UNC_SPIN Set to new spin value.

Note: • For Mill or Lathe toolpaths.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_SPIN ( Spin, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int      *Spin;      /* New spin value
/*      1 <= value <= 25000
/*
/* Output :
/*
int      *Status;      /*      0 : OK.
/*      -3 : Not valid.
/*      -11 : Spin value out of bounds.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

UNC_SPND Set to new spindle direction.

Note: • For Mill or Lathe toolpaths.

Syntax :

```
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_SPND ( Spindle, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int      *Spindle;      /* New spindle value:1 = Spindle CW
/*      2 : Spindle CCW
/*      3 : Spindle OFF
/*
/* Output :
/*
int      *Status;      /*      0 : OK.
/*      -3 : Not valid.
/*      -11 ; Spindle value out of bounds.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
```

UNC_VCVL Set to new Vc value.

Note: • For Mill or Lathe toolpaths.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_VCVL ( Vc, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*                                     */
                                /* Input :                           */
                                /*                                     */
float  *Vc;                    /* New Vc- in units Meter/minute */
                                /* 1. <= value <= 9999.          */
                                /*                                     */
                                /* Output :                     */
                                /*                                     */
int    *Status;                /* 0 : OK.                       */
                                /* -3 : Not valid.               */
                                /* -11 : Spindle value out of bounds. */
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



General Description

The following routines may be used to create markers.

Create an open approach marker or a close approach marker.

```

/*- - - - - */
void UNC_MAPP ( Opn_cls, Apr_typ, Status )
/*- - - - - */
/*
/* Input :
/*
int      *Opn_cls;      /*      1 : Open approach marker
/*      2 : Close approach marker
int      *Apr_typ;      /* Approach type:
/*      1 : Normal
/*      2 : Tangential
/*      3 : Zup
/*      4 : Opposite tang.
/*
/* Output :
/*
int      *Status;       /*      0 : OK.
/*      -3 : Not valid.
/*      -1 : Creation failed.
/*      -11 : OPN_CLS out of bounds.
/*      -12 : APR_TYP out of bounds.
/*- - - - - */

```

Create an open layer marker or a close layer marker.

```

/*- - - - - */
void UNC_MLAY ( Opn_cls, Zval, Status )
/*- - - - - */
/*
/* Input :
/*
int      *Opn_cls;
/*      1 : Open layer marker
/*      2 : Close layer marker
float    *Zval;
/* Z = Value for layer, in MM, relative to MACSYS.
/*
/* Output :
/*
int      *Status;
/*      0 : OK.
/*     -3 : Not valid.
/*     -1 : Creation failed.
/*     -11 : OPN_CLS out of bounds.
/*- - - - - */

```

UNC_MPAS

Create an open pass marker or a close pass marker.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_MPAS ( Opn_cls, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Opn_cls;    /*    1 : Open pass marker
/*    2 : Close pass marker
/*
/* Output :
/*
int    *Status;     /*    0 : OK.
/*   -3 : Not valid.
/*   -1 : Creation failed.
/*  -11 : OPN_CLS out of bounds.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_MRTR

Create an open retract marker or a close retract marker.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_MRTR ( Opn_cls, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
int    *Opn_cls;    /*    1 : Open retract marker
/*    2 : Close retract marker
/*
/* Output :
/*
int    *Status;     /*    0 : OK.
/*   -3 : Not valid.
/*   -1 : Creation failed.
/*  -11 : OPN_CLS out of bounds.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



MESSAGE

General Description

The following routine may be used to enter a message.

UNC_MSG

Enter a message.

Syntax :

```

/*- - - - - */
void UNC_MSG ( Typ, Nchar, Msg, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Typ;    /* Message type:1 = Regular message
/*          2 : Code with seq.
/*          3 : Code without seq.
/*          4 : User def.block name
int    *Nchar;  /* Number of characters in message.
/*          0 < NCHAR < 65
char   *Msg;    /* The message.
/*
/* Output :
/*
int    *Status; /*
/*          0 : OK.
/*          -3 : Not valid.
/*          -1 : Enter failed.
/*          -11 : TYP out of bounds.
/*          -12 : NCHAR out of bounds.
/*
/*- - - - - */

```



General Description

The following routines may be used to calculate the tool tangent point from the tool tip point or to calculate the tool tip point from the tool tangent point.

UNC_ID_TDRL

Get the parameters of a tool (by ID). **Note:** For drill tool.

Syntax:

```
/*- - - - - */
void unc_id_tdr1 ( ToolId, diam, tipa, clrl, cutl, gugl, life, status)
/*- - - - - */
/*
/* Input :
/*
int *ToolId; /* Tool's ID.
/*
/* Output :
/*
float *diam; /* Tool diameter, in MM
float *tipa; /* Tool tip angle, in degrees
float *clrl; /* Tool clear length, in MM
float *cutl; /* Tool cut length, in MM
float *gugl; /* Tool gauge length, in MM
float *life; /* Life length of tool.
int *status; /* Return value :
/* 0 = OK
/* -2 = Wrong tool type
/* -3 = Not allowed to get.
/*- - - - - */
```

UNC_ID_TGRV

Get the parameters of a tool (by ID). **Note:** For groove tool.

Syntax:

```
/*- - - - - */
void unc_id_tgrv ( ToolId, lang, crnr, xan, cutl, cutw, hlen, hwid, ctrp, life, status)
/*- - - - - */
/*
/* Input :
/*
int *ToolId; /* Tool's ID.
/*
/* Output :
/*
float *lang; /* Tool lead angle, in degrees
float *crnr; /* Tool corner radius, in MM
float *axan; /* Tool axis angle, in degrees
float *cutl; /* Tool cut length, in MM
float *cutw; /* Tool cut width, in MM
float *hlen; /* Holder length, in MM
float *hwid; /* Holder width, in MM
int *ctrp; /* Control point indication:
/* -1 = left
/* 0 = center
/* 1 = right
float *life; /* Life length of tool.
int *status; /* Return value :
/* 0 = OK
/* -2 = Wrong tool type
/* -3 = Not allowed to get.
/*- - - - - */
```

UNC_ID_TLATGet the parameters of a tool (by ID). *Note:* For lathe tool.**Syntax:**

```

/*- - - - - */
void unc_id_tlat ( ToolId, edgr, faca, baka, twid, hlen, hwid, ctrp, life, status)
/*- - - - - */
/*
/* Input :
/*
int *ToolId; /* Tool's ID.
/*
/* Output :
/*
/* tool edge radius, in degrees
float *edgr; /* tool face angle, in degrees
float *faca; /* tool back angle, in degrees
float *baka; /* tool width, in MM
float *twid; /* holder length, in MM
float *hlen; /* holder width, in MM
float *hwid; /* control point indication:
int *ctrp; /* 0 = center
/* 1 = tip
float *life; /* life length of tool.
int *status; /* Return value :
/* 0 = OK
/* -2 = Wrong tool type
/* -3 = Not allowed to get.
/*- - - - - */

```

UNC_ID_TMILGet the parameters of a tool (by ID). *Note:* For mill tool.**Syntax:**

```

/*- - - - - */
void unc_id_tmil ( ToolId, diam, hdiam, crnr, cona, clrl, cutl, gugl, teth, dimc,
/*- - - - - */
lenc, fixc, nhold, hldlow, hldup, hldch, hldth, life, status)
/*- - - - - */
/*
/* Input :
/*
int *ToolId; /* Tool's ID.
/*
/* Output :
/*
/* Tool diameter, in MM
float *diam; /* Holder diameter, in MM
float *hdiam; /* Tool corner radius, in MM
float *crnr; /* Tool taper angle, in degrees
float *cona; /* Tool clear length, in MM
float *clrl; /* Tool cut length, in MM
float *cutl; /* Tool gauge length, in MM
float *gugl; /* Number of teeth
int *teth; /* Diameter compensation
int *dimc; /* Length compensation
int *lenc; /* Fixture compensation
int *fixc; /* Number of holders
int *nhold; /* Hldlow[MAXHOLD]=holder lower diameter
float hldlow[]; /* Hldup[MAXHOLD]=holder uper diameter
float hldup[]; /* Hldch[MAXHOLD]=holder total high
float hldch[]; /* Hldth[MAXHOLD]=holder conus high
float hldth[]; /* Life length of tool.
float *life; /* Return value :
int *status; /* 0 = OK
/* -2 = Wrong tool type
/* -3 = Not allowed to get.
/*- - - - - */

```

UNC_ID_TPUNGet the parameters of a tool (by ID). *Note:* For punch tool.**Syntax:**

```

/*- - - - - */
void unc_id_tpun ( ToolId, shap, diam, leng, widt, angl, ovlp, sclp, life, status)
/*- - - - - */
/*
/* Input :
/*
int    *ToolId;    /* Tool's ID.
/*
/* Output :
/*
int    *shap;      /* Tool shape:
/*      1 = rectangular
/*      2 = oval
/*      3 = round
/* Tool diameter, in MM. valid only for round tool
float  *diam;      /* Tool length, in MM. valid only for rect. or oval tool
float  *leng;      /* Tool width, in MM. valid only for rect. or oval tool
float  *width;     /* Tool angle, in degrees
float  *angl;      /* Overlap, in MM. valid only for rect. or oval tool
float  *ovlp;      /* Scallop, in MM. valid only for round or oval tool
float  *sclp;      /* Life length of tool.
float  *life;      /* Return value :
int    *status;    /*      0 = OK
/*      -2 = Wrong tool type
/*      -3 = Not allowed to get.
/*- - - - - */

```

UNC_ID_TTHRGet the parameters of a tool (by ID). *Note:* For thread tool.**Syntax:**

```

/*- - - - - */
void unc_id_tthr (ToolId, ttyp, angl, base, radu, axan, cutl, cutw, hlen, hwid, life, status)
/*- - - - - */
/*
/* Input :
/*
int    *ToolId;    /* Tool's ID.
/*
/* Output :
/*
int    *ttyp;      /* Tool type:
/*      1 = witworth
/*      2 = metric
/*      3 = inch
/*      4 = triangle
/*      5 = trapezoid
/* Tool angle, in degrees
float  *angl;      /* Tool base, in MM. valid only for trapezoid tool
float  *base;      /* Tool radius, in MM. valid only for non trapezoid tool
float  *radu;      /* Tool axis angle, in degrees
float  *axan;      /* Tool cut length, in MM
float  *cutl;      /* Tool cut width, in MM
float  *cutw;      /* Holder length, in MM
float  *hlen;      /* Holder width, in MM
float  *hwid;      /* Life length of tool.
float  *life;      /* Return value :
int    *status;    /*      0 = OK
/*      -2 = Wrong tool type
/*      -3 = Not allowed to get.
/*- - - - - */

```

UNC_TPTA

Given the tool tip point and the tool parameters, calculate the tool tangent point.

Syntax :

```

/*- - - - - */
void UNC_TPTA ( Diam, Crnr, Tapa, Ttip, Tnrm, Tpnt, Status )
/*- - - - - */
/*
/* Input :
/*
float *Diam;      /* Tool diameter, in MM.
float *Crnr;      /* Tool corner radius, in MM.
float *Tapa;      /* Tool taper angle, in degrees.
float  Ttip[3];   /* Tool tip point ( x, y, z ), in MM, relative to the
/* MACSYS.
float  Tnrm[3];   /* Normal at TTIP ( i, j, k ).
/*
/* Output :
/*
float  Tpnt[3];   /* Tool tangent point (x, y, z), in MM, relative to the
/* MACSYS.
int    *Status;   /*      0 = OK.
/*      -3 = Not valid.
/*      -11 = DIAM out of bounds.
/*      -12 = CRNR out of bounds.
/*      -13 = TAPA out of bounds.
/*      -15 = Zeroed TNRM vector.
/*- - - - - */

```

UNC_TPTP

Given the tool tangent point and the tool parameters, calculate the tool tip point.

Syntax :

```

/*- - - - - */
void UNC_TPTP ( Diam, Crnr, Tapa, Tpnt, Tnrm, Ttip, Status )
/*- - - - - */
/*
/* Input :
/*
float *Diam;      /* Tool diameter, in MM.
float *Crnr;      /* Tool corner radius, in MM.
float *Tapa;      /* Tool taper angle, in degrees.
float  Tpnt[3];   /* Tool tangent point (x, y, z), in MM, relative to the
/* MACSYS.
float  Tnrm[3];   /* Normal at TPNT ( i, j, k ).
/*
/* Output :
/*
float  Ttip[3];   /* Tool tip point ( x, y, z ), in MM, relative to the
/* MACSYS.
int    *Status;   /*      0 = OK.
/*      -3 = Not valid.
/*      -11 = DIAM out of bounds.
/*      -12 = CRNR out of bounds.
/*      -13 = TAPA out of bounds.
/*      -15 = Zeroed TNRM vector.
/*- - - - - */

```

UNC_TTAN

Given the tool tip point, calculate the tangent point for the current tool. **Note:** For a mill tool.

Syntax :

```

/*- - - - - */
void UNC_TTAN ( Ttip, Tnrm, Tpnt, Status )
/*- - - - - */
/*
/* Input :
/*
float   Ttip[3];    /* Tool tip point ( x, y, z ), in MM, relative to the
/* MACSYS.
float   Tnrm[3];    /* Normal at TTIP ( i, j, k )
/*
/* Output :
/*
float   Tpnt[3];    /* Tool tangent point (x, y, z), in MM, relative to the
/* MACSYS.
int     *Status;    /*      0 = OK.
/*      -3 = Not valid.
/*      -12 = Zeroed TNRM vector.
/*- - - - - */

```

UNC_TTIP

Given the tool tangent point, calculate the tip point for the current tool. **Note:** For a mill tool.

Syntax :

```

/*- - - - - */
void UNC_TTIP ( Tpnt, Tnrm, Ttip, Status )
/*- - - - - */
/*
/* Input :
/*
float   Tpnt[3];    /* Tool tangent point (x, y, z), in MM, relative to the
/* MACSYS.
float   Tnrm[3];    /* Normal at TPNT ( i, j, k )
/*
/* Output :
/*
float   Ttip[3];    /* Tool tip point ( x, y, z ), in MM, relative to the
/* MACSYS.
int     *Status;    /*      0 = OK.
/*      -3 = Not valid.
/*      -12 = Zeroed TNRM vector.
/*- - - - - */

```



CLOSED TOOLPATH

General Description

The following routines may be used to handle closed Tool Paths in an NC application.

OBTPNM

Obtain the name of a tool path.

Syntax :

```

/*- - - - - */
void OBTPNM ( Tpid, Tpmx, Tpnm, Tpln, Status )
/*- - - - - */
/*
/* Input :
/*
int    *Tpid;    /* Tool path entity ID.
int    *Tpmx;    /* Maximum length of the tool path name.
/*
/* Output :
/*
char    *Tpnm;    /* The tool path name.
int    *Tpln;    /* Length of the tool path name.
int    *Status;   /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UGTNTTP

Get the next Tool Path ID.

Syntax :

```

/*- - - - - */
void UGTNTTP ( Id, Vnum, Status )
/*- - - - - */
/*
/* Input/Output :
/*
int    *Id;    /* Last Tool Path ID ( initially set to 0 ).
/* Next Tool Path ID ( if found ).
/*
/* Input :
/*
int    *Vnum;   /*      0 : Search in part file.
/*      > 0 : Search in MACSYS VNUM.
/*
/* Output :
/*
int    *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```



TOOLPATH

General Description

The following routines may be used to perform operations on Machine Coordinate Systems.

GETTPTEXT

Get the toolpath text.

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void GetTPText ( id, tpText, status)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
int    *id;                    /* The tool path ID.
                                /*
                                /* Output :
                                /*
T_CHAR tpText[81];            /* The tool path text.
int    *status;                /* 0 = OK, <>0 = Error.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TP_CHK

Check if the tool path exists.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TP_CHK( Mode, TpName, TpId, TpIndex, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
int    *Mode;                  /* The mode:
                                /* 1 = Find by tool path name.
                                /* 2 = Find by tool path ID.
                                /* 3 = Find by tool path index.
                                /*
                                /* Input/Output
                                /*
char    TpName[9];             /* The tool path name.
int    *TpId;                  /* The tool path ID.
int    *TpIndex;               /* The tool path index.
                                /*
                                /* Output:
                                /*
int    *Status;                /* See General Concepts-Status on page 1-2.
                                /* 3 = The tool path doesn't exist.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TP_CLOSE

Close a tool path.

Syntax :

```

/*- - - - - */
void UNC_TP_CLOSE( Mode, SaveOpt, BlankOpt, SaveFlag, Status )
/*- - - - - */
/*
/* Input:
/*
int *Mode;      /* If the tool path cannot be saved:
/* new tool path: 1 = abort tp,      0 = do not
/* old tool path: 1 = restore org tp, 0 = do not
int *SaveOpt;    /* The save option, create the new tp entity in:
/* 1 = active level, color, pen.
/* 2 = same as old tp ( level, color, pen ).
int *BlankOpt;   /* The blank option:
/* 0 = no information to this flag.
/* 1 = blank new tool path.
/* 2 = display new tool path.
/* 3 = blank original tool path.
/* 4 = display original tool path.
int *SaveFlag;   /* The save flag:
/* 0 = do not save.
/* 1 = save and do not exit.
/*
/* Output:
/*
int *Status;     /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

UNC_TP_CREA

Create a tool path.

Syntax :

```

/*- - - - - */
void UNC_TP_CREA( TpName, TpType, UcsId, Home, Status )
/*- - - - - */
/*
/* Input:
/*
char  TpName[9]; /* The tool path name.
int   *TpType;    /* The tool path type.
/* 110 = MILL 2.5
/* 120 = MILL 3
/* 130 = MILL 4 X-AXIS
/* 140 = MILL 4 Y-AXIS
/* 150 = MILL 4 Z-AXIS
/* 160 = MILL 5
/* 210 = LATHE
/* 310 = WIREDM AGIE
/* 320 = WIREDM CHARMILLE
/* 330 = WIREDM MAKINO
/* 410 = PUNCH
int   *UcsId;     /* The UCS ID for the origin location.
float Home[3];    /* The tool start point( x, y, z ).
/*
/* Output:
/*
int   *Status;    /* See General Concepts-Status on page 1-2.
/* 3 = too many tool paths in the current MACSYS.
/*- - - - - */

```

UNC_TP_FLAGS

Read the flags of a procedure in a ToolPath.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void unc_tp_flags ( tp_id, proc_id, flags, ier )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
/* ToolPath ID.
/* Procedure ID.
/*
/* Output :
/*
/* Flags value:
/* flags[0]: not delayed(1) / is delayed(2) / interrupted(3)
/* flags[1]: is geom valid      0-not valid,  >0 valid
/* flags[2]: is manually edited 0-not edited >0 edited
/*
/* Return value :
/* 0 = OK
/* -5 = Proc ID was not found in given TP ID.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TP_IDCUR

Returns the ID of the currently open tool path.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TP_IDCUR( TpId, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
/* The tool path ID.
/*
/* Output:
/*
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TP_NUM

Returns the number of tool paths in the current MACSYS.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TP_NUM( NumTps, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
/* The number of tool paths in the current MACSYS.
/*
/* Output:
/*
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TP_PAR

Get the tool path parameter list.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TP_PAR( TpId, TpName, TpType, UcsId, Home, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
int  *TpId;      /* The tool path ID.
/*
/* Output:
/*
/*
char  TpName[9]; /* The tool path name.
int  *TpType;    /* The tool path type.
/* 110 = MILL 2.5
/* 120 = MILL 3
/* 130 = MILL 4 X-AXIS
/* 140 = MILL 4 Y-AXIS
/* 150 = MILL 4 Z-AXIS
/* 160 = MILL 5
/* 210 = LATHE
/* 310 = WIREDM AGIE
/* 320 = WIREDM CHARMILLE
/* 330 = WIREDM MAKINO
/* 410 = PUNCH
int  *UcsId;     /* The UCS ID for the origin location.
float Home[3];   /* The tool start point( x, y, z ).
int  *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TP_REN

Rename an existing tool path.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TP_REN( TpId, TpNewName, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
int  *TpId;      /* The tool path ID.
char  TpNewName[9]; /* The new name of the tool path.
/*
/* Output:
/*
int  *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

UNC_TP_REOP

Reopen a tool path.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void UNC_TP_REOP( TpIndex, TpId, Status )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
int  *TpIndex;   /* The tool path index.
/*
/* Output:
/*
int  *TpId;      /* The tool path ID.
int  *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```



General Description

The following routines may be used to perform operations on Machine Coordinate Systems.

UNC_MACS_ACT Activate the defined MACSYS.

Syntax :

```
/*- - - - - */
void UNC_MACS_ACT( MacNum, Status )
/*- - - - - */
/*
/* Input:
/*
int *MacNum; /* The MACSYS number.
/*
/* Output:
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

UNC_MACS_DEL Delete a MACSYS.

Syntax :

```
/*- - - - - */
void UNC_MACS_DEL( MacNum, Status )
/*- - - - - */
/*
/* Input:
/*
int *MacNum; /* The MACSYS number.
/*
/* Output:
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

UNC_MACS_NEW Define a new MACSYS.

Syntax :

```
/*- - - - - */
void UNC_MACS_NEW( MacsName, StdCur, UcsId, BufEntId, NumId, ViewNum, Status )
/*- - - - - */
/*
/* Input:
/*
char MacsName[7]; /* The name of the new MACSYS.
int *StdCur; /* 1 = Current, 2 = Standard.
int *UcsId; /* The UCS ID number.
int *BufEntId; /* The buffer of entities to include to MACSYS.
int *NumId; /* | 0 = The size of BufEntId[], < 0 = Entire Model.
/*
/* Output:
/*
int *ViewNum; /* The new MACSYS number.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

□

PROCEDURE MANAGEMENT

General Description

The following routines may be used to perform operations on procedures.

UNC_PRC_CLOSE Close a procedure.
Syntax :

```
/*- - - - - */
void UNC_PRC_CLOSE( )
/*- - - - - */
```

UNC_PRC_DEL Delete a procedure.
Syntax :

```
/*- - - - - */
void UNC_PRC_DEL( ProcNum, Status )
/*- - - - - */
/*
/* Input:
/*
int *ProcNum; /* The procedure number.
/*
/* Output:
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

UNC_PRC_INIT Initialize a procedure by creating an NC procedure without interaction.
Syntax :

```
/*- - - - - */
void unc_prc_int(tprc, status)
/*- - - - - */
/*
/* I/O - procedures index.
/*
T_INT *tprc;
T_INT *status;
/*
/*- - - - - */
```

Note:

- This should be used in some cases when initializing the procedure should be separated from writing the procedure header to the list. The function **UNC_PRC_OPEN** (see page 6-52) should be used to write the header to the list.
- The tprc parameter returned by UNC_PRC_INIT should be used in the call to UNC_PRC_OPEN or UNC_PRC_OPEN should be called with *tprc = -1 to open with the default value already set.
- This should not affect any user functions which already use UNC_PRC_OPEN.

UNC_PRC_INTER

Do procedure interaction after running **UNC_PRC_INIT** (see page 6-51).

Do the extended interaction for an NC procedure. Only defined masks will be used. The first mask may be:

UNC_IM_CMPNS = allow cutter diam compensation modal
or 0.
The second mask must be 0.

Syntax :

```
/*- - - - - */
void unc_prc_inter(prMask, glMask, status)
/*- - - - - */
T_INT prMask;
T_INT glMask;
T_INT *status;
/*- - - - - */
```

Note: • The mask for the cutter compensation is defined in **for_user.h**.

UNC_PRC_NUM

Obtain the number of procedures in a tool path.

Syntax :

```
/*- - - - - */
void UNC_PRC_NUM( NumProc, Status )
/*- - - - - */
/*
/* Output:
/*
int *NumProc; /* The number of procedures in the TP.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

UNC_PRC_OPEN

Open a procedure.

Syntax :

```
/*- - - - - */
void UNC_PRC_OPEN( ProcNum, Status )
/*- - - - - */
/*
/* Output:
/*
int *ProcNum; /* The number of the new procedure.
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */
```

UNC_PRC_TXT

Get/Set the text of the procedure.

Syntax :

```

/*- - - - - */
void UNC_PRC_TXT( Mode, ProcNum, ProcTxt, Status )
/*- - - - - */
/*
/* Input:
/*
/*
int  *Mode;      /* 1 = Get, 2 = Set.
int  *ProcNum;   /* The number of the procedure.
/*
/* Input/Output:
/*
char  ProcTxt[21]; /* The text of the procedure.
/*
/* Output:
/*
int  *Status;    /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

TEMPLATES

General Description

The following set of functions enable reading and writing parameters from template files: PCT - Procedure template and TPT - ToolPath template.

The relevant data structures are defined in file **UncTmplRW.h**. The parameter keys and block numbers are defined in file **NcParamDef.h**. These *.h files can be found under your Cimatron^{it} installation directory in <root_cad>\inc.

UNC_APPEND_PCT2TPT Appends a PCT file to the end of a TPT file.

Syntax:

```
/*- - - - - */
void unc_append_pct2tpt ( iPctFileName, iTptFileName, oStatus)
/*- - - - - */
/*
/* Input :
/*
char * iPctFileName; /* PCT Template file name to read from
char * iTptFileName; /* TPT Template file name to append to.
/*
/* Output :
/*
int * oStatus; /* Return status;
/*- - - - - */
```

UNC_TMPL_RW Read/write a list of parameters from/to the PCT template file. The memory of ioParVal should be allocated/freed by the caller.

Syntax:

```
/*- - - - - */
void unc_tmpl_rw ( iMode, iFileName, iParValSize, ioParVal, oStatus)
/*- - - - - */
/*
/* Input :
/*
int iMode; /* Read: UNC_PCT_READ=0, Write: UNC_PCT_WRITE=1.
char* iFileName; /* Template file name to read from.
int iParValSize; /* The size of the ioParVal array.
/*
/* Input/Output :
/*
T_UNC_TMPL_PARINFO * ioParVal; /* The parameters value;
/*
/* Output :
/*
int * oStatus; /* Return status;
/*- - - - - */
```

UNC_TMPL_TPT_RW

Reading/writing parameter value within a PCT section in a TPT file.

Syntax:

```

/*- - - - - */
void unc_tmpl_tpt_rw (iMode, iPctNdx, iFileName, iParValSize, ioParVal, oStatus)
/*- - - - - */
/*
/* Input :
/*
/* Read: UNC_PCT_READ=0, Write: UNC_PCT_WRITE=1.
/* Index of PCT section in TPT. 0 <= iPctNdx < #PCTs
/* Template file name to read from.
/* The size of the ioRetVal array.
/*
/* Input/Output :
/*
T_UNC_TMPL_PARINFO * ioParVal; /* The parameters value;
/*
/* Output :
/*
/* Return status;
/*
int * oStatus;
/*- - - - - */

```

UNC_TPT_PROC_NUM

Returns the number of PCT sections in a given TPT file.

Syntax:

```

/*- - - - - */
void unc_tpt_proc_num ( iTptFileName, oProcNum, oStatus)
/*- - - - - */
/*
/* Input :
/*
/* TPT Template file name to append to.
/*
/* Output :
/*
/* The number of procedures inside the tpt.
/* Return status;
/*
int *oProcNum;
int *oStatus;
/*- - - - - */

```



Section IV

Solid Routines



Chapter 7

Solid Routines

Introduction

This chapter contains user routines associated with the Solid application of Cimatron^{it}.

The subsections contained in this chapter are:

SKETCHER

REFERENCE GEOMETRY

SOLID OPERATIONS

SOLID DATA BASE ACCESS

UTILITIES

SOLID PROCEDURE DATA ACCESS

SOLID PICK FUNCTIONS

ASSEMBLY

SOLID ATTRIBUTES

SOLID DISPLAY

SOLID EDIT

TRANSLATE WF-> SOLID

Basic concepts

CimaDEK Solid operation function.

Each CimaDEK Solid operation function has three kinds of parameters:

- input structure with operation data;
- input structure with sketcher data;
- output status value.

All structures defined in the **cdk_sol.h** file.

User defined Sketcher function.

Sketcher data contains a pointer to the sketcher function, written by the user. This sketcher function is called during a solid function execution. An example of the structure of the sketcher function is as follows:

```

int my_sk_fun ( P_CDK_SKETCH  skData )
{
    int i, Status, Cid[5];
    Cid[0] = Cid[1] = Cid[2] = Cid[3] = Cid[4] = 0;

    /* initialize sketcher for solid operations and set contour mode */
    if ( OK != cdk_sol_sk_init ( skData ) )    return ERROR;

    /* add geometry */
    for ( i = 0; ( OK == Status ) && ( i < skData->NumIds ); i++ )
        cdk_sk_gm_addx_crv ( skData->Ids[i], Cid, &Status );
    if ( OK != Status )
        return ERROR;

    /* set contour mode */
    cdk_sk_crmode_set ( skData->contourMode );
    if ( OK != Status )
        return ERROR;

    /* solve sketcher geometry */
    cdk_sk_solve ( CDK_SK_SOLVE_AUTOMATIC_SAVE, &Status );
    if ( OK != Status )
        return ERROR;

    /* get entity and instance IDs */
    cdk_sk_get_ids ( &skData->instId, &skData->entId, &Status );
    if ( OK != Status )
        return ERROR;

    /* execute sketcher */
    if ( OK != cdk_sol_sk_done ( skData ) )    return ERROR;

    return OK;
}

```

- Note:**
- The functions CDK_SOL_SK_INIT and CDK_SOL_SK_DONE should be called from within the Solid application only. To provide Sketcher execution without Solid, use CDK_INIT_NEW and CDK_SK_END accordingly.

CimaDEK Solid entity identifier.

CimaDEK Solid entity identifier is the structure with three fields:

- unique solid ID - permanent solid data base ID;
- solid ID - ID, that is used during one PFM session;
- type of the Solid entity (see **cdk_sol.h**).

To initialize field values use the function **CDK_SOL_ENT_CTOR**. To update the solid ID use function **CDK_SOL_UPDATE**. The type should be set before.

Status

A number of functions return a status flag which is defined as:

Status: = 0 OK \neq 0 Error.

SKETCHER

CDK_INIT_NEW

Initialize a new sketcher from CDK

Syntax :

```

/*- - - - - */
void CDK_INIT_NEW ( Status )
/*- - - - - */
/* Output: */
int *Status;      /* Status 0=OK, <0 error. */
/*- - - - - */

```

CDK_SK_END

End of the sketcher from CDK.

Syntax :

```

/*- - - - - */
void CDK_SK_END ( Status )
/*- - - - - */
/* Output: */
int *Status;      /* Status 0=OK, <0 error. */
/*- - - - - */

```

CDK_SK_SOLVE

Solve sketcher from CDK.

Syntax :

```

/*- - - - - */
void CDK_SK_SOLVE ( Mode, Status )
/*- - - - - */
/* Input: */
int Mode;      /* Solving mode. */
/* = 1 : Check if fully dimension. */
/* = 2 : Automatic ( Add dummy */
/* dimensions if UNDER DIMENSIONED ) */
/* = 3 : Solve Automatic and save */
/* = 4 : Show ( Like Dimens-Show ) */
/* Output: */
int *Status;   /* = 0 : FULLY DIMENSIONED */
/* = 1 : UNDER DIMENSIONED */
/* = 2 : OVERDIMENSIONED */
/* < 0 : Internal errors */
/* -11 : Mode is out of range */
/*- - - - - */

```

CDK_SK_SAVE

Solve and save the sketcher CDK in an SKF file.

Syntax :

```

/*- - - - - */
void CDK_SK_SAVE ( File_name, Status )
/*- - - - - */
                                /* Input: */
char *File_name;                /* File name ( with extension .skf ) */
                                /* Output: */
int *Status;                    /* = 0 : Saved successfully */
                                /* < 0 : Internal errors */
/*- - - - - */

```

CDK_SK_GET_IDS

Get entity and instance IDs.

Syntax :

```

/*- - - - - */
void CDK_SK_GET_IDS ( Iid, Eid, Status )
/*- - - - - */
                                /* Output: */
int *Iid;                      /* Sketcher instance data base ID. */
int *Eid;                      /* Sketcher entity data base ID. */
int *Status;                   /* Status 0=OK <0 error */
/*- - - - - */

```

CDK_SK_GM_ADDX_CRV

Add a curve as a wire frame geometry to the sketcher.

Syntax :

```

/*- - - - - */
void CDK_SK_GM_ADDX_CRV ( Id, Ids, Status )
/*- - - - - */
                                /* Input: */
int Id;                        /* The curve ID ( line, circle or */
                                /* cubic bezier. */
                                /* Output: */
int Ids[5];                   /* IDs added: */
                                /* [0] - entity */
                                /* [1] - 1st end */
                                /* [2] - 2nd end */
                                /* [4] - center ( if circle ) */
int *Status;                   /* = 0 : OK */
                                /* < 0 : error */
                                /* = -60 : Wrong type of curve or */
                                /* DB problems. */
                                /* = -61 : Wrong number of segments */
                                /* = -62 : Wrong degree of spline */
                                /* = -63 : Trimming of curve by */
                                /* symmetry failed */
/*- - - - - */

```

CDK_SK_GM_ADDX_SYM

Add symmetry lines to the sketcher.

Syntax :

```

/*- - - - - */
void CDK_SK_GM_ADDX_SYM ( Id1, Id2, Status )
/*- - - - - */
/*
/* Input:
/* The curve ID of first symmetry
/* line ( line only ).
/* The curve ID of second symmetry
/* line ( line only ).
/* Output:
/* = 0 : OK
/* < 0 : error
/* -71 : doesn't work in point
/*      pattern
/* -72 : Wrong angle
/*- - - - - */

```

Note: - For POINT PATTERN SKETCHER mode only horizontal/vertical symmetry axis could be added. If only one symmetry line exist id2 = 0.

CDK_SK_DIM_ADDX

Add an external dimension to the sketcher.

Syntax :

```

/*- - - - - */
void CDK_SK_DIM_ADDX ( Dim_id, Ustype, Symbol_id, Status )
/*- - - - - */
/*
/* Input:
/* ID of dimension
/* Dimension type:
/* Linear Horizontal
/*          CDK_SK_DIM_HOR
/* Linear Vertical
/*          CDK_SK_DIM_VER
/* Linear Line
/*          CDK_SK_DIM_LINE
/* Two lines
/*          CDK_SK_DIM_2LINE
/* Diameter dimension
/*          CDK_SK_DIM_DIA
/* Radial dimension
/*          CDK_SK_DIM_RAD
/* Angular dimension
/*          CDK_SK_DIM_ANG2LINE
/* Dimension symbol
/*
/* Output:
/* = 0 : OK
/* = -11 Wrong dimension ID
/* = -12 Wrong type value
/* = -13 Wrong symbol number
/* = -31 The type of dimension
/* is incompatible with 'ustype'
/* = -60 There is no construction
/* points of linear hor/ver dims.
/* = -71 Calculation of dimension
/* construction failed
/* = -100 Option isn't realized yet
/*- - - - - */

```

CDK_SK_ADD_CONSTRAINT Add sketcher constraints.
Syntax :

```

/*- - - - - */
void CDK_SK_ADD_CONSTRAINT ( C_type, C_id1, C_id2, C_id3, Status )
/*- - - - - */
int C_type;          /* Input:
                    /* Constraint type:
                    /* Parallel ( CDK_SK_CONST_PARALLEL )
                    /* Normal   ( CDK_SK_CONST_NORMAL   )
                    /* Tangent  ( CDK_SK_CONST_TANGENT  )
                    /* On_curve ( CDK_SK_CONST_ON_CRV   )
int C_id1;           /* Construction ONE
int C_id2;           /* Construction TWO
int C_id3;           /* Construction THREE
                    /*
int *Status;         /* Output:
                    /* =0 OK   <0 error
                    /* = -100 Option isn't realized yet
/*- - - - - */

```

CDK_SK_EXECUTE0 Execute sketcher.
Syntax :

```

/*- - - - - */
void CDK_SK_EXECUTE0 ( Mode, Iid, Eid, Status )
/*- - - - - */
int Mode;           /* Input:
                    /* I - Execution mode:
                    /* =1 Calculate and show geometry
int Iid;            /* Sketcher instance ID
int Eid;            /* Sketcher entity ID
                    /*
int *Status;        /* Output:
                    /* Status: 0=OK   <0 error
/*- - - - - */

```

CDK_SK_SYMBOL_CREATE Create a symbol for given sketcher dimension.
Syntax :

```

/*- - - - - */
void CDK_SK_SYMBOL_CREATE ( Val, Type, Dim_id, Symbol_id, Status )
/*- - - - - */
double Val;          /* Input: */
int Type;            /* Symbol value. */
                    /* Dimension type: */
                    /* Linear Horizontal */
                    /* Linear Vertical */
                    /* Linear Line */
                    /* Two lines */
                    /* Diameter dimension */
                    /* Radial dimension */
                    /* Angular dimension */
int Dim_id;          /* ID of dimension */
                    /* Output: */
int *Symbol_id;      /* ID of created symbol */
int *Status;         /* Status: 0=OK <0 error */
                    /* = -1 Cannot allocate new symbol. */
                    /* Probably initializing problems. */
/*- - - - - */

```

CDK_SK_CRMODE_SET Set creation mode of sketcher.
Syntax :

```

/*- - - - - */
void CDK_SK_CRMODE_SET ( Crmode )
/*- - - - - */
int Crmode;          /* Input: */
                    /* Creation mode: */
                    /* Free geometries */
                    /* Open contour */
                    /* Closed contour */
                    /* Point pattern */
/*- - - - - */

```

CDK_SOL_SK_INIT

Initialize sketcher within solid operation.

Syntax :

```

/*- - - - - */
int CDK_SOL_SK_INIT ( Sketchdata )
/*- - - - - */
/* Input: */
P_CDK_SKETCH Sketchdata; /* Sketch data. */
/* Return: Status */
/*- - - - - */

```

CDK_SOL_SK_DONE

Execute sketcher within solid operation.

Syntax :

```

/*- - - - - */
int CDK_SOL_SK_DONE ( Sketchdata )
/*- - - - - */
/* Input: */
P_CDK_SKETCH Sketchdata; /* Sketch data. */
/* Return: Status */
/*- - - - - */

```

REFERENCE GEOMETRY

CDK_SOL_REFGEOM_AXIS Add reference geometry axis.**Syntax :**

```

/*- - - - - */
void CDK_SOL_REFGEOM_AXIS ( Pntorg, Pntxdir, Status )
/*- - - - - */
/* Input: */
T_CDK_PARPNT *PntOrg; /* Axis origin parametric point. */
T_CDK_PARPNT *PntXDir; /* X direction parametric point. */
/* Output: */
int *Status; /* Status: 0 = OK, <0 = Error. */
/*- - - - - */

```

CDK_SOL_REFGEOM_ENTITY Add reference entities.**Syntax :**

```

/*- - - - - */
void CDK_SOL_REFGEOM_ENTITY ( Refbuf, Refnum, Status )
/*- - - - - */
/* Input: */
P_CDK_REFENTITY RefBuf; /* Ref.entities' buffer. */
int RefNum; /* Size of the buffer. */
/* Output: */
int *Status; /* Status: 0 = OK, <0 = Error. */
/*- - - - - */

```

CDK_SOL_USEGEOM Add reference entities to the sketch.**Syntax :**

```

/*- - - - - */
void CDK_SOL_USEGEOM ( geomBuf, geomNum, Status )
/*- - - - - */
/* Input: */
P_CDK_USEGEOM geomBuf; /* ref.entities buffer. */
int geomNum; /* ref.entities number. */
/* Output: */
int *Status; /* 0 -> OK, other -> ERROR. */
/*- - - - - */

```

SOLID OPERATIONS

CDK_SOL_ARRAY

Copy object / feature as COPY >> ARRAY.

Syntax:

```

/*- - - - - */
void CDK_SOL_ARRAY ( arrayData, Status )
/*- - - - - */
P_CDK_ARRAY      /*
arrayData;      /* I - Array parameters from user.
int *Status;     /* O - Status;
/*- - - - - */

```

CDK_SOL_AXIS

Create axis.

Syntax :

```

/*- - - - - */
void CDK_SOL_AXIS ( Axisdata, Status )
/*- - - - - */
P_CDK_AXIS Axisdata;      /* Input:
                          /* Axis data.
int* Status;              /* Output:
                          /* Status.
/*- - - - - */

```

CDK_SOL_BOOLEAN

Perform Boolean operations.

Syntax:

```

/*- - - - - */
void CDK_SOL_BOOLEAN ( boolData, status)
/*- - - - - */
P_CDK_BOOLEAN      /* Input:
boolData;          /* CimaDEK Solid boolean data.
int *status;        /* Output:
/*- - - - - */

```

CDK_SOL_DRIVE

Execute DRIVE operation.

Syntax :

```

/*- - - - - */
void CDK_SOL_DRIVE ( Drivedata, Trjskdata, Secskdata, Status )
/*- - - - - */
P_CDK_DRIVE Drivedata;    /* Input:
P_CDK_SKETCH Trjskdata;   /* The drive data.
P_CDK_SKETCH Secskdata;   /* The trajectories' sketch data.
                          /* The sections' sketch data.
int *Status;              /* Output:
                          /* Status: 0 = OK, <0 = Error.
/*- - - - - */

```

CDK_SOL_EXTRUDE Execute EXTRUDE operation

Syntax :

```

/*- - - - - */
void CDK_SOL_EXTRUDE ( Extrudedata, Sketchdata, Status )
/*- - - - - */
                                /* Input: */
P_CDK_EXTRUDE Extrudedata;      /* The extrude data. */
P_CDK_SKETCH Sketchdata;       /* The sketch data. */
                                /* Output: */
int *Status;                   /* Status: 0 = OK, <0 = Error. */
/*- - - - - */

```

CDK_SOL_HOLE Execute HOLE operation.

Syntax :

```

/*- - - - - */
void CDK_SOL_HOLE ( Holedata, Pointskdata, Contskdata, Status )
/*- - - - - */
                                /* Input: */
P_CDK_HOLE Holedata;           /* The hole data. */
P_CDK_SKETCH Pointskdata;      /* The points sketch data. */
P_CDK_SKETCH Contskdata;       /* The contour sketch data. */
                                /* Output: */
int *Status;                   /* Status. */
/*- - - - - */
Note: - The contour sketcher plane in the case of CDK_HOLE_SHAPED should
       cross through last point.

```

CDK_SOL_IMPORT Execute import operation.

Syntax:

```

/*- - - - - */
void CDK_SOL_IMPORT ( importData, Status )
/*- - - - - */
P_CDK_IMPORT importData;       /* I : Import Data. */
int *Status;                   /* 0 : Status = 0 : OK, <> 0 : ERROR. */
/*- - - - - */

```

CDK_SOL_IMPORT_DEFAULT Set default parameters to import data.

Syntax:

```

/*- - - - - */
void CDK_SOL_IMPORT_DEFAULT ( importData )
/*- - - - - */
P_CDK_IMPORT importData;       /* I : Import Data. */
/*- - - - - */

```

CDK_SOL_PLANE Create a plane entity.
Syntax :

```

/*- - - - - */
void CDK_SOL_PLANE ( Planedata, Status )
/*- - - - - */
                                /* Input:                */
P_CDK_PLANE Planedata;          /* The plane data.  */
                                /* Output:          */
int*      Status;               /* Status.          */
/*- - - - - */

```

CDK_SOL_REFCRV Execute REFERENCE CURVE operation.
Syntax :

```

/*- - - - - */
void CDK_SOL_REFCRV ( refcrvData, Status )
/*- - - - - */
                                /*
                                /* Input :
                                /*
P_CDK_REFCRV                    /*
    refcrvData;                 /* extrude data.
                                /*
                                /* Output :
                                /*
int      *Status;               /* 0 -> OK, other -> ERROR.
/*- - - - - */

```

CDK_SOL_REVOLVE Execute REVOLVE operation.
Syntax :

```

/*- - - - - */
void CDK_SOL_REVOLVE ( Revolvedata, Sketchdata, Status )
/*- - - - - */
                                /* Input:
P_CDK_REVOLVE Revolvedata;      /* Revolve data.
P_CDK_SKETCH Sketchdata;        /* Sketch data.
                                /* Output:
int      *Status;               /* Status: 0 = OK, <0 = Error.
/*- - - - - */

```

CDK_SOL_RIB Execute RIB operation.
Syntax :

```

/*- - - - - */
void CDK_SOL_RIB ( Ribdata, Sketchdata, Status )
/*- - - - - */
                                /* Input:
P_CDK_RIB Ribdata;              /* Rib data.
P_CDK_SKETCH Sketchdata;        /* Sketch data.
                                /* Output:
int* Status;                    /* Status.
/*- - - - - */

```

CDK_SOL_ROTATE Rotate object.**Syntax :**

```

/*- - - - - */
void CDK_SOL_ROTATE ( Rotatedata, Status )
/*- - - - - */
P_CDK_ROTATE Rotatedata;      /* Input:          */
/* Rotate object.             */
/* Output:                    */
int*          Status;         /* Status.         */
/*- - - - - */

```

CDK_SOL_ROUND Execute Round/Chamfer operation.**Syntax :**

```

/*- - - - - */
int CDK_SOL_ROUND ( Rounddata, Status )
/*- - - - - */
P_CDK_ROUND Rounddata;      /* Input:          */
/* Round data.              */
/* Output:                  */
int*          Status;       /* Status.         */
/*- - - - - */

```

CDK_SOL_SYMMETRY Create symmetry object.**Syntax :**

```

/*- - - - - */
void CDK_SOL_SYMMETRY ( Symmdata, Status )
/*- - - - - */
P_CDK_SYMM Symmdata;      /* Input:          */
/* Data for symmetry.       */
/* Output:                  */
int*          Status;     /* Status.         */
/*- - - - - */

```

CDK_SOL_TRANSLATE Move (translate) a solid object.**Syntax :**

```

/*- - - - - */
void cdk_sol_translate ( SolUid, DelX, DelY, DelZ )
/*- - - - - */
/*
/* Input :
/*
int      SolUid;      /* Solid uid of object to move.
double   DelX;        /* Delta X
double   DelY;        /* Delta Y
double   DelZ;        /* Delta Z
/*- - - - - */

```

SOLID DATA BASE ACCESS

CDK_SOL_AXIS_DATA

Get axis data.

Syntax :

```

/*- - - - - */
int CDK_SOL_AXIS_DATA ( Axis, Axisdata )
/*- - - - - */
/* Input: */
P_CDK_SOLENTITY Axis; /* The axis. */
/* Output: */
P_CDK_AXISDATA Axisdata; /* The axis data. */
/* Return: Status */
/*- - - - - */

```

CDK_SOL_AXIS_FIRST

Get the first axis.

Syntax :

```

/*- - - - - */
int CDK_SOL_AXIS_FIRST ( Axis )
/*- - - - - */
/* Input: */
P_CDK_SOLENTITY Axis; /* The axis. */
/* Return: */
/* axis doesn't exist - FALSE, */
/* else - TRUE. */
/*- - - - - */

```

CDK_SOL_AXIS_NEXT

Get the next axis.

Syntax :

```

/*- - - - - */
int CDK_SOL_AXIS_NEXT ( Axis )
/*- - - - - */
/* Input/Output: */
P_CDK_SOLENTITY Axis; /* Curent/Next axis. */
/* Return: */
/* FALSE - end of the sequence, */
/* else - TRUE. */
/*- - - - - */

```

CDK_SOL_EDGE_CRVLIM

Get the limits of the edge curve.

Syntax :

```

/*- - - - - */
int CDK_SOL_EDGE_CRVLIM ( Edge, Crvlim )
/*- - - - - */
/* Input: */
P_CDK_SOLENTITY Edge; /* The edge. */
/* Output: */
P_DOUBLE Crvlim; /* The curve's limits. */
/* Return: Status */
/*- - - - - */

```

CDK_SOL_EDGE_DATA

Get edge data.

Syntax :

```

/*- - - - - */
int CDK_SOL_EDGE_DATA ( Edge, Edgedata )
/*- - - - - */

P_CDK_SOLENTITY Edge;      /* Input: */
/* The edge. */
P_CDK_EDGEDATA Edgedata;   /* Output: */
/* The edge's data. */
/* Return: Status */
/*- - - - - */

```

CDK_SOL_EDGE_PARI

Get the surface parameters of both faces of an edge.

Syntax :

```

/*- - - - - */
int CDK_SOL_EDGE_PARI ( Edge, Index, Crvpar, UV1, UV2, Point )
/*- - - - - */

P_CDK_SOLENTITY Edge;      /* Input: */
/* The edge. */
int Index;                 /* The curve parameter's index. */
/* Output: */
P_DOUBLE Crvpar;           /* The curve parameter's value. */
double UV1[2];             /* The UV of the 1st face. */
double UV2[2];             /* The UV of the 2nd face. */
double Point[3];           /* The point coordinates. */
/* Return: Status */
/*- - - - - */

```

CDK_SOL_EDGE_PARP

Get the surface parameters of both faces of an edge.

Syntax :

```

/*- - - - - */
int CDK_SOL_EDGE_PARP ( Edge, Crvpar, UV1, UV2, Point )
/*- - - - - */

P_CDK_SOLENTITY Edge;      /* Input: */
/* The edge. */
double Crvpar;             /* The curve parameter's value. */
/* Output: */
double UV1[2];             /* The UV of the 1st face. */
double UV2[2];             /* The of the 2nd face. */
double Point[3];           /* The point coordinates. */
/* Return: Status */
/*- - - - - */

```

CDK_SOL_EDGE_SEQ

Get the next edge.

Syntax :

```

/*- - - - - */
int CDK_SOL_EDGE_SEQ ( Face, Startedge, Edge )
/*- - - - - */

P_CDK_SOLENTITY Face;          /* Input: */
P_CDK_SOLENTITY Startedge;     /* The face. */
                                /* The start edge. */
P_CDK_SOLENTITY Edge;          /* Input/Output: */
                                /* Current/Next edge. */
                                /* Return: */
                                /* FALSE - end of the sequence, */
                                /* else - TRUE */
/*- - - - - */

```

CDK_SOL_FACE2PLF

Create a planar face from a solid face.

Syntax:

```

/*- - - - - */
int CDK_SOL_FACE2PLF (face, PlfId)
/*- - - - - */
P_CDK_SOLENTITY face;          /* I: face. */
int *PlfId;                    /* 0: The planar face ID */
                                /* Return : =0 - OK, <>0 - errors */
/*- - - - - */

```

CDK_SOL_FACE2SRF

Create a planar face from a solid face.

Syntax:

```

/*- - - - - */
int CDK_SOL_FACE2SRF (face, SrfId)
/*- - - - - */
P_CDK_SOLENTITY face;          /* I: face. */
int *SrfId;                    /* 0: The planar face ID */
                                /* Return : =0 - OK, <>0 - errors */
/*- - - - - */

```

CDK_SOL_FACE2TRS

Create trimmed surface out of solid face.

Syntax :

```

/*- - - - - */
int CDK_SOL_FACE2TRS ( Tol, Face, TrsId )
/*- - - - - */
double Tol;                    /* Input: */
P_CDK_SOLENTITY Face;          /* The tolerance ( polygon approx. ) */
                                /* The face. */
int *TrsId;                    /* Output: */
                                /* Trimmed surface ID */
                                /* Return: Status */
/*- - - - - */

```

CDK_SOL_FACE2WFM Create a wireframe entity from a solid face.
Syntax:

```

/*- - - - - */
int   CDK_SOL_FACE2WFM ( face, WfId )
/*- - - - - */
P_CDK_SOLENTITY      /*
    face;              /* I: face.
int   *WfId;          /* 0: The planar face ID
/*- - - - - */
/* Return : =0 - OK, <>0 - errors
/*- - - - - */

```

CDK_SOL_FACE_BOX_WRK Calculate the enclosing box of an object in WORK coordinates.
Syntax:

```

/*- - - - - */
int   CDK_SOL_FACE_BOX_WRK (face, Box)
/*- - - - - */
P_CDK_SOLENTITY      /*
    face;              /* I: solid object.
double *Box;          /* 0: The solid object's enclosing box.
/*- - - - - */
/* Return: =0 OK, <>0 - err
/*- - - - - */

```

CDK_SOL_FACE_DATA Get face data.
Syntax :

```

/*- - - - - */
int CDK_SOL_FACE_DATA ( Face, Facedata )
/*- - - - - */
P_CDK_SOLENTITY Face;      /* Input:
                           /* The face.
P_CDK_FACEDATA Facedata;   /* Output:
                           /* The face's data.
/*- - - - - */
/* Return: Status
/*- - - - - */

```

CDK_SOL_FACE_FIRST Get the first face entity.
Syntax :

```

/*- - - - - */
int CDK_SOL_FACE_FIRST ( ObjSol, Face )
/*- - - - - */
P_CDK_SOLENTITY ObjSol;    /* Input:
                           /* Solid object.
P_CDK_SOLENTITY Face;      /* Output:
                           /* The first face entity.
/*- - - - - */
/* Return:
/* face doesn't exist - FALSE,
/* else - TRUE.
/*- - - - - */

```

CDK_SOL_FACE_NEXT

Get the next face entity.

Syntax :

```

/*- - - - - */
int CDK_SOL_FACE_NEXT ( ObjSol, Face )
/*- - - - - */
P_CDK_SOLENTITY ObjSol;      /* Input: */
                              /* Solid object. */
P_CDK_SOLENTITY Face;        /* Input/Output: */
                              /* Current/Next face entity. */
                              /* Return: */
                              /* FALSE - end of the sequence, */
                              /* else - TRUE. */
/*- - - - - */

```

CDK_SOL_FACE_SK_PLANE

Get the Sketcher plane of a face.

Syntax:

```

/*- - - - - */
int CDK_SOL_FACE_SK_PLANE ( face, rpt, depth, mat )
/*- - - - - */
P_CDK_SOLENTITY face;        /* I: face. */
int *rpt;                    /* O: Rotation pointer. */
double *depth;               /* O: Depth. */
double *mat;                 /* O: Rotation Matrix. */
                              /* Return : =0 - OK, < 0 - error, 1 - face is not */
                              /* a planar type, rpt returns a surface ID. */
/*- - - - - */

```

CDK_SOL_LOOP_FIRST

Get the first loop.

Syntax :

```

/*- - - - - */
int CDK_SOL_LOOP_FIRST ( Face, Loop, Edge )
/*- - - - - */
P_CDK_SOLENTITY Face;        /* Input: */
                              /* The face. */
P_CDK_SOLENTITY Loop;        /* Output: */
                              /* The first loop. */
P_CDK_SOLENTITY Edge;        /* The first edge in the loop. */
                              /* Return: */
                              /* loop doesn't exist - FALSE, */
                              /* else - TRUE */
/*- - - - - */

```

CDK_SOL_LOOP_NEXT

Get the next loop.

Syntax :

```

/*- - - - - */
int CDK_SOL_LOOP_NEXT ( Face, Loop, Edge )
/*- - - - - */

P_CDK_SOLENTITY Face;      /* Input: */
                             /* The face. */
P_CDK_SOLENTITY Loop;      /* Input/Output: */
                             /* Current/Next loop. */
P_CDK_SOLENTITY Edge;      /* Output: */
                             /* The first edge in the loop. */
                             /* Return: */
                             /* FALSE - end of the sequence, */
                             /* else - TRUE */
/*- - - - - */

```

CDK_SOL_OBJ_DATA

Get solid object data.

Syntax :

```

/*- - - - - */
int CDK_SOL_OBJ_DATA ( Objcsol, Objdata )
/*- - - - - */

P_CDK_SOLENTITY Objcsol;    /* Input: */
                             /* Solid object. */
P_CDK_OBJDATA Objdata;      /* Output: */
                             /* Solid object's data. */
                             /* Return: Status */
/*- - - - - */

```

CDK_SOL_OBJ_FIRST

Get the first solid object.

Syntax :

```

/*- - - - - */
int CDK_SOL_OBJ_FIRST ( Objcsol )
/*- - - - - */

P_CDK_SOLENTITY Objcsol;    /* Output: */
                             /* Solid object. */
                             /* Return: */
                             /* Object doesn't exist - FALSE, */
                             /* else - TRUE */
/*- - - - - */

```

CDK_SOL_OBJ_NEXT

Get the next solid object.

Syntax :

```

/*- - - - - */
int CDK_SOL_OBJ_NEXT ( Objcsol )
/*- - - - - */

P_CDK_SOLENTITY Objcsol;    /* Input/Output: */
                             /* Curent/Next solid object. */
                             /* Return: */
                             /* FALSE - end of the sequence, */
                             /* else - TRUE */
/*- - - - - */

```

CDK_SOL_PLANE_DATA

Get plane data.

Syntax :

```

/*- - - - - */
int CDK_SOL_PLANE_DATA ( Plane, Planedata )
/*- - - - - */
P_CDK_SOENTITY Plane;      /* Input:          */
                             /* The plane.      */
P_CDK_PLANEDATA Planedata; /* Output:         */
                             /* The plane data. */
                             /* Return: Status  */
/*- - - - - */

```

CDK_SOL_PLANE_FIRST

Get the first plane.

Syntax :

```

/*- - - - - */
int CDK_SOL_PLANE_FIRST ( Plane )
/*- - - - - */
P_CDK_SOENTITY Plane;      /* Input:          */
                             /* The plane.      */
                             /* Return:         */
                             /* The plane doesn't exist - FALSE, */
                             /* else - TRUE.    */
/*- - - - - */

```

CDK_SOL_PLANE_NEXT

Get the next plane.

Syntax :

```

/*- - - - - */
int CDK_SOL_PLANE_NEXT ( Plane )
/*- - - - - */
P_CDK_SOENTITY Plane;      /* Input/Output:   */
                             /* Curent/Next plane. */
                             /* Return:         */
                             /* FALSE - end of the sequence, */
                             /* else - TRUE.    */
/*- - - - - */

```

CDK_SOL_REFCRV_PRIM_IDS

Obtain the geometric (wireframe) ID's of primitive curves of a reference curve.

Syntax:

```

/*- - - - - */
int CDK_SOL_REFCRV_PRIM_IDS ( refCrv, buffSize, buffIDs )
/*- - - - - */
P_CDK_SOENTITY refCrv;      /* I: Reference curve. */
int buffSize;              /* I: Buffer size.      */
int *buffIDs;              /* O: Buffer of geometrical IDs. */
                             /* R: The number of primitive curves. */
                             /* > 0 - ERROR.        */
/*- - - - - */

```

CDK_SOL_REFCRV_PRIM_NUM

Get the number of primitive curves of a reference curve.

Syntax:

```
/*- - - - - */
int   CDK_SOL_REFCRV_PRIM_NUM ( refCrv )
/*- - - - - */
P_CDK_SOENTITY    /*
    refCrv;        /* I: Reference curve.
                  /* R: The number of primitive curves.
                  /* > 0 - ERROR.
/*- - - - - */
```

CDK_SOL_REFPT_COOR

Get reference point coordinates.

Syntax:

```
/*- - - - - */
int   CDK_SOL_REFPT_COOR ( refpt, coor )
/*- - - - - */
P_CDK_SOENTITY    /*
    refpt;        /* I: edge.
double coor[3];    /* O: edge data.
                  /* Return : =0 - OK, <>0 - errors
/*- - - - - */
```

CDK_SOL_SOLID2CURVES_GS

Get the number of edges on a solid object.

Syntax:

```
/*- - - - - */
T_INT CDK_SOL_SOLID2CURVES_GS( sol_uid )
/*- - - - - */
T_INT sol_uid;    /* I: Solid unique ID.
                  /* Return : >0 - Number of edges, <0 - Error
/*- - - - - */
```

CDK_SOL_SOLID2CURVES_OBTAIN

Obtain curves of edges of a given solid object.

Syntax:

```
/*- - - - - */
T_INT CDK_SOL_SOLID2CURVES_OBTAIN( sol_uid, crv_ids )
/*- - - - - */
T_INT sol_uid;    /* I: Solid unique ID.
T_INT *crv_ids;    /* O: Curve IDs. Allocate at least as returned from
                  /* CDK_SOL_SOLID2CURVES_GS().
                  /* Return : =0: OK, <0 - Error
/*- - - - - */
```

UTILITIES

CDK_ATTACH_SCREW

Attach a screw, nut or ejector to a solid entity.

Syntax:

```

/*- - - - - */
int  cdk_attach_screw ( edge_id, type, rad, length, text)
/*- - - - - */
/*
/* Input :
/*
int    edge_id;    /* ID number of the start edge
int    type;        /* 0 = Nut - Edge in plate
/* 1 = Screw - Edge in component
/* 2 = Ejector - Edge in component
double *rad;        /* Nominal (outer) radius (diameter/2) of thread.
/* In MM only. Must be : rad > 0.
double *length;     /* 0 = for Ejector.
/* >0 = for screw and nut, thread length.
char   *text;       /* The field "No" in screw dialog (M10, M-SC12)
/* A NULL terminated string.
/*- - - - - */

```

CDK_BB_LIMITS_AT_UCS

Calculates the bounding box of a solid object in a given UCS.

Syntax:

```

/*- - - - - */
int  cdk_bb_limits_at_uc ( amount, obj_id, sol_wf, ucsid, tol, offset, bb, bc )
/*- - - - - */
/*
/* Input :
/*
int    amount;      /* Amount of entities.
int    *obj_id;      /* Pointer to entities ID vector.
int    *sol_wf;      /* Pointer to vector of flags: solid=0, wireframe=1
int    ucsid;        /* Required UCS ID.
double tol;          /* Tolerance.
float  offset[6];    /* Offset from body (positive numbers)
/* 0,1,2 - to the positive direction of axes
/* 3,4,5 - to the negative direction of axes
/*
/* Output :
/*
double bb[6];        /* Box diagonal (0,1,2 - min values, 3,4,5 - max values)
double bc[3];        /* Box center
/*
/* Return value :
/* OK / ERROR
/*- - - - - */

```

CDK_SOL_BB_CENTER_AT_UCS Calculates the center of a solid object in a given UCS.
Syntax:

```

/*- - - - - */
int  cdk_sol_bb_center_at_ucs ( ipSolObj, iUcsId, oNewCenter )
/*- - - - - */
      /*
      /* Input :
      /*
P_CDK_SOLENTITY ipSolObj /* The solid object ID.
int             iUcsId;  /* The required UCS ID.
      /*
      /* Output :
      /*
double  oNewCenter[3]; /* The center of the solid in the UCS system.
/*- - - - - */

```

CDK_SOL_ENT_CTOR CimaDEK Solid entity constructor.
Syntax :

```

/*- - - - - */
void CDK_SOL_ENT_CTOR ( Solent )
/*- - - - - */
      /* Output:
P_CDK_SOLENTITY Solent; /* The solid entity.
/*- - - - - */

```

CDK_SOL_FND_SCREW_EDGE

Find the ID of the circular edge that coincides with the screw end.

Syntax:

```

/*- - - - - */
int  cdk_sol_fnd_screw_edge ( solid_uid, proc, screw_end, edge_id )
/*- - - - - */
/*
/* Input :
/*
int solid_uid; /* Solid uid
int proc;      /* Procedure to look for
double screw_end[3]; /* Screw end point in MODEL UCS.
/*
/* Output :
/*
int *edge_id;  /* Required edge ID
/*
/* Return value :
/* 0 = OK
/* -1 = Error
/* -2 = Edge not found
/*- - - - - */

```

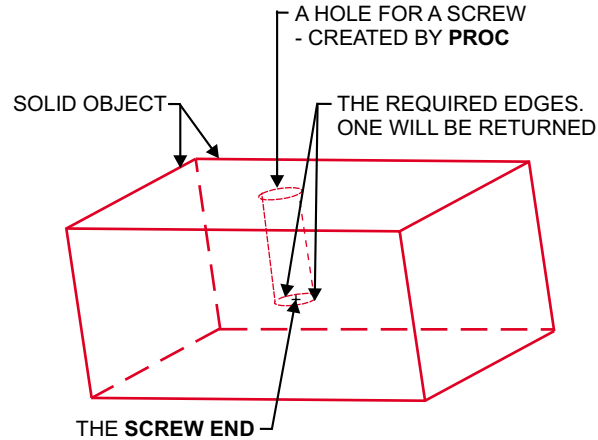


Figure 7-1: CDK_SOL_FND_SCREW_EDGE

CDK_SOL_PIERCE_OBJ

Find all pierce points of a ray and an object.

Syntax :

```

/*- - - - - */
P_CDK_TRACE CDK_SOL_PIERCE_OBJ ( SolObj, Origin, Direction, Num )
/*- - - - - */
/*
/* Input :
/*
T_CDK_SOLENTITY SolObj; /* The solid object.
double Origin[3]; /* Origin point of the ray (MODEL).
double Direction[3]; /* Ray direction (MODEL).
/*
/* Output :
/*
int *Num; /* > 0 : The number of face that was found.
/* < 0 : See Chapter 1, General Concepts, Status.
/* Return value:
/* A pointer to a new buffer of T_CDK_TRACE structures.
/* Use CDK_SOL_TRACE_FREE() after using this function.
/*- - - - - */

```

CDK_SOL_PIERCE_OBJ1

Find all pierce points of a ray and an object and sort them. This function is a newer version of function CDK_SOL_PIERCE_OBJ.

Syntax :

```

/*- - - - - */
void CDK_SOL_PIERCE_OBJ1 ( SolObj, Origin, Direction, oTrace )
/*- - - - - */
/* Input :
/*
T_CDK_SOLENTITY SolObj; /* The solid object.
T_DOUBLE Origin[3]; /* Origin point of the ray (MODEL).
T_DOUBLE Direction[3]; /* A point that is used to define the Ray direction.
/*
/* Output :
/*
P_CDK_TRACE1 oTrace; /* The structure containing the pierce points, sorted.
/*- - - - - */

```

CDK_SOL_SAME

Check if solid entities are the same.

Syntax :

```

/*- - - - - */
int CDK_SOL_SAME ( Solent1, Solent2 )
/*- - - - - */
/* Input:
/*
P_CDK_SOLENTITY Solent1; /* The first solid entity.
P_CDK_SOLENTITY Solent2; /* The second solid entity.
/* Return:
/* TRUE: same entities, else: FALSE
/*- - - - - */

```

CDK_SOL_TRACE_FREE

Free allocated trace buffer.

Syntax :

```

/*- - - - - */
void CDK_SOL_TRACE_FREE ( Trace )
/*- - - - - */
        /*
        /* Input :
        /*
P_CDK_TRACE      /*
    Trace;        /* Trace Buffer to free.
/*- - - - - */

```

CDK_SOL_TRACE_FREE1

Free trace buffer.

Syntax :

```

/*- - - - - */
void CDK_SOL_TRACE_FREE1 ( Trace )
/*- - - - - */
        /*
        /* Input :
        /*
P_CDK_TRACE1      /*
    Trace;        /* Trace Buffer to initialize.
/*- - - - - */

```

CDK_SOL_UPDATE

Update solid entity handling.

Syntax :

```

/*- - - - - */
int CDK_SOL_UPDATE ( Solent )
/*- - - - - */
        /* Input/Output:
P_CDK_SOENTITY Solent;    /* The solid entity.
        /* Return: Status
/*- - - - - */

```

CDK_UCLOSE

Close the current pfm file that was opened in read only mode.

Syntax :

```

/*- - - - - */
void cdk_uclose(ier)
/*- - - - - */
        /*
T_INT *ier;      /* O : Status
/*- - - - - */

```

CDK_UOPEN

Open a pfm file in read only mode.

Syntax :

```

/*- - - - - */
void cdk_uopen(fname,fnlen,ier)
/*- - - - - */
        /*
T_CHAR fname[];    /* I : The file name to read.
T_INT *fnlen;      /* I : The file name length.
T_INT *ier;        /* O : Status.
/*- - - - - */

```

SOLID PROCEDURE DATA ACCESS

CDK_SOL_PROC_ELEM

Read maxNum items with given entity and feat. type

Syntax :

```

/*- - - - - */
int CDK_SOL_PROC_ELEM ( Feattype, Enttype, Maxnum, Elembuf )
/*- - - - - */
/* Input: */
int Feattype; /* The CimaDEK feat type. */
int Enttype; /* The CimaDEK entity type. */
int Maxnum; /* The maximum items to read. */
/* Output: */
P_CDK_SOLENTITY Elembuf; /* Solid elements. */
/* Return: */
/* the number of full items */
/* actually read. */
/*- - - - - */

```

CDK_SOL_PROC_ELEM_NUM

Get the number of items.

Syntax :

```

/*- - - - - */
int CDK_SOL_PROC_ELEM_NUM( featType, entityType )
/*- - - - - */
/* Input : */
int featType; /* CimaDEK feat type. */
int entityType; /* CimaDEK entity type. */
/* Return value: the number of items . */
/*- - - - - */

```

CDK_SOL_PROC_ELEM_PROC

Retreive the procedure for a given solid element.

Syntax:

```

/*- - - - - */
int CDK_SOL_PROC_ELEM_PROC ( elem, Proc )
/*- - - - - */
P_CDK_SOLENTITY elem; /* I: Solid element. */
P_CDK_SOLENTITY Proc; /* O: The procedure. */
/* Return: =0-OK <>0-ERROR */
/*- - - - - */

```

CDK_SOL_PROC_END

Free Procedure Data.

Syntax :

```

/*- - - - - */
void CDK_SOL_PROC_END ( )
/*- - - - - */

```

CDK_SOL_PROC_FIRST Get the first procedure.
Syntax :

```

/*- - - - - */
int CDK_SOL_PROC_FIRST ( Proc )
/*- - - - - */
/*
/* Output :
/*
P_CDK_SOENTITY /*
Proc;          /* procedure.
/* Return: object exist - TRUE, else - FALSE.
/*- - - - - */

```

CDK_SOL_PROC_FREE_DIMENS

Free the dimension buffer

Syntax:

```

/*- - - - - */
void CDK_SOL_PROC_FREE_DIMENS ( dims )
/*- - - - - */
P_CDK_DIMENSION /*
dims;          /* The dimensions buffer.
/*- - - - - */

```

CDK_SOL_PROC_GET_DIMENS

Get feature dimensions.

Syntax :

```

/*- - - - - */
P_CDK_DIMENSION CDK_SOL_PROC_GET_DIMENS ( featEnt, NumDims )
/*- - - - - */
/*
/* Input :
/*
P_CDK_SOENTITY /*
featEnt;       /* I: solid entity
/*
/* Output :
/*
int *NumDims;   /* The number of dimensions found.
/* 0 : ERROR - No dimensions were found.
/* < 0 : ERROR.
/* Returns :
/* A pointer to a new buffer of dimension IDs.
/*- - - - - */

```

CDK_SOL_PROC_INIT

Init Procedure Data.

Syntax :

```

/*- - - - - */
void CDK_SOL_PROC_INIT ( Proc, Status )
/*- - - - - */
/* Input:
P_CDK_SOENTITY Proc; /* The solid procedure.
/* Output:
int* Status;          /* Status.
/*- - - - - */

```

CDK_SOL_PROC_INIT_DIMENS

Allocate and initialize dimension buffer.

Syntax :

```

/*- - - - - */
P_CDK_DIMENSION CDK_SOL_PROC_INIT_DIMENS ( NumDims )
/*- - - - - */
/*
/* Output :
/*
int      NumDims;
/* The number of dimensions to allocate.
/* Returns :
/* A pointer to a new buffer of dimension IDs.
/*- - - - - */

```

CDK_SOL_PROC_LAST Get last procedure.**Syntax :**

```

/*- - - - - */
int CDK_SOL_PROC_LAST ( Proc )
/*- - - - - */
/* Output:
/* The solid procedure.
P_CDK_SOLENTITY Proc;
/* Return: Status
/*- - - - - */

```

CDK_SOL_PROC_NAME Get the name of a procedure.**Syntax :**

```

/*- - - - - */
int CDK_SOL_PROC_NAME ( Proc, name )
/*- - - - - */
/*
/* Input :
/*
P_CDK_SOLENTITY
Proc;
/* procedure.
/*
/* Output :
/*
T_CHAR name[];
/* procedure name.
/* Size of name 20.
/* Return value: 0 -> OK, other -> ERROR
/*- - - - - */

```

CDK_SOL_PROC_NEXT Get the next procedure.**Syntax :**

```

*-----*/
int CDK_SOL_PROC_NEXT ( Proc )
/*- - - - - */
/*
/* Output :
/*
P_CDK_SOLENTITY
Proc;
/* procedure.
/* Return: FALSE - end of the sequence, else - TRUE.
/*- - - - - */

```

SOLID PICK FUNCTIONS

CDK_SOL_GET_PARPT Get parametric point.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_SOL_GET_PARPT ( prompt, opt, parpt )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
char *prompt; /* Prompt
/*
/* Input/Output:
/*
int *opt; /* Option to pick:
/*      1 : END,
/*      2 : MID,
/*      3 : CENTER,
/*      4 : KEY-IN,
/*      5 : INTERS,
/*      6 : PIERCE,
/*      7 : PICK.
/*
/* Output :
/*
P_CDK_PARPNT
parpt; /* Picked point.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_PICK Pick a solid entity.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void CDK_SOL_PICK ( solEty, solObj, response )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Output :
/*
P_CDK_SOLENTITY
solEty; /* picked solid entity
P_CDK_SOLENTITY
solObj; /* referenced solid object
int *response; /* response
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_PICK_ANY_FACE Pick open face or regular face.**Syntax :**

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void CDK_SOL_PICK_ANY_FACE ( Solobj, Face, Response )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/* Output:
/* Picked solid object.
P_CDK_SOLENTITY Solobj;
/* Picked face.
P_CDK_SOLENTITY Face;
/* Response.
int *Response;
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_PICK_ANY_LOOP Pick open loop or regular loop.
Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_ANY_LOOP ( Solobj, Face, Loop, Response )
/*- - - - - */
/* Output: */
P_CDK_SOLENTITY Solobj; /* Picked solid object. */
P_CDK_SOLENTITY Face; /* Picked face. */
P_CDK_SOLENTITY Loop; /* Picked loop. */
int *Response; /* Response. */
/*- - - - - */

```

CDK_SOL_PICK_AXIS Pick a reference axis within given solid object.
Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_AXIS ( Solobj, Axis, Response )
/*- - - - - */
/* Output: */
P_CDK_SOLENTITY Solobj; /* Picked solid object. */
P_CDK_SOLENTITY Axis; /* Picked axis. */
int *Response; /* Response. */
/*- - - - - */

```

CDK_SOL_PICK_EDGE Pick an edge.
Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_EDGE ( Solobj, Edge, Response )
/*- - - - - */
/* Output: */
P_CDK_SOLENTITY Solobj; /* Picked solid object. */
P_CDK_SOLENTITY Edge; /* Picked edge. */
int *Response; /* Response. */
/*- - - - - */

```

CDK_SOL_PICK_ENABLE Enable picking of solid elements
Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_ENABLE ( types )
/*- - - - - */
/* Input : */
/* A bit mask defining enabled types */
int types;
/*- - - - - */

```

CDK_SOL_PICK_FACE

Pick a face.

Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_FACE ( Solobj, Face, Response )
/*- - - - - */

P_CDK_SOLENTITY Solobj;      /* Output: */
P_CDK_SOLENTITY Face;        /* Picked solid object. */
int *Response;               /* Picked face. */
/*- - - - - */

```

CDK_SOL_PICK_FEATURE

Pick a feature.

Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_FEATURE( feature, atten, response )
/*- - - - - */

P_CDK_SOLENTITY feature;      /* Output : */
/* picked feature */
/* Input : */
int atten;                    /* attention option : 0 : OFF, 1: ON */
/* 2 : Include all children */
int *response;                /* Output: response */
/*- - - - - */

```

CDK_SOL_PICK_HOLE

Pick a hole (as part of the feature hole).

Syntax :

```

/*- - - - - */
int CDK_SOL_PICK_HOLE ( solObj, pnt, atten, enable, response )
/*- - - - - */

P_CDK_SOLENTITY solObj;      /* Output : */
double pnt[];                /* picked feature */
/* coordinates of the hole. */
/* Input : */
int atten;                    /* attention option : 0 : OFF, 1: ON */
int enable;                   /* 0 : enable hole only. */
/* 1 : enable any feture & hole. */
/* 2 : enable object & hole. */
int *response;                /* Output: response */
/*- - - - - */

```

CDK_SOL_PICK_LOOP

Pick a loop.

Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_LOOP ( Solobj, Face, Loop, Response )
/*- - - - - */

/* Output: */
P_CDK_SOLENTITY Solobj; /* Picked solid object. */
P_CDK_SOLENTITY Face; /* Picked face. */
P_CDK_SOLENTITY Loop; /* Picked loop. */
int *Response; /* Response. */
/*- - - - - */

```

CDK_SOL_PICK_OBJ

Pick a solid object.

Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_OBJ ( Solobj, Response )
/*- - - - - */

/* Output: */
P_CDK_SOLENTITY Solobj; /* Picked solid object. */
int *Response; /* Response. */
/*- - - - - */

```

CDK_SOL_PICK_PLANE

Pick a plane (planar face or ref. plane).

Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_PLANE ( Solobj, Plane, Response )
/*- - - - - */

/* Output: */
P_CDK_SOLENTITY Solobj; /* Picked solid object. */
P_CDK_SOLENTITY Plane; /* Picked plane. */
int *Response; /* Response. */
/*- - - - - */

```

CDK_SOL_PICK_REF_GEOM

Pick any reference geometry solid entity.

Syntax :

```

/*- - - - - */
void CDK_SOL_PICK_REF_GEOM ( Solobj, Refgeom, Response )
/*- - - - - */

/* Output: */
P_CDK_SOLENTITY Solobj; /* Picked solid object. */
P_CDK_SOLENTITY Refgeom; /* Picked reference entity. */
int *Response; /* Response. */
/*- - - - - */

```



Check the part file type.

Syntax:

```

/*-----*/
int      CDK_PFM_TYPE (fileName)
/*-----*/
/* Input :
/*
char      *fileName;
/* File name, without .pfm extension.
/*
/* Returns:
/*
/* >=0: file type
/* -1: errors
/* -2: file does not exist
/*-----*/

```

Load a solid assembly.

Syntax:

```

/*- - - - - */
int    CDK_ASSM_INIT()
/*- - - - - */
/* Returns: */
/* */
/* 0 - OK, <0 - error. */
/*- - - - - */

```

Terminate a solid assembly.

Syntax:

```
/*- - - - - */  
void CDK_ASSM_TERM()  
/*- - - - - */
```

Get the number of part files.

Syntax:

```

/*- - - - - */
int    CDK_ASSM_PART_NUM ()
/*- - - - - */
/* Returns:
/*
/*
/* >=0 - number of part files, <0 - error.
/*
/*- - - - - */

```

CDK_ASSM_PART_NEXT

Get data for the next part file.

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_ASSM_PART_NEXT (pfData)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /* Output:                                */
                                /*                                          */
P_CDK_ASSM_PFDATA              /*                                          */
    pfData;                    /* The part file's data.            */
                                /*                                          */
                                /* Returns:                          */
                                /*                                          */
                                /* TRUE - OK, FALSE - errors or end of the list. */
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

SOLID ATTRIBUTES

CDK_SOL_BLANK

Blank a solid entity.

Syntax :

```

/*- - - - - */
int CDK_SOL_BLANK ( SolEnt, num )
/*- - - - - */
    /*
    /* Input:
    /*
P_CDK_SOENTITY /*
SolEnt;        /* Solid entities buffer.
int    num;    /* Number of entities in the buffer.
            /* Return value: 0 -> OK, other -> err
/*- - - - - */

```

CDK_SOL_BLANK_ATTRIB

Get attributes of a blanked solid.

Syntax :

```

/*- - - - - */
int CDK_SOL_BLANK_ATTRIB ( SolEnt, Status )
/*- - - - - */
    /*
    /* Input:
    /*
P_CDK_SOENTITY /*
    SolEnt;    /* Solid entities buffer.
            /*
            /* Output:
            /*
int    *Status; /* = 0 OK, <>0 - err.
            /* Return Value: 0 -> unblank, 1 -> blank.
/*- - - - - */

```

CDK_SOL_COL_GET

Get solid color.

Syntax :

```

/*- - - - - */
int CDK_SOL_COL_GET ( SolEnt, color )
/*- - - - - */
    /*
    /* Input:
    /*
P_CDK_SOENTITY /*
    SolEnt;    /* Solid entity.
            /*
            /* Output:
            /*
int    *color; /* Color.
            /* Return value: 0 -> OK, other -> err
/*- - - - - */

```

CDK_SOL_COL_SET

Set solid color.

Syntax :

```

/*- - - - - */
int CDK_SOL_COL_SET ( SolEnt, color )
/*- - - - - */
/*
/* Input:
/*
P_CDK_SOLENTITY
SolEnt;
/* Solid entity.
int color;
/* Color.
/* Return value: 0 -> OK, other -> err
/*- - - - - */

```

CDK_SOL_LATT_GET

Get solid line attributes

Syntax :

```

/*- - - - - */
int CDK_SOL_LATT_GET ( SolEnt, style, pen, color )
/*- - - - - */
/*
/* Input:
/*
P_CDK_SOLENTITY
SolEnt;
/* Solid entity.
/* Output:
/*
int *style;
/* Line style.
int *pen;
/* Pen number.
int *color;
/* Color.
/* Return value: 0 -> OK, other -> err
/*- - - - - */

```

CDK_SOL_LATT_SET

Set solid line attributes.

Syntax :

```

/*- - - - - */
int CDK_SOL_LATT_SET ( SolEnt, style, pen, color )
/*- - - - - */
/*
/* Input:
/*
P_CDK_SOLENTITY
SolEnt;
/* Solid entity.
int style ;
/* Line style
int pen ;
/* Pen number.
int color ;
/* Color.
/* Return value: 0 -> OK, other -> err
/*- - - - - */

```

CDK_SOL_LEVEL_CHANGE

Move the solid object of the given type, to the desired level.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int  cdk_sol_level_change ( type, id, level )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input :
                                /*
                                /* Solid type.
                                /* ID of entity.
                                /* Level to be assigned.
                                /*
                                /* Return value :
                                /*   0 : OK
                                /*  -1 : ERROR
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_LEVEL_GET

Get the solid level.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int  CDK_SOL_LEVEL_GET ( SolEnt, level )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
                                /*
                                /* I : Solid entity.
                                /* I : Level number.
                                /* Return: =0 OK, <>0 - err
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_LEVEL_SET

Set the solid level

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int  CDK_SOL_LEVEL_SET ( SolEnt, level )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
                                /*
                                /* Input:
                                /*
                                /*
                                /* I : Solid entity.
                                /* I : Level number.
                                /* Return: =0 OK, <>0 - err
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_PIERCE_OBJ_SORT

Find all pierce points of a ray and an object.

Syntax:

```

/*- - - - - */
P_CDK_TRACE CDK_SOL_PIERCE_OBJ_SORT ( SolObj, Sort, Origin, Direction, Num )
/*- - - - - */
/*
/* Input :
/*
/*
/* The solid object.
/*
int SolObj; /* Sort option :
/* = 0 : Data is given unsorted, by the face sequence.
/* = 1 : Points are sorted along the ray.
double Origin[3]; /* Origin point of the ray (MODEL).
double Direction[3]; /* Ray direction (MODEL).
/*
/* Output :
/* > 0 : The number of face that was found.
/* < 0 : See General Concepts-Status on page 1-2.
int *Num; /* Returns :
/* A pointer to a new buffer of T_CDK_TRACE structures.
/* Use CDK_SOL_TRACE_FREE() after using this function.
/*- - - - - */

```

CDK_SOL_SLA_INIT

Initialize SLA package.

Syntax:

```

/*- - - - - */
int CDK_SOL_SLA_INIT (tol, sol)
/*- - - - - */
/* Input:
/* the tolerance
double tol;
/*
P_CDK_SOLENTITY
sol; /* solid object
/*
/* Return:
/* >0 - the number of the faces
/* <=0 - error.
/*- - - - - */

```

CDK_SOL_SLA_TRIANG_NEXT

Get triangle data for the next face.

Syntax:

```

/*- - - - - */
int CDK_SOL_SLA_TRIANG_NEXT (faceSLAData, status)
/*- - - - - */
/* Output:
/*
P_CDK_FACE_SLA
faceSLAData; /* Face triangle data
/*
int *status; /* Status:
/* CDK_SLA_OK, CDK_SLA_ERROR,
/* CDK_SLA_OPEN_FACE.
/*
/* Return:
/* =0 - Stop, <>0 - Continue
/*- - - - - */

```

CDK_SOL_TRIM

Delete all procedures after "Proc" from the solid tree.

Syntax:

```

/*- - - - - */
int CDK_SOL_TRIM ( Proc )
/*- - - - - */
      /*
      /* purpose :
      /*
P_CDK_SOENTITY /*
      Proc;      /* I : The procedure.
      /*
      /* R : = 0 - OK, <> 0 - ERROR.
      /*
/*- - - - - */

```

CDK_SOL_UNBLANK

Unblank a solid entity.

Syntax :

```

/*- - - - - */
int CDK_SOL_UNBLANK ( SolEnt, num )
/*- - - - - */
      /*
      /* Input:
      /*
P_CDK_SOENTITY /*
      SolEnt;    /* Solid entities buffer.
int      num;    /* Number of entities in the buffer.
      /* Return value: 0 -> OK, other -> err
/*- - - - - */

```

SOLID DISPLAY

CDK_SOL_ATTENTION_OFF Turn attention mode OFF

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void CDK_SOL_ATTENTION_OFF ( solEnt )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
/*
P_CDK_SOLENTITY /*
SolEnt; /* solid entity
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_ATTENTION_ON Turn attention mode ON.

Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
void CDK_SOL_ATTENTION_ON ( solEnt )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input:
/*
/*
P_CDK_SOLENTITY /*
SolEnt; /* solid entity
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_ATTEN_HOLE_OFF Turn hole attention mode OFF.

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_SOL_ATTEN_HOLE_OFF (Proc,pnt)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
P_CDK_SOLENTITY /*
Proc; /* I: solid entity.
double pnt[]; /* I: coordinates of the hole (MODEL).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_ATTEN_HOLE_ON Turn hole attention mode ON.

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_SOL_ATTEN_HOLE_ON (Proc,pnt)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */
P_CDK_SOLENTITY /*
Proc; /* I: solid entity.
double pnt[]; /* I: coordinates of the hole (MODEL).
/*- - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_DISP_RESUME

Unsuspend the display.

Syntax :

```

/*- - - - - */
void CDK_SOL_DISP_RESUME ( )
/*- - - - - */

```

CDK_SOL_DISP_SUSPEND

Temporarily suspend the display.

Syntax :

```

/*- - - - - */
void CDK_SOL_DISP_SUSPEND ( )
/*- - - - - */

```

CDK_SOL_DISPLAY_OFF

Delete a solid entity from the display file.

Syntax :

```

/*- - - - - */
int CDK_SOL_DISPLAY_OFF ( solEnt )
/*- - - - - */
/*
/* Input :
/*
P_CDK_SOLENTITY
solEnt;
/* Solid entity .
/* Return value: 0 -> OK, other -> ERROR.
/*- - - - - */

```

CDK_SOL_DISPLAY_ON

Create a solid entity in the display file.

Syntax :

```

/*- - - - - */
int CDK_SOL_DISPLAY_ON ( solEnt )
/*- - - - - */
/*
/* Input :
/*
P_CDK_SOLENTITY
solEnt;
/* Solid entity.
/* Return value: 0 -> OK, other -> ERROR.
/*- - - - - */

```

CDK_SOL_DISPLAY_UPDATE

Update the display of a solid entity.

Syntax :

```

/*- - - - - */
int CDK_SOL_DISPLAY_UPDATE ( solEnt )
/*- - - - - */
/*
/* Input :
/*
P_CDK_SOLENTITY
solEnt;
/* Solid entity to update.
/* Return value: 0 -> OK, other -> ERROR.
/*- - - - - */

```

CDK_SOL_DISPLAY_UPDATE_FULL

Updates the entire solid display.

Syntax:

```
/*- - - - - */
void CDK_SOL_DISPLAY_UPDATE_FULL ( )
/*- - - - - */
```

CDK_SOL_PROC_ATTEN_OFF

Turn feature attention mode OFF.

Syntax :

```
/*- - - - - */
void CDK_SOL_PROC_ATTEN_OFF( featEnt )
/*- - - - - */
/*
/* Input:
/*
P_CDK_SOLENTITY
FeatEnt; /* solid entity
/*- - - - - */
```

CDK_SOL_PROC_ATTEN_ON

Turn feature attention mode ON.

Syntax :

```
/*- - - - - */
void CDK_SOL_PROC_ATTEN_ON( featEnt,children )
/*- - - - - */
/*
/* Input:
/*
P_CDK_SOLENTITY
FeatEnt; /* solid entity
int children; /* TRUE with children, FALSE only the feature.
/*- - - - - */
```

CDK_SOL_PREVIEW_CLOSE

Close the display and the imported file.

Syntax:

```
/*- - - - - */
void CDK_SOL_PREVIEW_CLOSE ( Status )
/*- - - - - */
int *Status; /* 0 : Status = 0 : OK, <> 0 : ERROR.
/*- - - - - */
```

CDK_SOL_PREVIEW_FREE

Free all allocated memory.

Syntax:

```
/*- - - - - */
void CDK_SOL_PREVIEW_FREE ( )
/*- - - - - */
```

CDK_SOL_PREVIEW_INIT

Initialize and display a preview of an imported file.

Syntax:

```

/*- - - - - */
void CDK_SOL_PREVIEW_INIT ( importData, Status )
/*- - - - - */
P_CDK_IMPORT      /*
importData;      /* I : Import  Data.
int  *Status;     /* O : Status  = 0 : OK,  <> 0 : ERROR.
/*- - - - - */
Note: This function allocates memory, use CDK_SOL_PREVIEW_FREE to free this
memory.

```

CDK_SOL_PREVIEW_LEVELS

Returns a list of levels in an external pfm file that contains objects in them.

Syntax:

```

/*- - - - - */
void CDK_SOL_PREVIEW_LEVELS ( importData, iLevels, Status )
/*- - - - - */
P_CDK_IMPORT      /*
importData;      /* I : Import  Data.
P_CDK_IMPORT_LEVELS /*
iLevels;        /* O : The levels table.
P_INT  Status;   /* O : Status  = 0 : OK,  <> 0 : ERROR.
/*- - - - - */
Note: This function allocates memory, use CDK_SOL_PREVIEW_LEVELS_FREE to free
this memory.

```

CDK_SOL_PREVIEW_LEVELS_FREE

Free the memory of the allocated P_CDK_IMPORT_LEVELS.

Syntax:

```

/*- - - - - */
void CDK_SOL_PREVIEW_LEVELS_FREE ( iLevels, Status )
/*- - - - - */
P_CDK_IMPORT_LEVELS /*
iLevels;           /* O : The levels table.
P_INT  Status;     /* O : Status  = 0 : OK,  <> 0 : ERROR.
/*- - - - - */

```

CDK_SOL_PREVIEW_MOVE

Move the display list of objects.

Syntax:

```

/*- - - - - */
void CDK_SOL_PREVIEW_MOVE ( importData, Status )
/*- - - - - */
P_CDK_IMPORT      /*
importData;      /* I : Import  Data.
int  *Status;     /* O : Status  = 0 : OK,  <> 0 : ERROR.
/*- - - - - */

```

CDK_SOL_PREVIEW_PICK

Set all objects as pickable / unpickable

Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
void CDK_SOL_PREVIEW_PICK ( solObj, response )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
P_CDK_SOLENTITY          /*
    solObj;               /* 0: picked solid object
int    *response;         /* 0: response
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

SOLID EDIT

CDK_SOL_DEL_PROC Delete a procedure (feature).
Syntax :

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_SOL_DEL_PROC ( Proc, DispMode )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* Input :
/*
P_CDK_SOLENTITY
/* The procedure to be deleted.
Proc;
/* Display mode :
int DispMode;
/* = 0 : Do not update the display.
/* = 1 : Update the display.
/* Return value:
/* See General Concepts-Status on page 1-2.
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

Notes : - Deletion of feature x will delete all features created after x,
 and those that are related to x.
 - To delete a solid object that was created using the "new" option,
 delete the first feature that creates that solid object.

WARNING : - Deletion of the first feature will delete ALL solid entities !!!

CDK_SOL_EDSK_END Close an open Sketcher without resetting it.
Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
void CDK_SOL_EDSK_END ( )
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_EDSK_GET_NUM Get the number of Sketchers within a procedure.
Syntax:

```

/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
int CDK_SOL_EDSK_GET_NUM (Proc)
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*
/* purpose :
/*
P_CDK_SOLENTITY
/* I : The procedure.
Proc;
/*
/* R : > 0 - the number of sketchers.
/* < 0 - ERROR.
/*
/*- - - - - - - - - - - - - - - - - - - - - - - - - - - */

```

CDK_SOL_EDSK_REGEN

Regenerate solid after resketch.

Syntax:

```

/*- - - - - */
int CDK_SOL_EDSK_REGEN ( )
/*- - - - - */
/* R : = 0 - OK, <> 0 - ERROR. */
/*- - - - - */

```

CDK_SOL_EDSK_SET

Reset a Sketcher of a procedure by an index.

Syntax:

```

/*- - - - - */
int CDK_SOL_EDSK_SET ( sketch )
/*- - - - - */
/*
/* purpose :
/*
P_CDK_SKETCH
sketch; /* I : The sketcher.
/* R : = 0 - OK, <> 0 - ERROR.
/*- - - - - */
NOTE: This function can only be called after CDK_SOL_EDSK_GET was called.

```

CDK_SOL_EXPLODE_GS

Get the number of faces when a given solid is exploded. Define which entities are to be exploded.

This number is used for calculating the size of array **surf ids** in utility **CDK_SOL_EXPLODE_OBTAIN** (page 7-48).

Syntax:

```

/*- - - - - */
int cdk_sol_explode_gs ( solid_uid, proc_num, proc )
/*- - - - - */
/*
/* Input :
/*
int solid_uid; /* Solid uid
int proc_num; /* > 0 = Explode according to Procedures list
/* 0 = Explode only REVOLVE and HOLE
/* < 0 = Explode the whole solid
int *proc; /* Procedures list to exclude
/*
/* Return value :
/* Number of faces
/* -1 = Error
/*- - - - - */

```

CDK_SOL_EXPLODE_OBTAIN Explode a solid to a set number of faces.

The number of faces is calculated by utility **CDK_SOL_EXPLODE_GS** (page 7-48) where a user defines which entities are to be exploded.

Syntax:

```

/*- - - - - */
int  cdk_sol_explode_obtain ( tol, solid_uid, proc_num, proc, surf_ids )
/*- - - - - */
/*
/* Input :
/*
/* Tolerance for explode
double  tol;
/* Solid uid
int  solid_uid;
/* > 0 = Explode according to Procedures list
int  proc_num;
/* 0 = Explode only REVOLVE and HOLE
/* < 0 = Explode the whole solid
int  *proc;
/* Procedures list to exclude
/*
/* Output :
/*
/* Faces list
int  *surf_ids;
/*
/* Return value :
/* 0 = OK, <>0 = Error
/*- - - - - */

```

CDK_SOL_EXPLODE_SOLID Explode a solid to a set number of surfaces at a specified distance from its faces.

The number of surfaces is calculated by utility **CDK_SOL_EXPLODE_SOLID_GS** (page 7-50).

Syntax:

```

/*- - - - - */
int  cdk_sol_explode_solid ( tol, offset, solid_uid, surf_ids, surf_num )
/*- - - - - */
/*
/* Input :
/*
/* Tolerance (polygon approx.)
double  tol;
/* Offset to explode with
double  offset;
/* Solid uid
int  solid_uid;
/*
/* Output :
/*
/* Faces list
int  *surf_ids;
/* Number of created surfaces
int  *surf_num;
/*
/* Return value :
/* 0 = OK, <>0 = Error
/*- - - - - */

```

CDK_SOL_EXPLODE_SOLID_GS

Get the number of faces for exploding a given solid.

This number is used for calculating the size of array **surf ids** in utility

CDK_SOL_EXPLODE_SOLID (page 7-48).

Syntax:

```
/*- - - - - */
int cdk_sol_explode_solid_gs ( solid_uid )
/*- - - - - */
/*
/* Input :
/*
int solid_uid; /* Solid uid
/*
/* Return value :
/*      Number of faces
/*      -1 = Error
/*- - - - - */
```

CDK_SOL_MODIFY_DIMENSION

Edit procedure parameters.

Syntax:

```
/*- - - - - */
int CDK_SOL_MODIFY_DIMENSION ( DispMode, Dim, numDim )
/*- - - - - */
/*
/* Input :
/*
int DispMode; /* Display mode :
/*      = 0 : Do not update the display.
/*      = 1 : Update the display.
/*
P_CDK_DIMENSION
Dim; /* A modifying dimension buffer.
int numDim; /* Buffer size.
/* Returns :
/* See General Concepts-Status on page 1-2.
/*- - - - - */
```

TRANSLATE WF -> SOLID

CDK_CYLINDER_RAY_TRACE

Find the cutting curves defined by a cylinder on a group of surfaces. These cutting curves are used by utility **CDK_EJECTOR_CUT** (page 7-52) to create an open solid object. Utility **CDK_EJECTOR_CUT2** (page 7-52) uses this open solid object to cut the solid cylinder (ejector).

Syntax:

```

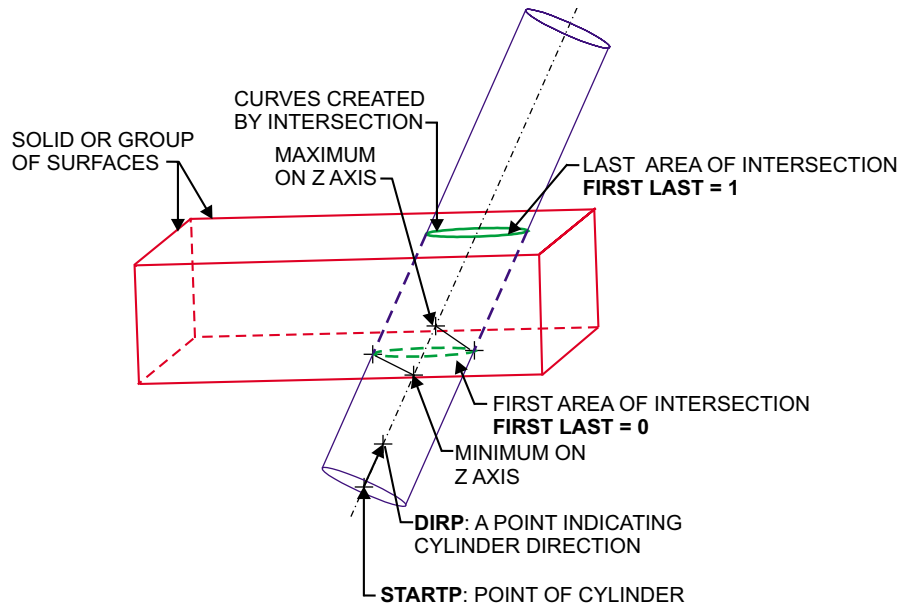
/*- - - - - */
T_INT cdk_cylinder_ray_trace (startp, dirp, rad, height, first_last, in_id,
                             flagsol, tol, tucs, min, max, cur_ids, trs_ids,
                             trsnum, curnum)
/*- - - - - */

/* Input : */
T_DOUBLE startp[]; /* Starting point (point 0,0,0 of the cylinder in model UCS) */
T_DOUBLE dirp[]; /* Direction on Z axis (in model UCS) */
T_DOUBLE rad; /* Radius of cylinder */
T_DOUBLE height; /* Cylinder max height */
T_INT first_last; /* Flag indicating point of intersection (0-first, 1-last) */
T_INT in_id; /* ID of group surface OR UID of solid */
T_INT flagsol; /* Flag to indicate solid/surface group (0-surface 1-solid) */
T_DOUBLE tol; /* Tolerance */

/* Output : */
T_INT*tucs; /* Temporary UCS ID (from ray trace) */
T_DOUBLE *min; /* Minimum on Z axis of cylinder cut (in mold UCS) */
T_DOUBLE *max; /* Maximum on Z axis of cylinder cut (in mold UCS) */
T_INT cur_ids[]; /* Vector of curves created by the trace */
T_INT trs_ids[]; /* Vector of trimmed surfaces created by the trace */
T_INT *trsnum; /* Number of trimmed surfaces created in trs_ids[]; */
T_INT *curnum; /* Number of curves created in cur_ids */

/* Return value : */
/* 0=O.K. */
/* 1=O.K.Perpendicular flat surface (no cut will be done) */
/* -1=ERROR: Cutting curve is not a closed curve */
/* -2=ERROR: Projected cutting curves don't form a circle */
/* -3=ERROR: Too many curves per one surface were created */
/* -4=ERROR: Problem with the data of exploded surfaces */
/*- - - - - */

```

**Figure 7-2: CDK_CYLINDER_RAY_TRACE**

CDK_EJECTOR_CUT

Create an open solid object according to the curves created by utility **CDK_CYLINDER_RAY_TRACE** (page 7-51).

Syntax:

```

/*- - - - - */
T_INT cdk_ejector_cut (cur_ids, trs_ids, curnum, trsnum, tol, tucs, datum)
/*- - - - - */
/*
/* Input :
/*
T_INT cur_ids[]; /* Vector of curve ID's (from cdk_cylinder_ray_trace) */
T_INT trs_ids[]; /* Vector of trimmed surfaces (from cdk_cylinder_ray_trace) */
T_INT curnum; /* Number of curves created in cur_ids */
T_INT trsnum; /* Number of trimmed surfaces in trs_ids */
T_DOUBLE tol; /* Tolerance */
T_INT*tucs; /* Temporary UCS (from ray trace) */
/*
/* Output :
/*
T_CDK_SOLENTITY *datum; /* Solid datum surface for cut-extrude */
/*
/* Return value :
/* 0 = O.K.
/* -1 = ERROR: no solid body was created
/* -2 = ERROR: boolean remove did not succeed
/* -3 = ERROR: no datum solid was created
/*- - - - - */

```

CDK_EJECTOR_CUT2

Cut the ejector (solid cylinder) with the open solid object created by utility **CDK_EJECTOR_CUT** (page 7-52).

Syntax:

```

/*- - - - - */
T_INT cdk_ejector_cut2 (SolEnt, cur_ids, tol, tucs, solid)
/*- - - - - */
/*
/* Input :
/*
T_CDK_SOLENTITY SolEnt; /* The solid object ID (ejector). */
T_INT cur_ids[]; /* Vector of curve ID's (from cdk_cylinder_ray_trace) */
T_DOUBLE tol; /* Tolerance */
T_INT*tucs; /* Temporary UCS (from ray trace) */
T_CDK_SOLENTITY solid; /* Solid open face for boolean remove */
/*
/* Return value :
/* 0 = O.K.
/* -1 = ERROR: no solid body was created
/* -2 = ERROR: boolean remove did not succeed
/* -3 = ERROR: no datum solid was created
/*- - - - - */

```

CDK_EJECTOR_CUT_CLEAN Delete temporary entities after ejector cut. See **CDK_EJECTOR_CUT** (page 7-52).
Syntax:

```

/*- - - - - */
void cdk_ejector_cut_clean (cur_ids, trs_ids, tucs, curnum, trsnum)
/*- - - - - */
/*
/* Input :
/*
T_INT cur_ids[]; /* Vector of curve ID's (product of cdk_cylinder_ray_trace) */
T_INT trs_ids[]; /* Vector of trimmed surfaces (out of cdk_cylinder_ray_trace) */
T_INT tucs;      /* Temporary ucs of the cylinder
T_INT curnum;    /* Number of curves created in cur_ids
T_INT trsnum;    /* Number of trimmed surfaces in trs_ids
/*- - - - - */

```

CDK_SOL_SRF2SOL Create solid object from surfaces.
Syntax:

```

/*- - - - - */
void CDK_SOL_SRF2SOL ( srf2solData, Status )
/*- - - - - */
P_CDK_SRF2SOL /*
srf2solData; /* I : The surface data.
int *Status; /* O : Status.
/*- - - - - */

```

CDK_SOL_WF2DATUM Translate wireframe entities to solid datum.
Syntax:

```

/*- - - - - */
void CDK_SOL_WF2DATUM ( wf2datumData, Status )
/*- - - - - */
P_CDK_WF2DATUM /*
wf2datumData; /* I : The WF data.
int *Status; /* O : Status.
/*- - - - - */

```

CDK_SRF_TRACE Perform a Ray Trace on a group of surfaces.
Syntax:

```

/*- - - - - */
T_INT cdk_srf_trace (srf_id, pos, dir, first_last, point)
/*- - - - - */
/*
/* Input :
/*
T_INT srf_id; /* Surface ID (one or group)
T_DOUBLE *pos; /* Start point of the ray(MODEL)
T_DOUBLE *dir; /* Direction point of the ray(MODEL)
T_INT first_last; /* Flag to indicate: first=0, last=1
/*
/* Output :
/*
T_DOUBLE *point; /* The pierce point on the surface(MODEL)
/*
/* Return value :
/* 0 = O.K.
/* <0 = ERROR
/*- - - - - */

```



Section V

External User Package



Chapter 8

External User Package

Introduction

This chapter contains user routines associated with the EUSYS application. The External User package is contained in the directory **eusys**.

This package enables you to create External User programs which may be run out of Cimatron^{it} environment. All functions available in the Cimatron^{it} User package may be used for writing External User programs, with the exception of the interactive functions (such as MODI, RTEXT, UPICK etc.).

All Cimatron^{it} interactive routines, or routines requiring a user response, are not applicable in the External User Package. All other user package routines may be used.

Commands

The following commands are used:

- eumake** - to compile and link External User programs.
- eumake -c** or **eucomp** - to compile External User programs.
- eumake -l** or **eulink** - to link External User programs.

The result of compiling and linking the External User program will be an executable file in the current directory. For additional information on commands, see Operating Instructions in Chapter 2.

Development Authorization

The following codes are required:

- x_user_dev** - to develop and run External User programs.
- x_user_run** - to run External User programs.

Routines for Accessing Part Files

EUREAD

Check and read a part file.

Syntax :

```

/*- - - - - */
void EUREAD ( Pname, Pnlen, Exmode, Status )
/*- - - - - */
/*
/* Input :
/*
char *Pname; /* Name of file to be read ( without the extension ).
int *Pnlen; /* 0 : Use LEN ( PNAME )
/* > 0 : Length of PNAME
int *Exmode; /* 0 : Only existing files.
/* > 0 : Allow new files:
/* 1 : MM
/* 2 : CM
/* 3 : METER
/* 4 : INCH
/* 5 : FEET
/* < 0 : Similar to > 0 but start new file.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```

EUSAVE

Save a part file.

Syntax :

```

/*- - - - - */
void EUSAVE ( Pname, Pnlen, Mode, Status )
/*- - - - - */
/*
/* Input :
/*
char *Pname; /* Name of the file to be saved (without the extension).
int *Pnlen; /* 0 : Current part file.
/* > 0 : Length of PNAME.
int *Mode; /* 1 : With backup.
/* 2 : Without backup.
/*
/* Output :
/*
int *Status; /* See General Concepts-Status on page 1-2.
/*- - - - - */

```



Section VI

EDMSLink



Chapter 9

EDMSLink

Introduction

The EDMS gives data management systems access to operations done within Cimatron^{it} such as:

- opening or closing part files
- placing external references (such as **PLACE >> SUB-ASSM** and **EXTRACT**).

Use **EDMSLink** to automatically invoke one or more **EDMSLink** programs every time you enter or exit part files.

The **EDMSLink** consists of two levels:

Basic Level - USEREXEC Mechanism.

Provides an “interrupt” in the Cimatron^{it} interactive application at every occurrence of loading and saving a part file. During this interrupt, the predefined interactive or non-interactive functions are run. Control is automatically returned to the interrupted system after the last function has been run.

Advanced Levels.

Provides an “interrupt” in the Cimatron^{it} interactive application at every occurrence of the prompt **ENTER PART FILE NAME**. During this interrupt, control is passed to an external system, such as EDMS, to continue the operation. Upon termination, the EDMS system returns control to the interrupted system.

EDMSLink programs should be written using the same rules as for other **CimaDEK** programs and any Internal or External User Package function may be used. Below is a list of some of the most useful functions. Refer to the appropriate section in this manual for a full description of each function.

UGTDTM	Get the current date and time.
UGTNID	Get the node ID.
UGTPFN	Get the name of the current part file.
UGVNAM	Get view name and type.
UGTMNM	Obtain the name of a master.
UGTMID	Obtain the ID of a master.
UGTNTP	Get the next tool path ID.
OBTPNM	Obtain the name of a tool path.

Internal User Routines:

AXI_TYPE_OPEN	Get the type of the active part.
UAXI_NUM ASM_COMPS	Get the number of assembled components on all levels.
UAXI_INDEX_ DATA	Get the data of the components of the assembly tree.
UGTERF	Get the list of external references in the current .pfm .
UGTNMERF	Get the number of external references in the current .pfm .

External User Routines

INIT_PATH_CHG	Initialize the number of external reference changes.
ADD_PATH_CHG	Add old and new pathnames.
DOCHPN	Change pathnames.
DOCHPN1	Change wireframe pathnames.
DOCHPN2	Change solid pathnames.

Commands

The following commands are used:

Internal EDMSLink:

lumake	- to compile and link EDMSLink programs.
lumake -c or lucomp	- to compile EDMSLink programs.
lumake -l or lulink	- to link EDMSLink programs.

External EDMSLink:

elumake	- to compile <i>and</i> link EDMSLink programs.
elumake -c or elucomp	- to compile EDMSLink programs.
elumake -l or elulink	- to link EDMSLink programs.

The result of compiling and linking will be an executable file in the directory **<root_cad>\user**. For additional information on commands, see Operating Instructions in Chapter 2.

Development Authorization

The following codes are required:

Internal EDMSLink:

user_dev and **edmslink** - to develop and run EDMSLink programs.
edmslink - to run EDMSLink programs.

External EDMSLink:

x_user_dev and **edmslink** - to develop and run External User programs.
edmslink - to run External User programs.

See Chapter 10 for an example of the External User program.

Basic Level - USEREXEC Mechanism

Concept

The USEREXEC mechanism can be accessed during:

- opening or closing part files
- placing external references (such as PLACE >> SUB-ASSM and EXTRACT).

The names and sequence of predefined interactive or non-interactive functions run by the interactive system at this level, are entered in the file **userexec.dat** described below. This file resides in the directory **<root_cad>/cadsys**.

Running the Basic Level

The Basic Level can be run if you load Cimatron^{it} using the command:

cimit -eu or **cimit -exuser**.

userexec.dat File Structure

The following is an example of the **userexec.dat** file:

```
120, \, \, 'USEREXEC', 2, Example
    9, \ld_tree', \ld_tree', 3, \ld_tree'
    11, \fp', \fp', 3, \fp'
999, \, \, 'END', 0, End-Of-File
```

Data lines format:

```
MODE, 'NAME1' , 'NAME2' , TYPE, 'name3'
```

where:

MODE = 1 - Run after loading part file ONLY IN interactive mode

Advanced Levels

- 2 - Run before saving part file ONLY IN interactive mode
- 3 - Run after loading part file BOTH IN interactive AND non-interactive (EXTRACT, . . .) modes
- 4 - Run before saving part file BOTH IN interactive AND non-interactive modes
- 5 - Run after saving part file ONLY IN interactive mode
- 6 - Run after saving part file BOTH IN interactive AND non-interactive (EXTRACT, . . .) modes
- 7 - Run after close/abandon
- 8 - Before exit quit or continue
- 9 - After open assembly, when assembly is loaded
- 10 - Before edms hook, after interface file is written
- 11 - After edms hook

NAME1 = Actual name of function / segment

NAME2 = Name to be displayed
(relevant in modes 1 and 2 only)

TYPE = 0 - Local function
1 - System segment
3 - User segment

NAME3 = translated name to be displayed if language is not ENGLISH
(relevant in modes 1 and 2 only)

Notes:

- This procedure applies only to the interactive system. It will not be applied to part files created or updated through external programs.
- The usual Cimatron^{it} file search path applies to the file. See Chapter 1 of the **Fundamentals & General Functions Manual**.
- Format of header line, trailer line, and data lines (except MODE) is identical to **classes.dat**, **keysdef.dat**, etc.
- Only type 3 (user segment) is allowed for modes 3 and 4. User segments written for this purpose *should not contain any interaction*.
- Local functions and system segments will be executed only if they are available in the current context.
- The userexec mechanism cannot be used without authorization codes.

Advanced Levels

Concept

The Advanced Levels can be accessed during:

- opening or closing part files
- placing external references (such as PLACE >> SUB-ASSM and EXTRACT).

The communication protocol is based on the following assumptions:

- Cimatron^{it} is invoked by the EDMS system.
- Communication between the two systems is initiated only by Cimatron^{it}.

- The EDMS system “listens” constantly for service requests (see below) from Cimatron^{it}.

Files Created

Messages are sent and received through regular ASCII files. There are four files involved in the communication protocol:

edms_<hostname>.c2m Contains messages sent from Cimatron^{it} to the EDMS system.

edms_<hostname>.m2c Contains messages sent to Cimatron^{it} by the EDMS system.

edms_<hostname>.log Contains a log of all the messages sent and received. This file is maintained by Cimatron^{it}, and can be deleted periodically.

edms_<hostname>.tmp A temporary file used by Cimatron^{it} only.

Notes: • **cimit -edms_file <full_path_name_excluding_extension>**

All files are created, by default, in the <root_work> directory.

You can change the file location and the file name prefix by using the command:

cimit -edms_file <full_path_name_excluding_extension>.

For example, the command:

cimit -edms_file \udd\ed

will create the four files (mentioned above) in the directory **udd** with the prefix **ed** (instead of **edms_<hostname>**).

This option is useful if you are using a PC-DOS system with the 8.3 file name length restriction.

- At every new protocol connection, the files **.c2m** and **.m2c** are overwritten.

Service Requests

Cimatron^{it} to EDMS (**edms_<hostname>.c2m**)

The following types of service requests are sent by Cimatron^{it} to the EDMS:

\$active	Request for the name of the active part file you wish to create/update. The next line in the .c2m file contains the number of currently open work files.
\$reference	Request for the name of a part file to be referenced as sub-assembly, catalog, etc..
\$extract	Request for the name of a part file to be created/updated in an EXTRACT operation.
\$import	Request for the name of a part file to be referenced in an IMPORT operation.
\$export	Request for the name of a part file to be referenced in an EXPORT operation.
\$unknown	Unknown request type.
\$plot	Request for a name of a plot file to be created during a plot operation. If you work with edms_level >=EDLEV1 , and use the file device.sdf , there is a plotter named : "\$EDMS". When plotting to this device, Cimatron ^{it} enters the "\$PLOT" hook. In this case, after the line : \$EDMS, we have the following lines: ActiveFileName, ActiveDocumentType(MODEL - "", VIEW-"1", DRAWING-"2") ActiveDocumentName.

EDMS to Cimatron^{it} (**edms_<hostname>.m2c**)

The following types of instructions are sent by the EDMS to Cimatron^{it}:

\$terminate	Terminate the Cimatron ^{it} session and exit. This instruction can be executed only if the request is \$active and there is only one part file currently open.
\$partfile	A part file name was selected. The name of the selected part file (excluding the .pfm extension) is on the next line in the .m2c file.
\$abort	Abort this request.
\$ignore	Ignore this request.
\$exit	Simulate the Cimatron ^{it} EXIT function.
\$reject	Simulate the Cimatron ^{it} REJECT function.

Log file (edms_<hostname>.log)

The following type of instruction is written in the log file:

\$starting This signifies that a Cimatron^{it} session has started.

Running the Advanced Levels

The Advanced Levels can be run if you load Cimatron^{it} using the command:

command cimit -edms_level <NN>

where <NN> specifies the EDMS level you wish to operate on, and can receive one of the following values:

- 11** No service requests are supported.
If \$edms is entered: every part file name is taken from the EDMS system, for *all* the **.c2m** service requests (active, reference, extract, etc.).

- 21** Every part file name is taken from the EDMS system, *only* for **active** service requests (opening a part file).
If \$edms is entered: every part file name is taken from the EDMS system, for *all* the **.c2m** service requests (active, reference, extract, etc.).

- 31** Every part file name is taken from the EDMS system, for all the **.c2m** service requests (active, reference, extract, etc.). ☐

Index by Routines

Chapter 3 General Routines

ATTRIB

ATDEID 3-3

Attach/detach a record, whose ID is specified, to/from a selected geometric entry.

ATGECO 3-4

Retrieve the total number of geometric entities attached to all records in this table, or to records which have a given value attached to a specified field position number.

ATSUMU 3-4

Given the ID of a record type table and a field position number; find the sum of all the values in this field for all records in this table which are attached to geometric entities.

ATTDET 3-5

Attach/detach a record of a record type table whose name and length are specified, to/from a selected geometric entity.

ATTINI 3-5

Prepare the database for use with ATTRIB subroutines.

GTABID 3-5

Given the name of a record type table, retrieve its ID number.

GTABNA 3-6

Given the ID number of a record type table, retrieve the name of the record type.

GTATTR 3-6

Given an ID number of a geometric entity, retrieve ID numbers of record type tables attached to the entity and pair them with the appropriate record number of the records in each.

GTFLNA 3-7

For a given record type table named TYPNAM, output the names of its fields and indicate the type for each.

GTFLV1 3-7

Given the ID of a record type table, a sequential number in the table and a field position number within the record; retrieve the value in the field.

GTFLVL 3-8

Retrieve the value (contents) of a given field in a given record.

GTGEOM	3-8
Given the ID of a record type table and the sequential number of a record in the table; retrieve the ID numbers of the geometric entities attached to it.	
GTRCNU	3-9
Given the contents of a record, retrieve its table and the number which indicates its position in the table.	
GTTYPE	3-9
Given the ID of a record type table and a field position number, retrieve the type of the field.	
IGTABID	3-10
Given the name of a record type table, retrieve its ID number.	
IGTATTR	3-10
Given an ID number of a geometric entity, retrieve ID numbers of record type tables attached to the entity and pair them with the appropriate record number of the records in each.	
IGTGEOM	3-11
Given the ID of a record type table and the sequential number of a record in the table; retrieve the ID numbers of the geometric entities attached to it.	
NGCPAT	3-11
Attach to an entity all NGD records attached to another entity.	
PREREC	3-12
Construct a string (OUTSTR), representing the contents of a record by appending field values and inserting @ between them.	
RECCRE	3-12
Create a new record in a specified record type table.	
RECDEL	3-13
Delete a record from a specified record type table.	
RECUPD	3-13
Replace an existing record in a specified record type table with a new one.	
RETYIN	3-14
Given the ID of a record type table, retrieve the total number of records in the table.	
DISPLAY	
CDK_RUBBER_CIRCLE	3-15
Draws a rubber circle.	

CDK_RUBBER_LINE	3-15
Draws a rubber line.	
CDK_WINDOW_ACTIVE	3-15
Activate an existing window.	
CDK_WINDOW_DELETE	3-16
Delete an existing window.	
CDK_WINDOW_MULTY	3-16
Recall existing multi-window layout.	
CDK_WINDOW_NEW_DIVISION	3-16
Define new windows by a division point.	
CDK_WINDOW_NEW_NUMERIC	3-17
Define new windows by numeric divide.	
CDK_WINDOW_SINGLE	3-17
Change to a single window.	
CDK_WINDOW_SUB_DIVIDE	3-17
Subdivide a given window by a given division point.	
ENTATT	
EGATTR	3-18
Get the line attributes of an entity.	
EGBLNK	3-18
Check if a given entity is blanked.	
EGLEVL	3-19
Get the level of an entity.	
EGNGD	3-19
Check if NGD is attached to this entity.	
EGTRAN	3-19
Get the matrix transformation of an entity.	
EVISM	3-20
Get/set visibility mask from/to the entity ID.	
OBATTR	3-20
Obtain entity attribute. The only valid attributes are:	
UEVISM	3-21
Get visibility mask of entity ID	
INTERACTION	
CDK_FILE_BROWSE	3-23
Interactive file browser.	

CDK_HOT_VIEW	3-23
Change the view picture (front, side, iso, top) according to the hot-key pressed.	
CDK_LEVEL_ACTIVATE.	3-23
Activate a predefined level.	
CDK_LEVELS_DSP_MASK	3-24
Obtain displayed level mask of a view port.	
CDK_LEVELS_ENT_MASK	3-24
Obtain level mask of all entities.	
CDK_MENU_POPUP_DEF	3-24
Invoke a dynamic popup menu with options of line and column definition.	
CDK_MODAL_SET_COLORS.	3-25
Set the colors for a table of modals.	
CDK_MODAL_SET_SIZE	3-25
Set number of modals and size for the next row of modals.	
DBLNK	3-25
Display a number (NUMBER) of blanks in the interaction area.	
DISPOF.	3-26
Stop display of only those entities created after this subroutine is activated.	
DISPON.	3-26
Enable display of entities created after this routine is activated. Disable the previously activated DISPOF subroutine.	
DMATT	3-26
Change selection mark on button of dynamic menu.	
DMDEF	3-26
Define a dynamic menu field and display it immediately.	
DMDEFN	3-27
Define a dynamic menu field and display it immediately. When working with the option '-lang other', the field name is translated.	
DMINP	3-27
Select dynamic menu item, Immediate-Access functions allowed.	
DMINPT.	3-27
Select dynamic menu item.	
DMINZ	3-28
Initialize the dynamic menu buffer for one-line menus.	

DMINZ5	3-28
Initialize the dynamic menu buffer for multi-line menus.	
DMSHOW	3-28
Show a dynamic menu and set displaying a one or more line group containing a given item number.	
DPIXEL	3-28
To display a pixel.	
DTEXT	3-29
Display text in the interaction area.	
GSCLR	3-29
GET/SET no clear frame status ON or OFF.	
GUPDAT	3-29
Read user input during segment execution.	
MENU	3-30
Display a menu in the interaction area.	
MODCOM	3-30
Flip mode of complement of modal and redisplay the modal.	
MODDIM	3-30
Get dimensions of modal table with maximum item size.	
MODFX	3-31
Define a new numeric (float) modal parameter in the interaction area to be displayed/changed, independent of the unit of measure of the part.	
MODFX2	3-31
Define a new numeric modal parameter to be displayed/changed in interaction area.	
MODHIX	3-32
Return length value of the MODALS AREA.	
MODI	3-32
Define a new numeric (integer) modal parameter to be displayed/changed in the interaction area.	
MODI2	3-32
Define a new numeric modal parameter to be displayed/changed in interaction area.	
MODIF1	3-33
Unlimited modal modification interaction with immediate return.	
MODIFY	3-33
Wait for the operator to change modal values in the interaction area, terminated by an extra <CR>.	

MODINT	3-33
Initialize modal parameters interaction buffer.	
MODINZ	3-34
Initialize (clear) the interaction area by deleting any previous contents.	
MODISP	3-34
Display or redisplay modal parameters from modals buffer.	
MODM	3-34
Set the next modal parameter to be displayed in the interaction area, to be a list of items. The items in the list will be pulled down every time the appropriate modal is picked.	
MODM2	3-35
Set the next modal parameter to be displayed in the interaction area, to be a list of items. The items in the list will be pulled down every time the appropriate modal is picked.	
MODMP2	3-35
Set the next modal parameter to be displayed in the interaction area, to be a list of items. The items in the list will be pulled down every time the appropriate modal is picked. When there are two items they are both opened.	
MODMPL	3-36
Set the next modal parameter to be displayed in the interaction area, to be a list of items. The items in the list will be pulled down every time the appropriate modal is picked. When there are two items they are both opened.	
MODR	3-36
Define a new numeric (float) modal parameter (in the unit of measure of the part) to be displayed/changed in the interaction area.	
MODR2	3-37
Define a new numeric modal parameter (in the unit of measure of the part) to be displayed/changed in interaction area.	
OUTCB	3-37
To display a character string on the screen.	
OUTCGB	3-38
To display a character string on the screen.	
PROMPT	3-38
Display text in the prompt area.	
REPLY	3-38
Accept YES/NO to the question PROMPT in the prompt area.	

RTEXT	3-39
Read characters from the keyboard (up to a maximum of MAXMUM), echo them in the interaction area and store them in the character string NAME.	
SETGC	3-39
Set the start position on the screen for an output string.	
SETIR	3-39
Set an immediate return for PICK functions.	
SGDCLR	3-40
Set the active color for the pixel to be displayed.	
TOPOPT	3-40
Create a pulldown menu next to the prompt area.	
UHLIGHT	3-40
Highlight instance (display in attention mode).	
UMDCLR	3-40
Get the number of the default/selected color from the pull down menu.	
UMODS	3-41
Define a new modal parameter string to be displayed/changed in the interaction area.	
USRERR	3-41
Sound a bell and display a text message.	
USRMES	3-42
Display a message on the screen without sounding the bell.	
USRMSG	3-42
Display a message on the screen <i>and</i> the window pad without sounding the bell.	
LEVELS	
CDK_LEVELS_LIST	3-43
Get the list of names and number of levels.	
CDK_LEV_GSNAME	3-43
Get/Set the name of a level by its number in the levels table.	
CLRLVB	3-44
Clear bit corresponding to LEVNUM in LVMASK.	
DSPLEV	3-44
Display/erase all entities of a specified level.	
GETLVB	3-44
Get bit corresponding to LEVNUM in LVMASK.	

GLLVD1.	3-45
Define a new level but do not set it as active.	
GLLVDF	3-45
Define a new level and set as active.	
GLLVSA	3-45
Activate a pre-defined level.	
LEVSET.	3-46
Turn the level On/Off.	
PUTLVB	3-46
Put bit corresponding to LEVNUM in LVMASK.	
SETLVB.	3-46
Set bit corresponding to LEVNUM in LVMASK.	
UDBLEV	3-47
Looks in the database for entities in the given level.	
UGTALE	3-47
Get name and number of the active level.	
VIEWLM	3-47
Given a view ID, get/set its level mask.	
LINATT	
ENCHA1	3-48
Change line attribute(s) of an entity.	
ENCHAT	3-49
Assign the specified line attribute or the level of the entity IDFROM to the entities IDTO.	
GSMSLA	3-49
Get/set level mask, active level color, pen, line font.	
PKATRB	3-50
To pack/unpack the active level, font, pen and color into/from word ATTRIB.	
STACLA	3-50
Set active line attributes.	
MANAGEMENT	
ADD_PATH_CHG.	3-51
Add old and new pathnames to the static array of ChangeExternalReference module. This is an External User routine.	
CDK_DSPTOL.	3-52
Set the display tolerance of curves, surfaces and planar faces.	

DBCLEA	3-52
Remove all temporary entities from the screen, and delete them from the database.	
DBFGET	3-52
Get the status of DBFLAG to confirm which method of creating permanent entities is being used.	
DBFLOF	3-52
Set the DBFLON flag to off, making all subsequently created entities permanent.	
DBFLON	3-53
Set the DBFLON flag to on, making all subsequently created entities temporary unless they are specified as permanent by an explicit call to DEFADD.	
DEFADD	3-53
Write down the list of permanent entities from the first position, or add given IDs to the list of permanent entities.	
DLDEDO	3-53
Delete an entity from the database. The only entities that may be deleted are those created when DISPOF is active in the current running of the user program.	
DLDELE	3-54
Delete an entity from the database. Only entities created during the run of the current procedure can be deleted.	
DLDELE1	3-54
Delete an entity from the database. Only entities created during the run of the current procedure can be deleted.	
Advantages : The ID of deleted entities can be reused.	
Disadvantages : Does not inform the association list, so do not use associated entities.	
DLDISP	3-54
To display an entity.	
DLDISPW	3-55
Display entity ID in mode MODE in ALL view ports.	
DLFORC	3-55
Delete an entity from the data base; only pickable entities can be deleted.	
DOCHPN	3-55
Change pathnames. This is an External User routine.	
DOCHPN1	3-55
Change wireframe pathnames. This is an External User routine.	

DOCHPN2	3-56
Change solid pathnames. This is an External User routine.	
DSPGET	3-56
Confirm the status of the DISPON/DISPOF status flag.	
IMMAWA	3-56
Set the AUTO-WINDOW for all the windows.	
INIT_PATH_CHG	3-56
Initialize no_of_changes (number of external reference changes). This is a static parameter used from external user programs. This is an External User routine.	
REDRAW	3-57
Repaint the display and rebuild the display file (optionally).	
SEGABSET	3-57
May be called from a user program to kill it upon exit.	
SETCR	3-57
Set “on” for output to the alphanumeric terminal or window from the interactive system.	
SETNCR	3-57
Set “off” to return to the interactive system from the alphanumeric terminal or window.	
TPLCLR.	3-58
Clear the list of temporary entities. Subsequently, all entities become permanent.	
UBLANK	3-58
Blank / unblank the entity.	
UGTNMERF	3-58
Get the number of external references in the current .pfm file.	
UMDGET	3-58
Retrieve the values of modal parameters.	
UMDPUT	3-59
Store the values of modal parameters.	
MATHEMATICS	
CVI2T	3-60
Convert an integer NUMBER (± 2147483647) to text.	
CVR2CV	3-60
Convert a real number to an array of characters.	

CVR2TS	3-61
Convert a real number to a character string (very large or small numbers will be returned in E format).	
CVT2I	3-61
Convert a character variable to its corresponding integer.	
CVT2R	3-61
Convert a character string to the corresponding real number.	
DEGRD	3-62
Convert radians to degrees.	
DEGREE	3-62
Convert radians to degrees.	
RADD	3-62
Convert degrees to radians.	
RADIAN.	3-63
Convert degrees to radians.	
CDK_POINT_EQUAL.	3-64
Check if two 3D points are equal (within tolerance).	
EQMAT	3-64
Check whether two 3D matrices are equal.	
EQMATD	3-65
Check whether two matrices are equal.	
EQPT2	3-65
Check whether two 2D points are equal.	
EQPT3	3-65
Check whether two 3D points are equal.	
EQPTD	3-66
Check whether two points are equal.	
EQPTDD	3-66
Check whether two points are equal.	
EQV0DD	3-67
Check whether A is equal to 0.	
EQVAL	3-67
Check whether A and B are equal.	
EQVAL0	3-67
Check whether A is equal to 0.	

EQVALD	3-68
Check whether A and B are equal.	
EQVL0D	3-68
Check whether A is equal to 0.	
EQVLDD	3-69
Check whether A and B are equal.	
M3CROS	3-69
Multiply M1 by M2.	
M3CROT	3-69
Multiply M1 by the transpose of M2.	
M3CRSD	3-70
Multiply M1 by M2.	
M3CRTD	3-70
Multiply M1 by the transpose of M2.	
M3DET	3-70
Find the determinant of a 3x3 matrix M.	
M3DETD	3-71
Find the determinant of a 3x3 matrix M.	
M3INV.	3-71
Find the inverse matrix of a 3x3 matrix.	
M3INVD.	3-71
Find the inverse matrix of a 3x3 matrix.	
M3LNED	3-72
Find the solution to 3 linear equations:	
M3LNEQ	3-72
Find the solution to 3 linear equations:	
M3VEC	3-73
Multiply a 3D-vector by a 3x3 matrix.	
M3VECD	3-73
Multiply a 3D-vector by a 3x3 matrix.	
PIFACT	3-73
Evaluate factor of π .	
V2ADD	3-74
Add a 2D scaled vector to another 2D vector: $V3 = V1 +$ (SC*V2)	

V2CROS	3-74
Calculate the vector product of two 2D vectors: $V1 * V2$.	
V2CRSD	3-74
Calculate the vector product of two vectors: $V1 * V2$.	
V2DOT	3-75
Calculate the dot product of two 2D vectors: $V1 \cdot V2$.	
V2LNED	3-75
Find the solution to 2 linear equations:	
V2LNEQ	3-76
Find the solution to 2 linear equations:	
V2SCAL.	3-76
Scale a 2D vector by a factor SC.	
V2SIZD	3-76
Find the length of the 2D vector V1.	
V2SIZE	3-77
Find the length of the 2D vector V1.	
V2SUB	3-77
Subtract a 2D vector from another 2D vector: $V3 = V1 - V2$	
V2UNIT	3-77
Change a 2D vector into a unit vector.	
V2ZERD	3-78
Check whether a 2D vector is a zero vector.	
V2ZERO	3-78
Check whether a 2D vector is a zero vector.	
V3ADD	3-78
Add a 3D scaled vector to another 3D vector: $V3 = V1 + (SC*V2)$	
V3CROS	3-79
Calculate the vector product of two 3D vectors: $V3 = V1 * V2$.	
V3CRSD	3-79
Calculate the vector product of two 3D vectors: $V3 = V1 * V2$.	
V3DOT	3-79
Calculate the dot product of two 3D vectors: $V1 \cdot V2$.	
V3SCAL.	3-80
Scale a 3D vector by a factor SC.	

V3SIZD	3-80
Calculate the length of 3D vector V1.	
V3SIZE	3-80
Calculate the length of 3D vector V1.	
V3SUB	3-81
Subtract a 3D vector from another 3D vector: $V3 = V1 - V2$	
V3UNIT	3-81
Change a 3D vector into a unit vector.	
V3ZERD	3-81
Check whether 3D vector is a zero vector.	
V3ZERO	3-82
Check whether 3D vector is a zero vector.	
VASDDD	3-82
Add a scaled vector to another vector.	
VDOTD	3-83
Calculate the dot product of two vectors: $V1 \bullet V2$.	
VECSCL	3-83
Scale a vector by a factor.	
VSUBD	3-84
Subtract a vector from another vector: $V3 = v1 - v2$	
VUNITD	3-84
Change a vector into a unit vector.	
CDK_WRKPLN_3PNT_ROT	3-85
Set a new work plane, defined as passing through 3 points. A work plane will not be created ONLY if there is a work plane with the same rotation that the 3 points define.	
CKPT2D	3-85
Check whether points are lying on a plane within a given tolerance.	
DISAXP	3-86
Find the distance between an axis and a point.	
DIST2	3-86
Find the distance between two 2D points.	
DIST3	3-86
Find the distance between two 3D points.	
DISTD	3-87
Find the distance between two points.	

IND2C	3-87
Calculate the intersection points of two circles.	
INPOLD	3-88
Find whether a point lies within a closed polygon.	
INT2C	3-88
Calculate the intersection points of two circles.	
LSQRPD	3-89
Find the least square plane.	
PLCOF	3-89
Define plane coefficients from three points.	
PLCOFD	3-89
Define plane coefficients from three points.	
V2ANGL	3-90
Find the angle between two 2D vectors V1, V2.	
V2PRP	3-90
Calculate a perpendicular vector, to the right or left of a given vector.	
V3ANGL	3-90
Find the angle between two 3D vectors V1, V2.	
V3LIN	3-91
Determine whether three 3D points lie on one line.	
WVCHK	3-91
Check if an entity is lying on the work plane.	
PICK	
CDK_CURIN	3-92
Receive input from the mouse or keyboard.	
CDK_MENU_POPUP	3-93
Call the popup menu from a user application.	
CDK_MOUSE_INPUT	3-93
Receive input from the mouse.	
CDK_PICK_WINDOW	3-94
Pick a window.	
ENA3DC	3-94
Enable picking of 3D curves.	
ENAALL	3-94
Enable picking of all types of entities.	

ENAARC	3-94
Enable picking of arcs.	
ENABLE	3-95
Enable the selection of class and type lists.	
ENACRV	3-95
Enable picking of 2D curves.	
ENAINS	3-95
Enable picking of instances.	
ENALIN	3-95
Enable picking of lines.	
ENAPLF	3-95
Enable picking of planar faces.	
ENAPT	3-96
Enable picking of points.	
ENASAT	3-96
Enable picking of surfaces.	
ENAUCS	3-96
Enable picking of UCS.	
GETPD	3-97
Get the coordinates of a picked point.	
GETPT	3-98
Get the coordinates of a picked point.	
GINPTD	3-98
Get the display coordinates of the picked point.	
MPICKP	3-99
Multiple selection option.	
PICK	3-99
Listen to input devices. If a graphical input is activated, pick and push the ID and coordinates of the picked entity into the SELECTION LIST.	
SELARR	3-99
Chose one of two opposite displayed directions.	
SETMF	3-100
Enable picking within an instance.	
SLCLR	3-100
Clear the selection list.	

SLLEN	3-100
Return the number of entities in the selection list.	
SLPOP	3-100
Pop an entry from the selection list.	
SLPOP3	3-101
Pop an entry from the selection list and return the XYZ coordinates of the picked point.	
SLPSHI	3-101
Push an entry into the selection list with dummy attention point.	
SLPSH3.	3-101
Push an entry into the selection list.	
SLRJCT.	3-101
Remove an entry from the selection list.	
SLTOP3.	3-102
Get the top entry from the selection list and return the XYZ coordinates of the picked point.	
UGETPNT	3-102
Get coordinates of a picked point.	
UMPICK	3-102
Pick entities and store their IDs in the array IDS.	
UPICK.	3-103
Pick a single entity.	
UPICKP.	3-103
Pick a curve, and create a point at the picked position.	
PLOT	
CDK_PLOT_BOX	3-104
Immediate plot.	
CDK_PLOT_DEVICE_LIST	3-104
Get the plot device list.	
CDK_PLOT_GET_DEVICE_NAME.	3-105
Get the name of the device.	
CDK_PLOT_SET_COLORMAP	3-105
Set color map.	
CDK_PLOT_SET_OFFSET	3-105
Set the plotting offsets.	
CDK_PLOT_SET_ROTATE	3-106
Set plotting rotate flag.	

CDK_PLOT_SET_SPEED	3-106
Set the plotting speed.	
REPORTS	
CDK_QE_REPORT_CREATE.	3-107
Create an electrode report.	
STATUS	
AXI_TYPE_OPEN.	3-108
Get the type of the active part.	
CDK_EXTRACT_SUB_ASSY	3-108
Extract as EXTRACT >> EXTRACT SUB ASSEMBLY.	
CDK_PRG_INFO	3-109
Return a string containing the version number, build number, etc.	
GTLANG	3-109
Return the translated version.	
GTMACH	3-109
Return the type of machine.	
LASTID	3-110
Get the last used ID number.	
OB2DMD	3-110
Obtain the work mode - 2D/3D.	
OBACAT	3-110
Obtain the active font, pen and color.	
OBACME	3-110
Get the part file measurement unit.	
OBACRT	3-111
Get the pathname of the Cimatron ^{it} root directory.	
U2DMOD	3-111
Check if system is in 2D mode.	
UAXI_INDEX_DATA	3-111
Get the data of assembly tree components.	
UAXI_NUM_ASM_COMPS	3-111
Get the number of assembled components on all levels.	
UGTAPL	3-112
Get the current application number.	
UGTDTM	3-112
Get the current date and time.	

UGTERF	3-112
Get the list of external references in the current PFM.	
UGTNID.	3-113
Get the node ID.	
UGTPFN	3-113
Get the name of the current part file.	
UGTPNM	3-114
Get full path name	
UGTSNM	3-114
Get the name of a system file/directory.	
UGTVNM	3-115
Get the view number of the active view.	
UGVNAM	3-115
Get view name and type.	
USAVE	3-116
Save the current file with a given name.	
VPDRWN	3-116
Set the ERASE flag = 0 for a given view port.	
VPERSD	3-116
Get the ERASE flag for a given view port.	
VPGSDX	3-117
Get/Set the display transformation data from the view port table.	
VPGVPN	3-117
Get the current view port number.	
VPLDIS	3-117
Get a list of all displayed windows.	
STRING	
CHCOMP	3-118
Compare two characters with the option of ignoring lower/upper case.	
CONVLO	3-118
Convert a string to lower case.	
CONVUP	3-119
Convert a string to upper case.	
LENACT	3-119
Get the actual length of a string (without trailing blanks).	

LENINT	3-119
Find the length of an integer number.	
STCOMP	3-120
Compare two strings with the option of ignoring lower/upper case. The values of FROM1 and TO1 determine how many characters from STRNG2 will be included in the comparison.	

Chapter 4 Modeling Routines

ANALYZE

GFSPIS	4-2
Add an island to a closed contour.	
GFSPOC	4-2
Initialize contour properties and define an outer contour. The contour must be “well-defined” i.e. the curves defining the contour must meet at their endpoints, and no overlapping of lines is permitted.	
GFSPUT	4-3
Calculate and output various properties of the contour.	

CIRCLE

AR2CRV	4-4
Create an arc/circle entity on the work plane, tangent to 2 curves. The entity may also trim the curves to produce a fillet.	
AR3PTD	4-4
To create an arc/circle defined by 3 points.	
AR3PTS	4-5
Create an arc/circle entity on the work plane, defined by 3 points.	
ARCNRA	4-5
Create an arc/circle entity on the work plane, defined by the ID of its center and its radius RADIUS. If an arc is defined, ANGLE is its angle relative to the X axis in degrees, and DELTAN is the angle relative to ANGLE.	
ARCNRAD	4-6
Create an arc/circle on the work plane, defined by it's center point and radius.	
AREX2C	4-6
Create an arc/circle entity on the work plane, tangent to 2 curves or their extensions. It may also trim the curves to produce a fillet.	
BEST_CRC.	4-7
Find the “best” circle that can be created by a given points sequence.	

CDK_ARC_3CRV	4-7
Create arc/circle tangent to three curves.	
CDK_ARC_2PNT	4-8
Create arc/circle by two points and radius.	
CDK_ARC_3PNT	4-8
To create an arc/circle defined by coordinates of 3 points.	
CDK_ARC_CENT_RAD	4-9
To create a arc/circle on the work plane, defined by it's center point and radius.	
CDK_ARC_PNT2CRV	4-9
Create all circles tangent to two given curves and point.	
CONIC	
CDK_CONIC_3PNT.	4-10
Create conic curve by 3 points.	
CDK_CONIC_WP	4-11
Create conic curve parallel to Work Plane.	
CDK_ELLIPSE.	4-11
Create an ellipse entity parallel to the work plane.	
PARABO	4-12
Create an entity that is a parabola, defined by its two endpoints, ID1 and ID2, and the slopes at these points, SL1 and SL2.	
UCREL P	4-12
Create an ellipse on the work plane, defined by its center ID, and its major & minor radii A and B. If an arc is defined, ANGLE gives its angle (relative to the X work axis).	
CONTOUR	
CDK_BALCON.	4-13
Get all contours which contain a given entity.	
CDK_COMCRV_GET.	4-13
Obtain the components of the component curve.	
CDK_CONTOUR_MULTY	4-14
Create a 2D multi-contour buffer from a list of 2D curves.	
UCOMCR.	4-14
Given a contour buffer, create a composite curve.	
UCOMSP.	4-15
Approximate a contour by a Bezier spline.	

UDFDC	4-15
Create a CLOSED contour buffer interactively.	
UDFDO	4-16
Create an OPEN contour buffer interactively.	
USMKCB	4-17
Create a contour buffer from a list of curves. The contour must be "well-defined" i.e. the curves defining the contour must meet at their endpoints and no overlapping of lines is permitted.	
CORNER	
GFCHEMF	4-18
Create a chamfer as it is done by the function CORNER in the interactive system.	
GFCORN	4-19
Trim curves at their intersection point as it is done by the function CORNER in the interactive system.	
GFFILT	4-20
Create a fillet corner.	
EXPLODE	
GREXPL	4-21
Explode an instance.	
UEXPLD	4-21
Explode an instance by levels.	
GROUP	
CDK_UMDDEL	4-22
Delete a modal group.	
GRCLCR	4-22
Create a master composed of entities, the IDs of which are given in the array IDS.	
GRCLOS	4-23
Stop designating entities which will be grouped to form a master in the current session of the interactive USER function.	
GRCOPN	4-23
Start designating entities which will be grouped to form a master in the current session of the interactive USER function.	
GRDLUN	4-24
Delete an unnamed master.	
GRIDEM	4-24
Find the ID of an external master.	

GRIDES.	4-25
Find the ID of an external sub-view.	
GRIDIM.	4-25
Find the ID of an internal master.	
GRIDIS	4-26
Find the ID of an internal sub-view.	
GRIDSA.	4-26
Find the ID of a sub-assembly.	
UMADEL	4-27
Delete a named master.	
WSNMX1.	4-27
Export the master from the current PFM file to another file by its full path name.	
LINE	
CDK_LINE_OFFSET	4-28
To create an offset line.	
LN2ENT.	4-28
Create a line entity defined by two point entities.	
LN2ENTD.	4-29
To create a line defined by two points.	
LN2PNT.	4-29
Create a line entity defined by two points.	
LN2PNTD.	4-30
Create a line defined by two points.	
LNCVCV	4-30
Create a line using the 2 CURVES option, as in the interactive system.	
LNIPVE	4-31
Create a line entity defined by the ID of a point, and a vector.	
LNIPVED	4-31
Create a line defined by a point and a vector.	
LNLINC	4-32
Create a line using LN CURVE option as in the interactive system.	
LNOFFS	4-32
Create an offset line.	
LNPOEN	4-33
Create a sequence of lines as an open/closed polygon.	

LNPOEND	4-33
Create a sequence of lines as a polygon.	
LNPTCV	4-34
Create a line using the PT-CURVE option as in the interactive system.	
LNPTVE	4-34
Create a line entity defined by a point and a vector.	
LNPTVED	4-35
Create a line defined by a point and a vector.	
MOVE	
CDK_OFFSET	4-36
Create an entity by offsetting another entity.	
CDK_OFFSET_CONT	4-37
Offset of the contour.	
CDK_SCALE	4-38
Create a scaled entity.	
ENINSTD	4-39
Create a transformed copy of an entity.	
ENMILID	4-39
Create a copy of an entity by mirroring about the line.	
ENMIPLD	4-39
Create a copy of an entity by mirroring about the plane.	
ENMIPOD	4-40
Create a copy of an entity by mirroring through the point.	
ENMUIND	4-40
To create multiply copies of an entity.	
ENOFFS	4-41
Create an offset entity at a distance, OFFSET.	
ENROAXD	4-41
Create a copy of an entity by rotation it about the axis.	
ENSCA1D	4-42
To copy an entity and transform it by a given scale factor, relative to a fixed point. The point remains unscaled.	
ENSHIFD	4-42
Create a copy of an entity by shifting the vector.	
UDRAG	4-42
Drag an entity interactively.	

PLACE

GRIN3P. 4-43

Create an instance of a previously defined master, relating the 3 reference points of the master to 3 other points (ID1, ID2, and ID3) and scaling by a factor.

GRINRO 4-44

Create an instance of a previously defined master, by shifting its origin to ID1, rotating it by an angle ANGLE about the Z axis, mirroring it about the X or Y axis, and scaling it.

GRINST. 4-44

Create an instance of a previously defined master.

PLEXGR 4-45

Create an instance of an external master entity.

PLANAR FACE

CDK_PLF_BOXES 4-46

Get the boxes for the given planar face.

CDK_PLF_CONCRV 4-46

Get the number of curves in a single contour.

CDK_PLF_CREATE 4-47

Create the planar face entity.

CDK_PLF_EXPLODE. 4-47

Explode the planar face.

CDK_PLF_GETCRV 4-47

Get the ID of a curve from the planar face definition.

CDK_PLF_TOTCRV 4-48

Get the number of contours and total number of curves.

UPLFTS. 4-48

Convert a planar face to a trimmed surface.

UTPFWP 4-49

Trim a planar face by the WORK plane.

UTRMPF 4-49

Trim a planar face by contours.

POINT

CDK_PNT_BSPLINE 4-50

Create points of a B-Spline.

CDK_PNT_COOR. 4-50

Create a point entity defined by its coordinates.

PNTEX3	4-51
Create a point entity defined as the intersection of two entities ID1 and ID2 or their extensions closest to two given points IDP1 and IDP2.	
PNTIN3	4-51
Create a point entity defined as the intersection of two entities ID1 and ID2 closest to two given points IDP1 and IDP2.	
PTCDIS	4-52
Create point entities as in POINT >> MULTI-POINTS >> BY DISTANCE. See Chapter 1, of the Modeling Manual .	
PTCENT	4-52
Create a point entity defined as the center point of an arc/circle.	
PTCOMO	4-53
Create a point defined by its coordinates in the model system.	
PTCOWO	4-53
Create a point defined by its coordinates in the work system.	
PTEND	4-53
Create a point entity defined as the endpoint of a curve.	
PTEXIN	4-54
Create a point entity defined as the intersection of two entities ID1 and ID2, or their extensions. Only those points defining the intersection of coplanar entities are created.	
PTGET	4-54
Create a point via interaction with the user.	
PTINCR	4-55
Find all intersection points of two coplanar curves.	
PTINTR	4-55
Create a point entity defined as the intersection of two entities, ID1 and ID2. Only those points defining the intersection of coplanar entities are created.	
PTM2PT	4-56
Create a point at a specified distance, DIST, from the midpoint of a diagonal or one of the edges of a rectangle. The rectangle is defined by the endpoints of one diagonal.	
PTMIDD	4-57
Create a point entity defined as the midpoint of a curve.	
UPNTCENT	4-57
Create a point entity defined as the center of the curve.	
UPNTCM	4-57
Create a point entity defined by its coordinates in the Model System.	

UPNTCW	4-58
Create a point entity defined by its coordinates in the WORK System.	
UPNTEND	4-58
Create a point entity defined as the endpoint of the curve.	
UPNTGET	4-59
Create a point entity interactively.	
UPNTMID.	4-59
To create a point entity defined as a middle point of the curve.	
PROJECT	
CDK_PROJ_CNTR_PLANE	4-60
Project a curve onto the active work plane.	
CDK_PROJ_CNTR_SRF.	4-61
Project a contour onto a surface.	
CDK_PROJ_CNTR_SRF_FREE	4-61
Free memory that was allocated in CDK_PROJ_CNTR_SRF.	
CDK_PROJ_CURVE_PLANE	4-62
Project a curve onto the active work plane.	
CDK_PROJ_CURVE_SRF.	4-62
Project a curve onto a surface.	
CDK_PROJ_CURVE_SRF_FREE	4-63
Free memory that was allocated in CDK_PROJ_CURVE_SRF.	
CDK_PROJ_POINT_PLANE	4-63
Project a point onto a plane.	
CDK_PROJ_POINT_SRF	4-64
Project a point onto a surface.	
CDK_PROJ_POINT_SRF_FREE	4-64
Free memory that was allocated in CDK_PROJ_POINT_SRF.	
CDK_PROJ_WPDIR	4-65
Create Projected Entities onto the work plane according to a given direction.	
CDK_PROJ_WPNORM	4-65
Create Projected Entities onto the work plane according to the Normal direction.	
CDK_PROJ_XYDIR.	4-66
Create Projected Entities onto the XY plane of the active UCS according to a given direction.	

CDK_PROJ_XYNORM	4-66
Create Projected Entities onto the XY plane of the active UCS according to the Normal direction.	
PRCRSR	4-67
Create the projection of a curve on a surface.	
PRJDR1	4-67
Create entities projected onto the XY plane of the active UCS, in the direction DIR.	
PRJPD1.	4-68
Create entities projected perpendicularly onto the XY plane of the active UCS.	
PROJDR	4-69
Create entities projected onto the current work plane, in the direction DIR.	
PROJPD	4-70
Create entities projected perpendicularly onto the current work plane.	
SPLINE	
CDK_CRV2POLY	4-71
Create a polygon by stepping along a curve within tolerance.	
CDK_CRV_APPROX	4-72
Approximation curves by continuous arcs and lines.	
CDK_CRV_DST_CRV	4-73
Find the closest points between two curves.	
CDK_CRV_TRMLOOP	4-73
Prepare given planar curve to chain of curves without loops.	
CDK_CURVE_FAIR.	4-74
Execute fairing on a curve.	
CDK_SPLINE_ADJOIN.	4-74
Adjoin curves.	
CDK_SPLINE_CP.	4-75
Create a B spline by defining its control points.	
CDK_SPLINE_CREATE	4-76
Create a NURBS spline entity (with knots).	
CDK_SPLINE_CUBIC	4-77
Create a cubic spline.	
CDK_SPLINE_FP	4-78
Create a B spline by defining its fairing points.	

CDK_SPLINE_NURBS_DATA	4-78
Get control points and knots of a a B-spline.	
CDK_SPLINE_NURBS_SIZE	4-79
Get parameters (dim, ncp, ord) of a NURBS spline, in order to know how much to allocate for knots and control points. Use before function CDK_SPLINE_NURBS_DATA.	
CDK_SPLINE_TP	4-79
Create a B spline by defining its through points.	
CLOSCV	4-80
Check whether a curve is closed and/or periodic.	
GTCRPT	4-80
For a given point on the curve and a distance (by curve) from it to another point on the curve, get the coordinates and derivatives of the second point.	
GTCRPTD	4-81
Get coordinates and derivatives of the point on the curve, according to the distance (by curve) to the given curve's point.	
SPLNCR	4-81
Create a spline.	
UBSPLN	4-82
Create a B-spline entity.	
UCRBSP	4-83
Create a Bezier spline.	
UCRVLN	4-84
Calculate the length of a curve.	
UEVCRV	4-84
Given a curve parameter, get the coordinates of a point on it and the derivatives of this point.	
UGPARM	4-85
Given the ID of the curve and coordinates of the point, retrieve the parameter on the curve closest to indication.	
UGSSRF	4-85
Get/Set refine factor of the spline.	
ULMCRD	4-85
Get curve limits from data base.	
USPLNP	4-86
Get number of control points	

SURFACE

CDK_BLEND_REGION.	4-87
Create a trimmed surface given its outer loop as a 3D contour.	
CDK_BLEND_SECTIONS	4-88
Create a blend section surface.	
CDK_PLN_INTERSEC	4-88
Calculate the intersection points of the plane and curve.	
CDK_SLA_CREATE	4-89
Create an SLA surface.	
CDK_SLA_EXIT	4-89
Close the CDK_SLA package and clear allocated memory.	
CDK_SLA_FACE_ADD.	4-89
Create an SLA face.	
CDK_SLA_INI	4-89
Activate the CDK_SLA package.	
CDK_SLA_NODE_ADD	4-89
Create the SLA node.	
CDK_SRF_BOX.	4-90
Calculate a surface enclosing box in the MODEL coordinate system.	
CDK_SRF_CRPIP.	4-90
Create a NURBS surface as in the following DRIVE options of the Modeling application:	
DRIVE >> SPINE >> ONE SECTION	
DRIVE >> SPINE >> TWO SECTIONS	
DRIVE >> SPINE & PLANE	
CDK_SRF_DRPRL	4-91
Create a drive surface by moving section contour along path defined by a drive contour so that it is always parallel to itself.	
CDK_SRF_FAIR.	4-91
Execution of fairing per surface.	
CDK_SRF_INTERSEC	4-92
Calculate the intersection points surface & curve.	
CDK_SRF_INTERSECTION.	4-92
Calculating the GLOBAL OR LOCAL intersection between two surfaces.	
CDK_SRF_MODIFY_LIM	4-93
Modify surface limits.	
CDK_SRF_MULTY_DRIVE	4-93
Create a multiple drive surface.	

CDK_SRF_NORMAL	4-94
Get the normal of a surface at an indicated point.	
CDK_SRF_REFINE	4-94
Get/Set the refine factor of a surface.	
CDK_SRF_RULCS	4-95
Create a draft-angle ruled surface between a given contour and a surface.	
CDK_SRF_SECWPL	4-95
Create the splines which are intersections between a surface and a work plane.	
CDK_SRF_TRM_CON	4-96
Trim a surface/trimmed surface by a contour lying on it.	
CDK_SRF_TRM_PAR	4-97
Trim a surface/trimmed surface by a parameter.	
CDK_SRF_TRM_PCON	4-98
Trim a surface/trimmed surface by a projected contour.	
CDK_SRF_TRM_SRF	4-99
Trim a surface/trimmed surface by a surface.	
CDK_SRF_TRM_SRF1	4-100
Trim a base surface by multi surfaces.	
CDK_SRF_TRM_WPL	4-101
Trim a surface/trimmed surface by the work plane.	
CDK_SURFACE_CREATE	4-102
Create a NURBS surface in the DataBase.	
CDK_SURFACE_NURBS_DATA	4-103
Get control points and knots of a NURBS surface.	
CDK_SURFACE_NURBS_SIZE	4-103
Get ord_u, ord_v, ncp_u, ncp_v of a NURBS surface in order to know how much to allocate for knots and control points.	
Use before function CDK_SURFACE_NURBS_DATA.	
CPTPCS	4-104
Calculate the closest point to an iso-parametric curve surface.	
DBDSBX	4-104
Get/Set the enclosing box of a surface.	
DRPARL	4-105
Create a drive surface by moving a section curve along a path defined by a drive curve so that it is always parallel to itself.	

PRDRSR	4-105
Project a point on a surface at a given direction.	
PRNRSR	4-106
Project points on a surface in the normal direction.	
REVOLS	4-106
Create a surface of revolution by rotating a specified curve around an axis.	
RULEDS	4-107
Create a ruled surface by joining the corresponding points on two curves by a set of straight line segments.	
SRFPTD	4-108
Find the picked point indication parameters (uc, vc) on a surface/trimmed surface, immediately after the picking.	
TRSGID.	4-109
Get the surface ID on which the Trimmed surface is defined.	
TRSGSZ	4-109
Get the number of polygons and the UV polygon size of a trimmed surface.	
TSD_BOX	4-109
Get/Set the parametric/geometric enclosing box.	
UBSSPW	4-110
Get/Set a list of control points & weights (One section).	
UBSURF	4-111
Create a Nurbs surface.	
UBZSRF	4-111
Create a Bezier Mesh surface.	
UCRBCM	4-112
Create a bi-cubic mesh surface (free slopes).	
UCR2SP	4-113
Create a Bezier Mesh surface as in the DRIVE >> TWO SPINES command of the Modeling application.	
UCRPIP.	4-114
Create a Bezier Mesh surface as in the following DRIVE options of the Modeling application:	
DRIVE >> SPINE >> ONE SECTION	
DRIVE >> SPINE >> TWO SECTIONS	
DRIVE >> SPINE & PLANE	

UEVSRD	4-115
Given surface parameters get coordinates of point on it and derivations in this point.	
UFILLC	4-116
Create a fillet surface between two surfaces with a constant radius.	
UFILLV	4-117
Create a fillet surface between two surfaces with variable radius.	
UGNRBS	4-117
Get properties of NURBS.	
UGPARS	4-118
Given the ID of a surface and parameter, create a parametric spline.	
UGSBZP	4-118
Get/Set control points of a Bezier Mesh surface.	
UGSPRM	4-119
Given the ID of a surface and coordinates of a point, retrieve the parameters of the surface closest to the indicated point.	
UGSSDC	4-119
Get/Set a number of displayed curves to display a surface.	
UGTRPO	4-120
Given the ID of a trimmed surface, retrieve the UV polygon's data.	
ULMSRF	4-120
Get surface limits from data base.	
UNPBZS	4-120
Get the number of patches defining a bezier surface.	
UREVOLS	4-121
Create a surface of revolution by rotating a curve around an axis.	
USRFBD	4-122
Get the boundaries of a surface.	
USRFIN	4-122
Create splines along the intersection of two given surfaces.	
USRFPT	4-123
Find parameters of a surface/trimmed surface defining the point on the surface closest to the given point.	
USRNOR	4-123
Get the coordinates of a point which lies at a given distance from a point, that is normal to surface direction, on a given surface.	

USRSCL	4-124
Scale a general surface.	
USRSEC	4-124
Create splines along intersection of two given surfaces.	
UTRSCO	4-125
Trim a surface/trimmed surface by a contour lying on it.	
UTRSPA	4-125
Trim a surface/trimmed surface by a parameter.	
UTRSPC	4-126
Trim a surface/trimmed surface by a projected contour.	
UTRSPL	4-126
Trim a surface/trimmed surface by the work plane.	
UTRSSR	4-127
Trim a surface/trimmed surface by a surface.	
UVRBSR	4-127
Get/set control points of a Bezier Mesh surface.	
SWEEP	
CDK_SWEEP_ANGLE	4-128
To sweep entities angular.	
CDK_SWEEP_DELTA	4-128
To sweep entities linearly.	
ENSWLR	4-129
Sweep entities linearly.	
ENSWRT	4-129
Sweep entities angularly.	
TRANSFORMATION	
GTPLMA	4-130
Obtain a transformation matrix which is defined interactively either by indicating 3 points, or, by specifying a position point and a rotation angle (around the Z axis).	
MADM2W.	4-130
Transform a point from the MODEL coordinate system to the current work coordinate system.	
MADW2M.	4-131
Transform a point from the current work coordinate system to the MODEL coordinate system.	

MAPM2W.	4-131
Transform a point from the model coordinate system to the current work coordinate system.	
MAPW2M.	4-131
Transform a point from the current work coordinate system to the model coordinate system.	
TR3PTS.	4-132
Calculate a transformation matrix.	
TRCONC	4-132
Calculate the transformation matrix (TRRESU) resulting from concatenation of transformation matrices TRA and TRB. TRRESU is equivalent to having first applied TRA and then TRB.	
TRCONM.	4-132
Calculate the transformation matrix (TRRESU) resulting from concatenation of transformation matrices TRA and TRB. TRRESU is equivalent to having first applied TRA and then TRB.	
TRCONMD	4-133
Calculate the transformation resulting from concatenation of transformation matrices TransfA and TransfB. The resulting transformation matrix TransfRes is equivalent of having first applied TransfA and THEN TransfB.	
TRMILI	4-133
Calculate a transformation matrix which defines mirroring about the line.	
TRMILID	4-133
Calculate the transformation which defines mirroring about the line.	
TRMIPL.	4-134
Calculate the transformation matrix which defines mirroring about the plane, PLANE.	
TRMIPLD	4-134
Calculate the transformation which defines mirroring about the plane.	
TRMIPO	4-134
Calculate the transformation matrix which defines mirroring through the point, POINT.	
TRMIPOD	4-135
Calculate the transformation which defines mirroring about the point.	
TRROAX	4-135
Calculate the transformation matrix which defines a rotation about the axis, AXIS, through the point, POINT by an angle, ANGLE.	
TRROAXD	4-135
Calculate the transformation which defines a rotation about the axis.	

TRSCA1	4-136
Build the transformation matrix required to change the scale of an entity by a given factor, relative to a specified position point.	
TRSCA1D	4-136
Build the transformation required to change the scale of an entity by a given factor, relative to a specified position point.	
TRSHIF	4-136
Calculate the transformation which defines a shift by the vector SHIFT.	
TRSHIFD	4-137
Calculate the transformation which defines a shift.	
UTR3PNT	4-137
Calculate a transformation matrix.	
TRIM	
CDK_TRS_BYPLN	4-138
Trim a surface/trimmed surface by the work plane.	
CVDELP	4-138
Delete the part of a curve "ldolcv", which lies between the curves "ldtcv1" and "ldtcv2".	
DIVCRV.	4-139
Divide a curve by its intersection with two other curves.	
GFDIVD.	4-139
Divide a curve by 2 other curves.	
GFDVPT	4-140
Divide an open curve by a point, or a closed curve by 2 points.	
GFTRIM.	4-140
Trim curve by another curve "ldtcv".	
GFTRPL	4-141
Trim curve by work plane.	
GFTRPT	4-141
Trim a curve by a point.	
TRMCRV	4-142
Trim a curve at its intersection with another curve.	
UCS	
ACTUCS	4-143
Activate an existing UCS.	
CDK_UCS_CREATE	4-143
Create a UCS by a transformation matrix.	

CDK_UCS_CREATE_PTS.	4-144
Create a UCS by three points.	
CDK_UCS_DISPLAY.	4-144
Display UCS.	
CDK_UCS_LIST.	4-144
Create a list of UCS names.	
CDK_UCS_MAP.	4-145
Map a point from UCS to MODEL or WORK system.	
CDK_UCS_MAP_FROM.	4-145
Map a point from UCS to MODEL or WORK system.	
CDK_UCS_MAP_TO.	4-145
Map a point from MODEL or WORK system to UCS.	
CDK_UCS_TRANS.	4-146
Transform three points into a rotation matrix.	
CDK_UCS_ZAXIS.	4-146
Create a UCS from the Z axis only.	
CDK_WORKPLANE.	4-146
Get / set active work plane parameters (double precision).	
CREUCS.	4-147
Create a UCS, without making it the active coordinate system.	
CRUCS3.	4-147
Create a UCS entity as in the 3-PTS option.	
DEFPLN.	4-148
Define one of the planes of a specified UCS as the active work plane.	
DELUCS.	4-148
Delete an existing UCS. (The active UCS cannot be deleted).	
GACUCS.	4-148
Retrieve the ID of the active UCS.	
ID2NAM.	4-149
Given a UCS ID, retrieve the UCS name.	
MADU2M.	4-149
Map a point from the UCS to the MODEL system.	
MADUCS.	4-149
Map a point from the UCS to the MODEL or WORK system.	
MAPM2U.	4-150
Map a point from the Model coordinate system to a UCS.	

MAPU2M	4-150
Map a point from a specified UCS to the Model system.	
MAPU2W	4-150
Map a point from a selected UCS to the Work system.	
MAPW2U	4-151
Map a point from the Work system to a specified UCS.	
NAM2ID	4-151
Given a UCS name, retrieve the UCS ID.	
UCSMAT	4-151
Create a matrix given 3 points for UCS purposes.	
VERIFY	
CDK_VERIFY_ENTITY	4-152
Obtain the class and type of an entity.	
OBCRCL	4-152
Obtain arc/circle data.	
OBGROU	4-153
Obtain data about a master.	
OBINST	4-153
Obtain an instance data by its ID.	
OBLINE	4-154
Obtain line data.	
OBPNT	4-154
Obtain point data.	
OBTUCS	4-154
Obtain information on the UCS.	
UGTMNM	4-155
Obtain the name of a master.	
UODCIR	4-155
Obtain circle data in double.	
UODCON	4-156
Obtain conic data in double.	
UODHLX	4-156
Obtain helix data in double.	
UODLIN	4-157
Obtain line model coordinates in double.	

UODPNT	4-157
Obtain point model coordinates in double.	
UODPRJ	4-157
Obtain the projected matrix of an entity.	
UODSPL	4-158
Obtain spline data in double.	
WORKPL	
CDK_WPGETD	4-159
Obtain the coefficients of the work plane in the equation $Ax+By+Cz+D=0$. Type: Double.	
CDK_WRKPLN_3PNT	4-159
Set a new work plane, defining as passing through 3 points.	
UWP2CRV	4-159
Define a new working plane by planar curve or 2 lines.	
UWP3PTID	4-160
Set a new work plane, defining as passing through 3 points.	
UWPCOEF	4-160
Define a new working plane by 4 coefficients.	
UWPDSPL	4-160
Define a new working plane as parallel to the screen and passing through a given point.	
UWPLNPT	4-161
Set a new work plane, defining as perpendicular to a line and passing through a point.	
WORKDF	4-161
Define a work plane interactively.	
WPCRVS	4-162
Set a new work plane defined by a planar curve or two lines.	
WPGET	4-162
Obtain the coefficients of the work plane in the equation $Ax + By + Cz + D = 0$. Type: Float.	
WPLNPT	4-162
Set a new work plane, defined as perpendicular to a line and passing through a point.	

Chapter 5 Drafting Routines

ANGULAR DIMENSION

DIAN1A.	5-3
Create a dimension entity for an angle defined by a line and an axis.	
DIAN2A.	5-4
Create a dimension entity for an angle defined by two lines.	
DIAN3A.	5-5
Create a dimension entity for an angle defined by three points.	
DIANEX.	5-6
Extract all specific modals for angular dimensions.	
DIANG1.	5-7
Create a dimension entity for an angle defined by a line and an axis.	
DIANG2.	5-8
Create a dimension entity for an angle defined by two lines.	
DIANG3.	5-8
Create a dimension entity for an angle defined by three points.	
DIANPR.	5-9
Set specific modal values for dimension entities of angles. Refer to the Cimatron ^{it} Drafting Manual , for explanations of the modal parameters.	
DIARPA.	5-10
Set all specific modals for angular dimensions.	

CENTER LINES

CEN2LN	5-11
Create center lines.	

CHAMFER DIMENSION

DICHEX.	5-12
Extract all the specific modals for chamfer dimension entities.	
DICHMA	5-12
Create a chamfer dimension entity.	
DICHPA.	5-13
Set all the specific modals for chamfer dimension entities.	

DIMENSION PARAMETERS

DIEXTR.	5-14
Extract the non-geometrical data of a dimension from the database to the memory structures.	

DIGLEX.	5-15
Extract all the global modals for dimension entities of all kinds.	
DIGLMD	5-16
Accept the modal values of the interactive system set by the DRAF_PAR function as the global parameters for the Drafting functions in CimaDEK .	
DIGLPA.	5-17
Set all the global modals for dimension entities of all kinds.	
DIGLPR.	5-22
Set global parameters for the Drafting USER functions. Refer to the Cimatron ^{it} Drafting Manual , for explanations of the modal parameters.	
DIGSNA.	5-22
To get names of existing standards.	
DIMODI.	5-23
Modify the non-geometrical data of a dimension with the data stored in the memory structures (the result of previous calls to DIEXTR and DIGLPA).	
STDNID.	5-23
Given the name of a drafting standard, get its internal number in the standard table.	

GEOMETRY

Creates a point associated with the CENTER of an ARC/CIRCLE.	
Creates a point associated with the END of a curve.	
Creates a point through interaction.	
Creates a point associated with the point of INTERSECTION of two curves.	
Creates a point associated with the MIDDLE of a curve.	
Creates a point associated with the PICK option.	
Creates a geometric curve by its ID.	
Creates a geometric curve interactively.	
CRVASO	5-25
Create a user defined geometrical entity for a curve.	
CRVASP	5-25
Create a geometrical entity for a curve interactively.	
PTASCE	5-25
Create a geometric point for the CENTER option.	
PTASEN	5-26
Create a geometric point for the END option.	

PTASID	5-26
Create a geometric point interactively.	
PTASIN	5-27
Create a geometric point for the INTERSECTION option.	
PTASMI	5-27
Create a geometric point for the MIDDLE option.	
PTASPI	5-28
Create a geometric point for the PICK option.	
GEOTOL	
CDK_GEOTOL_CREATE	5-29
Create a geometrical tolerance.	
CDK_GEOTOL_GET	5-30
Obtain the geometrical tolerance data.	
CDK_GEOTOL_SET	5-31
Sets variables for the geometrical tolerance data.	
HATCH	
DHATCH	5-32
Create hatching. Refer to the Cimatron ^{it} Drafting Manual , for explanations of the parameters.	
IDNUM	
CDK_IDNUM_GSC	5-33
Get/Set/Create the graphical part of ID_NUM.	
LABEL	
CDK_LABEL_CONNECTION	5-34
Edit connection between the leader and label frame.	
CDK_LABEL_CREATE	5-35
Create a label.	
CDK_LABEL_LEADER	5-36
Edit the leader of the label.	
CDK_LABEL_LEADER2	5-37
Get a label leader and plane.	
CDK_LABEL_NUMLEADERPOINTS	5-37
Obtain the number of leader points.	
CDK_LABEL_POSITION	5-38
Edit the label frame position.	

CDK_LABEL_SET	5-38
Set label parameters.	
CDK_LABEL_TEXT	5-39
Edit label text.	
CDK_LABEL_UPDATE	5-39
Update a label with a plane definition.	
LINEAR DIMENSION	
DILINA	5-40
Create a dimension entity for a linear.	
DILNEX	5-41
Extract all the specific modals for dimension entities of linear.	
DILNPA	5-41
Set all the specific modals for dimension entities of linear.	
DILNPN	5-42
Set modal values of linear dimensions.	
DILNPR	5-43
Set modal values of linear dimensions.	
DIMLIN	5-43
Create a dimension entity for a line.	
NOTE	
CDK_EXPNOTE	5-45
Explode a note and give all the entities (line & arc) from which this note consists. The note must have been created using the options HOR-NORMAL and VER-NORMAL.	
CDK_FONT_CLOSE	5-45
Close the font list.	
CDK_FONT_GET_ACT	5-45
Get the active font name.	
CDK_FONT_GET_ACT_UNI	5-46
Get active font name in UNICODE.	
CDK_FONT_GETNUM	5-46
Get the number of fonts.	
CDK_FONT_INIT	5-46
Initialize the font list.	
CDK_FONT_LIST	5-47
Get the list of fonts.	

CDK_FONT_IND2NAME	5-47
Get the font name by index.	
CDK_FONT_SET_ACTI	5-47
Set the active font by index.	
CDK_FONT_SET_ACTN	5-48
Set the active font by name.	
CDK_NOTE_CREATE	5-48
Create a note.	
CDK_NOTE_DIMENSIONS	5-48
Get box dimensions for a given text, with the current active font, size and scheme.	
CDK_NOTE_POSITION	5-49
Edit the frame position of notes.	
CDK_NOTE_SET	5-49
Set the parameters to be used when creating notes.	
CDK_NOTE_TEXT	5-50
Get / set note text.	
CDK_TEXT_EDIT_PARAMS	5-51
Get/Set the text parameters of a text entity (Note/Label).	
CDK_TEXT_EDITOR	5-51
Open the text editor window and wait for the user to edit text.	
CDK_TEXT_PARAMS	5-52
Set text parameters.	
ORDINATE DIMENSION	
DIMORD	5-53
Create an ordinate dimension for a point entity.	
DIORDA	5-54
Create an ordina dimension.	
DIOREX.	5-54
Extract all the specific modals for dimension entities of ordina.	
DIORPA.	5-55
Set all the specific modals for dimension entities of ordina.	
DIORPR	5-56
Set modal values for ordinate dimensions. Refer to the Cimatron ^{it} Drafting Manual , for explanations of the modal parameters.	
RADIAL DIMENSION	

DIMRAD	5-57
Create dimension entities for arcs or circles.	
DIRDEX.	5-58
Extract all the specific modals for dimension entities of linear.	
DIRDPA.	5-58
Set all the specific modals for dimension entities of circular.	
DIRDPR.	5-59
Set modal values for radial dimensions. Refer to the Cimatron ^{it} Drafting Manual , for explanations of the modal parameters.	
STYLE	
CDK_FONT_ASCII_TO_UNICODE.	5-60
Convert a string written in ASCII to Unicode.	
CDK_FONT_UNICODE_TO_ASCII.	5-60
Convert a string written in Unicode to ASCII.	
CDK_STYLE_DELETE	5-61
Delete a style from the styles list.	
CDK_STYLE_EDIT	5-61
Edit style attributes.	
CDK_STYLE_GET_ACTIVE.	5-61
Get the name of the active style.	
CDK_STYLE_GET_BY_NAME	5-62
Get style attributes by the StyleName.	
CDK_STYLE_GET_DEFAULT	5-62
Return the default style values.	
CDK_STYLE_LIST	5-62
Get styles list.	
CDK_STYLE_MODIFY_ENTITIES	5-63
Modify the style of an existing entity.	
CDK_STYLE_MODIFY_ENTITIES_IA	5-63
Modify styles of entities interactively.	
CDK_STYLE_NEW	5-63
Create a new style.	
CDK_STYLE_SET_ACTIVE.	5-64
Set the active style.	
CDK_STYLE_TEXT_INIT	5-64
Set style attributes for TEXT and NOTE functions.	

VIEWS

CDK_DRAW_NEW	5-65
Define a new drawing.	
CDK_SUBS_DAT	5-65
Get a subsystem's name, type and ID by its number.	
CDK_SUBS_TOT	5-66
Get numbers of subsystems for the given type.	
CDK_VIEW_ACT	5-66
Activate a pre-defined view/drawing.	
CDK_VIEW_DEL	5-66
Delete a view.	
CDK_VIEW_NEW	5-67
Define a new view.	
CDK_VIEW_SVR	5-67
Create a reference subview for the current PFM view.	
OBDRVW	5-68
Get the drawing internal views and their reference points and scales.	
OBDVID	5-68
Given a drawing/view number, retrieve the ID of its entity.	
OBDVLC	5-69
Obtain the name of the drawing/view/level, given its number.	
OBDVLL	5-69
List all defined drawings/views/levels.	
OBDVLN	5-70
Obtain the number of the drawing/view/level, given its name.	
OBVWLS	5-70
Given the ID of an entity, list all the views in which it is seen.	
OBVWTR	5-71
Given a view number, return its transformation.	
VWACTV	5-71
Activate and display a pre-defined view.	
VWDFCD	5-72
Define a view using the CURRENT DISPLAY option.	
VWDFMP	5-72
Define a view using the MODEL PROJ. option.	

VWEXIT.	5-73
Close current view with an optional exit to the Modeling application.	

Chapter 6 NC Routines

START PROGRAM

UNC_STR	6-3
Start a USER NC program. Must be called before all the other USER NC subroutines, and may only be called once in a program.	
UNC_STR activates an interaction which is consistent with the general interaction in all the Cimatron ^{it} NC segments. The following modal table is shown: (Refer to the NC Manual)	
By this interaction the USER running the USER NC program may set or verify the tool, machine parameters, global parameters and comment for the created NC procedure.	

OPEN TOOLPATH

UNC_HOME	6-4
Get the toolpath home position.	
UNC_MTYP	6-4
Get the current machine type.	

GLOBAL PARAMS

UNC_CLER	6-5
Get the current clear value.	
UNC_CMPNS	6-5
Send the compensation block to the machine.	
UNC_MORG	6-5
Get the current origin.	
UNC_PRC_CHORG	6-6
Change origin.	
UNC_PRC_CLON.	6-6
Get/Set the clear ON/OFF value of the procedure.	
UNC_PRC_DIA_COMP	6-6
Get/Set the Diameter Compensation ON/OFF value.	
UNC_PRC_DSTL	6-7
Get/Set the tool display flag.	
UNC_PRC_GCLA.	6-7
Get/Set the absolute internal clear value.	

UNC_PRC_GCLI	6-8
Get/Set the incremental internal clear value.	
UNC_PRC_GSCL	6-8
Get/Set the clear value.	
UNC_PRC_INAB	6-8
Get/Set the internal clear mode of the procedure.	
TOOL POSITION	
UNC_TPOS	6-9
Get the current tool position.	
UNC_TVEC	6-9
Get the current tool axis.	
MACHINE PARAMS	
UNC_LATH.	6-10
Get the current machine parameters for Lathe tool paths.	
UNC_MILL	6-10
Get the current machine parameters for Mill tool paths.	
UNC_PUNC	6-11
Get the current machine parameters for Punch tool paths.	
UNC_PRC_BOTT	6-11
Update the bottom menu.	
UNC_PRC_CFD.	6-11
Get/Set the corner feed value for Mill or Lathe tool paths.	
UNC_PRC_COOL.	6-12
Get/Set the cool value for Mill or Lathe tool paths.	
UNC_PRC_DWFD	6-12
Get/Set the down feed value for Mill or Lathe tool paths.	
UNC_PRC_FST	6-13
Get/Set the feed type value for Mill or Lathe tool paths.	
UNC_PRC_NIB	6-13
Get/Set the NIBBLING ON/OFF value for Punch tool paths.	
UNC_PRC_PFD.	6-14
Get/Set the plunge feed value for Mill or Lathe tool paths.	
UNC_PRC_PUN.	6-14
Get/Set the PUNCH ON/OFF value for Punch tool paths.	
UNC_PRC_RFD.	6-14
Get/Set the feed value for Mill or Lathe tool paths.	

UNC_PRC_SDFD	6-15
Get/Set the side feed value for Mill or Lathe tool paths.	
UNC_PRC_SPD	6-15
Get/Set the spindle value.	
UNC_PRC_SPN	6-15
Get/Set the spin value.	
UNC_PRC_VCS	6-16
Get/Set the Vc or spin value for Lathe tool paths.	
UNC_PRC_VCV	6-16
Get/Set the Vc value.	
CURRENT TOOL	
UNC_TAGI	6-17
Get the parameters of the current tool.	
UNC_TCHR	6-18
Get the parameters of the current tool.	
UNC_TDRL	6-18
Get the parameters of the current tool.	
UNC_TGRV	6-19
Get the parameters of the current tool.	
UNC_THLD	6-19
Get the holder number of the current tool.	
UNC_TLAT	6-20
Get current tool parameters for lathe tool.	
UNC_TL_ATT	6-20
Return tool attributes for given ID.	
UNC_TL_CAGI	6-21
Create/Update AGIE tool.	
UNC_TL_CDRL	6-21
Create/Update drill tool.	
UNC_TL_CGRV	6-22
Create/Update groove tool.	
UNC_TL_CLAT	6-22
Create/Update lathe tool.	
UNC_TL_CLDR	6-23
Create/Update lathe drill tool.	

UNC_TL_CMIL	6-23
Create/Update mill tool.	
UNC_TL_CTHR	6-24
Create/Update thread tool.	
UNC_TL_CUR	6-24
Set the current tool.	
UNC_TL_DEL	6-25
Delete the tool entity.	
UNC_TL_ID	6-25
Return tool ID for given type and index.	
UNC_TL_NUM	6-26
Return number of tools of a specific type.	
UNC_TL_SYM	6-26
Set/Unset user defined symbol.	
UNC_TMIL	6-27
Get the parameters of the current tool.	
UNC_TMKN	6-27
Get the parameters of the current tool.	
UNC_TPUN	6-28
Get the parameters of the current tool.	
UNC_TTHR	6-28
Get the parameters of the current tool.	
UNC_TTYP	6-29
Get the type of the current tool.	
TOOL MOTIONS	
UNC_CIR5	6-30
Create a circular motion to end point. 5 axis circular motion. The motion is on the MACSYS XY plane with a Z value of the current tool position.	
UNC_CIRC	6-31
Create circular motion to end point. The motion is on the MACSYS XY plane with a Z value of the current tool position.	
UNC_CIRS	6-31
Create circular motion to end point. 3 axis motion, with 3D compensation vector. The motion is on the MACSYS XY plane with a Z value of the current tool position.	
UNC_LIN5	6-32
Create linear motion to end point. 5 axis linear motion.	

UNC_LINE	6-32
Create linear motion to end point.	
UNC_LINS	6-32
Create linear motion to end point. 3 axis motion, with 3D compensation vector.	
NEW MACHINE PARAMS	
UNC_BYVC	6-33
Set by Vc or by SPIN.	
UNC_COOL	6-33
Set to new cool value.	
UNC_FAST.	6-34
Set to fast motion.	
UNC_FEED	6-34
Set to new feed value.	
UNC_PONF	6-34
Set to punch off or punch on.	
UNC_RFED	6-35
Set to feed motion.	
UNC_SPIN.	6-35
Set to new spin value.	
UNC_SPND	6-35
Set to new spindle direction.	
UNC_VCVL.	6-36
Set to new Vc value.	
MARKERS	
UNC_MAPP	6-37
Create an open approach marker or a close approach marker.	
UNC_MLAY	6-37
Create an open layer marker or a close layer marker.	
UNC_MPAS	6-38
Create an open pass marker or a close pass marker.	
UNC_MRTR	6-38
Create an open retract marker or a close retract marker.	
MESSAGE	
UNC_MSG	6-39
Enter a message.	

TOOL

UNC_ID_TDRL	6-40
Get the parameters of a tool (by ID). Note: For drill tool.	
UNC_ID_TGRV	6-40
Get the parameters of a tool (by ID). Note: For groove tool.	
UNC_ID_TLAT.	6-41
Get the parameters of a tool (by ID). Note: For lathe tool.	
UNC_ID_TMIL	6-41
Get the parameters of a tool (by ID). Note: For mill tool.	
UNC_ID_TPUN	6-42
Get the parameters of a tool (by ID). Note: For punch tool.	
UNC_ID_TTHR	6-42
Get the parameters of a tool (by ID). Note: For thread tool.	
UNC_TPTA.	6-43
Given the tool tip point and the tool parameters, calculate the tool tangent point.	
UNC_TPTP.	6-43
Given the tool tangent point and the tool parameters, calculate the tool tip point.	
UNC_TTAN.	6-44
Given the tool tip point, calculate the tangent point for the current tool. Note: For a mill tool.	
UNC_TTIP	6-44
Given the tool tangent point, calculate the tip point for the current tool. Note: For a mill tool.	

CLOSED TOOLPATH

OBTPNM	6-45
Obtain the name of a tool path.	
UGTNTP	6-45
Get the next Tool Path ID.	

TOOLPATH

GETTPTEXT	6-46
Get the toolpath text.	
UNC_TP_CHK.	6-46
Check if the tool path exists.	
UNC_TP_CLOSE	6-47
Close a tool path.	

UNC_TP_CREA	6-47
Create a tool path.	
UNC_TP_FLAGS	6-48
Read the flags of a procedure in a ToolPath.	
UNC_TP_IDCUR	6-48
Returns the ID of the currently open tool path.	
UNC_TP_NUM	6-48
Returns the number of tool paths in the current MACSYS.	
UNC_TP_PAR	6-49
Get the tool path parameter list.	
UNC_TP_REN	6-49
Rename an existing tool path.	
UNC_TP_REOP	6-49
Reopen a tool path.	
MACSYS	
UNC_MACS_ACT	6-50
Activate the defined MACSYS.	
UNC_MACS_DEL	6-50
Delete a MACSYS.	
UNC_MACS_NEW	6-50
Define a new MACSYS.	
PROCEDURE MANAGEMENT	
UNC_PRC_CLOSE	6-51
Close a procedure.	
UNC_PRC_DEL	6-51
Delete a procedure.	
UNC_PRC_INIT	6-51
Initialize a procedure by creating an NC procedure without interaction.	
UNC_PRC_INTER	6-52
Do procedure interaction after running UNC_PRC_INIT (see page 6-51).	
Do the extended interaction for an NC procedure. Only defined masks will be used. The first mask may be:	
UNC_IM_CMPNS = allow cutter diam compensation modal	
or 0.	
The second mask must be 0.	
UNC_PRC_NUM	6-52
Obtain the number of procedures in a tool path.	

UNC_PRC_OPEN.	6-52
Open a procedure.	
UNC_PRC_TXT	6-53
Get/Set the text of the procedure.	
TEMPLATES	
UNC_APPEND_PCT2TPT.	6-54
Appends a PCT file to the end of a TPT file.	
UNC_TMPL_RW	6-54
Read/write a list of parameters from/to the PCT template file. The memory of ioParVal should be allocated/freed by the caller.	
UNC_TMPL_TPT_RW	6-55
Reading/writing parameter value within a PCT section in a TPT file.	
UNC_TPT_PROC_NUM	6-55
Returns the number of PCT sections in a given TPT file.	

Chapter 7 Solid Routines

SKETCHER

CDK_INIT_NEW.	7-4
Initialize a new sketcher from CDK	
CDK_SK_END.	7-4
End of the sketcher from CDK.	
CDK_SK_SOLVE	7-4
Solve sketcher from CDK.	
CDK_SK_SAVE	7-5
Solve and save the sketcher CDK in an SKF file.	
CDK_SK_GET_IDS.	7-5
Get entity and instance IDs.	
CDK_SK_GM_ADDX_CRV	7-5
Add a curve as a wire frame geometry to the sketcher.	
CDK_SK_GM_ADDX_SYM	7-6
Add symmetry lines to the sketcher.	
CDK_SK_DIM_ADDX.	7-6
Add an external dimension to the sketcher.	
CDK_SK_ADD_CONSTRAINT	7-7
Add sketcher constraints.	

CDK_SK_EXECUTE0	7-7
Execute sketcher.	
CDK_SK_SYMBOL_CREATE	7-8
Create a symbol for given sketcher dimension.	
CDK_SK_CRMODE_SET	7-8
Set creation mode of sketcher.	
CDK_SOL_SK_INIT	7-9
Initialize sketcher within solid operation.	
CDK_SOL_SK_DONE	7-9
Execute sketcher within solid operation.	
REFERENCE GEOMETRY	
CDK_SOL_REFGEOM_AXIS	7-10
Add reference geometry axis.	
CDK_SOL_REFGEOM_ENTITY	7-10
Add reference entities.	
CDK_SOL_USEGEOM	7-10
Add reference entities to the sketch.	
SOLID OPERATIONS	
CDK_SOL_ARRAY	7-11
Copy object / feature as COPY >> ARRAY.	
CDK_SOL_AXIS	7-11
Create axis.	
CDK_SOL_BOOLEAN	7-11
Perform Boolean operations.	
CDK_SOL_DRIVE	7-11
Execute DRIVE operation.	
CDK_SOL_EXTRUDE	7-12
Execute EXTRUDE operation	
CDK_SOL_HOLE	7-12
Execute HOLE operation.	
CDK_SOL_IMPORT	7-12
Execute import operation.	
CDK_SOL_IMPORT_DEFAULT	7-12
Set default parameters to import data.	
CDK_SOL_PLANE	7-13
Create a plane entity.	

CDK_SOL_REFCRV	7-13
Execute REFERENCE CURVE operation.	
CDK_SOL_REVOLVE	7-13
Execute REVOLVE operation.	
CDK_SOL_RIB	7-13
Execute RIB operation.	
CDK_SOL_ROTATE	7-14
Rotate object.	
CDK_SOL_ROUND	7-14
Execute Round/Chamfer operation.	
CDK_SOL_SYMMETRY	7-14
Create symmetry object.	
CDK_SOL_TRANSLATE	7-14
Move (translate) a solid object.	
SOLID DATA BASE ACCESS	
CDK_SOL_AXIS_DATA	7-15
Get axis data.	
CDK_SOL_AXIS_FIRST	7-15
Get the first axis.	
CDK_SOL_AXIS_NEXT	7-15
Get the next axis.	
CDK_SOL_EDGE_CRVLIM	7-15
Get the limits of the edge curve.	
CDK_SOL_EDGE_DATA	7-16
Get edge data.	
CDK_SOL_EDGE_PARI	7-16
Get the surface parameters of both faces of an edge.	
CDK_SOL_EDGE_PARP	7-16
Get the surface parameters of both faces of an edge.	
CDK_SOL_EDGE_SEQ	7-17
Get the next edge.	
CDK_SOL_FACE2PLF	7-17
Create a planar face from a solid face.	
CDK_SOL_FACE2SRF	7-17
Create a planar face from a solid face.	

CDK_SOL_FACE2TRS.	7-17
Create trimmed surface out of solid face.	
CDK_SOL_FACE2WFM	7-18
Create a wireframe entity from a solid face.	
CDK_SOL_FACE_BOX_WRK.	7-18
Calculate the enclosing box of an object in WORK coordinates.	
CDK_SOL_FACE_DATA.	7-18
Get face data.	
CDK_SOL_FACE_FIRST	7-18
Get the first face entity.	
CDK_SOL_FACE_NEXT.	7-19
Get the next face entity.	
CDK_SOL_FACE_SK_PLANE	7-19
Get the Sketcher plane of a face.	
CDK_SOL_LOOP_FIRST	7-19
Get the first loop.	
CDK_SOL_LOOP_NEXT.	7-20
Get the next loop.	
CDK_SOL_OBJ_DATA.	7-20
Get solid object data.	
CDK_SOL_OBJ_FIRST	7-20
Get the first solid object.	
CDK_SOL_OBJ_NEXT.	7-20
Get the next solid object.	
CDK_SOL_PLANE_DATA	7-21
Get plane data.	
CDK_SOL_PLANE_FIRST.	7-21
Get the first plane.	
CDK_SOL_PLANE_NEXT	7-21
Get the next plane.	
CDK_SOL_REFCRV_PRIM_IDS	7-21
Obtain the geometric (wireframe) ID's of primitive curves of a reference curve.	
CDK_SOL_REFCRV_PRIM_NUM	7-22
Get the number of primitive curves of a reference curve.	

CDK_SOL_REFPT_COOR	7-22
Get reference point coordinates.	
CDK_SOL_SOLID2CURVES_GS.	7-22
Get the number of edges on a solid object.	
CDK_SOL_SOLID2CURVES_OBTAIN	7-22
Obtain curves of edges of a given solid object.	
UTILITIES	
CDK_ATTACH_SCREW	7-23
Attach a screw, nut or ejector to a solid entity.	
CDK_BB_LIMITS_AT_UCS	7-23
Calculates the bounding box of a solid object in a given UCS.	
CDK_SOL_BB_CENTER_AT_UCS	7-24
Calculates the center of a solid object in a given UCS.	
CDK_SOL_ENT_CTOR	7-24
CimaDEK Solid entity constructor.	
CDK_SOL_FND_SCREW_EDGE.	7-25
Find the ID of the circular edge that coincides with the screw end.	
CDK_SOL_PIERCE_OBJ	7-26
Find all pierce points of a ray and an object.	
CDK_SOL_PIERCE_OBJ1	7-26
Find all pierce points of a ray and an object and sort them. This function is a newer version of function CDK_SOL_PIERCE_OBJ.	
CDK_SOL_SAME	7-26
Check if solid entities are the same.	
CDK_SOL_TRACE_FREE.	7-27
Free allocated trace buffer.	
CDK_SOL_TRACE_FREE1	7-27
Free trace buffer.	
CDK_SOL_UPDATE	7-27
Update solid entity handling.	
CDK_UCLOSE.	7-27
Close the current pfm file that was opened in read only mode.	
CDK_UOPEN	7-27
Open a pfm file in read only mode.	
SOLID PROCEDURE DATA ACCESS	

CDK_SOL_PROC_ELEM	7-28
Read maxNum items with given entity and feat. type	
CDK_SOL_PROC_ELEM_NUM.	7-28
Get the number of items.	
CDK_SOL_PROC_ELEM_PROC.	7-28
Retreive the procedure for a given solid element.	
CDK_SOL_PROC_END	7-28
Free Procedure Data.	
CDK_SOL_PROC_FIRST	7-29
Get the first procedure.	
CDK_SOL_PROC_FREE_DIMENS	7-29
Free the dimension buffer	
CDK_SOL_PROC_GET_DIMENS	7-29
Get feature dimensions.	
CDK_SOL_PROC_INIT	7-29
Init Procedure Data.	
CDK_SOL_PROC_INIT_DIMENS	7-30
Allocate and initialize dimension buffer.	
CDK_SOL_PROC_LAST.	7-30
Get last procedure.	
CDK_SOL_PROC_NAME	7-30
Get the name of a procedure.	
CDK_SOL_PROC_NEXT	7-30
Get the next procedure.	
SOLID PICK FUNCTIONS	
CDK_SOL_GET_PARPT.	7-31
Get parametric point.	
CDK_SOL_PICK	7-31
Pick a solid entity.	
CDK_SOL_PICK_ANY_FACE.	7-31
Pick open face or regular face.	
CDK_SOL_PICK_ANY_LOOP	7-32
Pick open loop or regular loop.	
CDK_SOL_PICK_AXIS.	7-32
Pick a reference axis within given solid object.	

CDK_SOL_PICK_EDGE	7-32
Pick an edge.	
CDK_SOL_PICK_ENABLE	7-32
Enable picking of solid elements	
CDK_SOL_PICK_FACE	7-33
Pick a face.	
CDK_SOL_PICK_FEATURE	7-33
Pick a feature.	
CDK_SOL_PICK_HOLE	7-33
Pick a hole (as part of the feature hole).	
CDK_SOL_PICK_LOOP	7-34
Pick a loop.	
CDK_SOL_PICK_OBJ	7-34
Pick a solid object.	
CDK_SOL_PICK_PLANE	7-34
Pick a plane (planar face or ref. plane).	
CDK_SOL_PICK_REF_GEOM	7-34
Pick any reference geometry solid entity.	
ASSEMBLY	
CDK_PFM_TYPE	7-35
Check the part file type.	
CDK_ASSM_INIT	7-35
Load a solid assembly.	
CDK_ASSM_TERM.	7-35
Terminate a solid assembly.	
CDK_ASSM_PART_NUM	7-35
Get the number of part files.	
CDK_ASSM_PART_NEXT.	7-36
Get data for the next part file.	
SOLID ATTRIBUTES	
CDK_SOL_BLANK	7-37
Blank a solid entity.	
CDK_SOL_BLANK_ATTRIB.	7-37
Get attributes of a blanked solid.	
CDK_SOL_COL_GET	7-37
Get solid color.	

CDK_SOL_COL_SET	7-38
Set solid color.	
CDK_SOL_LATT_GET	7-38
Get solid line attributes	
CDK_SOL_LATT_SET	7-38
Set solid line attributes.	
CDK_SOL_LEVEL_CHANGE	7-39
Move the solid object of the given type, to the desired level.	
CDK_SOL_LEVEL_GET	7-39
Get the solid level.	
CDK_SOL_LEVEL_SET	7-39
Set the solid level	
CDK_SOL_PIERCE_OBJ_SORT	7-40
Find all pierce points of a ray and an object.	
CDK_SOL_SLA_INIT	7-40
Initialize SLA package.	
CDK_SOL_SLA_TRIANG_NEXT	7-40
Get triangle data for the next face.	
CDK_SOL_TRIM	7-41
Delete all procedures after "Proc" from the solid tree.	
CDK_SOL_UNBLANK	7-41
Unblank a solid entity.	
SOLID DISPLAY	
CDK_SOL_ATTENTION_OFF	7-42
Turn attention mode OFF	
CDK_SOL_ATTENTION_ON	7-42
Turn attention mode ON.	
CDK_SOL_ATTEN_HOLE_OFF	7-42
Turn hole attention mode OFF.	
CDK_SOL_ATTEN_HOLE_ON	7-42
Turn hole attention mode ON.	
CDK_SOL_DISP_RESUME	7-43
Unsuspend the display.	
CDK_SOL_DISP_SUSPEND	7-43
Temporarily suspend the display.	

CDK_SOL_DISPLAY_OFF	7-43
Delete a solid entity from the display file.	
CDK_SOL_DISPLAY_ON	7-43
Create a solid entity in the display file.	
CDK_SOL_DISPLAY_UPDATE	7-43
Update the display of a solid entity.	
CDK_SOL_DISPLAY_UPDATE_FULL	7-44
Updates the entire solid display.	
CDK_SOL_PROC_ATTEN_OFF	7-44
Turn feature attention mode OFF.	
CDK_SOL_PROC_ATTEN_ON	7-44
Turn feature attention mode ON.	
CDK_SOL_PREVIEW_CLOSE	7-44
Close the display and the imported file.	
CDK_SOL_PREVIEW_FREE	7-44
Free all allocated memory.	
CDK_SOL_PREVIEW_INIT	7-45
Initialize and display a preview of an imported file.	
CDK_SOL_PREVIEW_LEVELS	7-45
Returns a list of levels in an external pfm file that contains objects in them.	
CDK_SOL_PREVIEW_LEVELS_FREE	7-45
Free the memory of the allocated P_CDK_IMPORT_LEVELS.	
CDK_SOL_PREVIEW_MOVE	7-45
Move the display list of objects.	
CDK_SOL_PREVIEW_PICK	7-46
Set all objects as pickable / unpickable	
SOLID EDIT	
CDK_SOL_DEL_PROC	7-47
Delete a procedure (feature).	
CDK_SOL_EDSK_END	7-47
Close an open Sketcher without resetting it.	
CDK_SOL_EDSK_GET_NUM	7-47
Get the number of Sketchers within a procedure.	
CDK_SOL_EDSK_REGEN	7-48
Regenerate solid after resketch.	

CDK_SOL_EDSK_SET	7-48
Reset a Sketcher of a procedure by an index.	
CDK_SOL_EXPLODE_GS	7-48
Get the number of faces when a given solid is exploded. Define which entities are to be exploded.	
This number is used for calculating the size of array surf ids in utility CDK_SOL_EXPLODE_OBTAIN (page 7-48).	
CDK_SOL_EXPLODE_OBTAIN	7-49
Explode a solid to a set number of faces.	
The number of faces is calculated by utility CDK_SOL_EXPLODE_GS (page 7-48) where a user defines which entities are to be exploded.	
CDK_SOL_EXPLODE_SOLID	7-49
Explode a solid to a set number of surfaces at a specified distance from its faces.	
The number of surfaces is calculated by utility CDK_SOL_EXPLODE_SOLID_GS (page 7-50).	
CDK_SOL_EXPLODE_SOLID_GS	7-50
Get the number of faces for exploding a given solid.	
This number is used for calculating the size of array surf ids in utility CDK_SOL_EXPLODE_SOLID (page 7-48).	
CDK_SOL_MODIFY_DIMENSION	7-50
Edit procedure parameters.	
TRANSLATE WF -> SOLID	
CDK_CYLINDER_RAY_TRACE	7-51
Find the cutting curves defined by a cylinder on a group of surfaces.	
CDK_EJECTOR_CUT	7-52
Create an open solid object according to the curves created by utility CDK_CYLINDER_RAY_TRACE (page 7-51).	
CDK_EJECTOR_CUT2	7-52
Cut the ejector (solid cylinder) with the open solid object created by utility CDK_EJECTOR_CUT (page 7-52).	
CDK_EJECTOR_CUT_CLEAN	7-53
Delete temporary entities after ejector cut. See CDK_EJECTOR_CUT (page 7-52).	
CDK_SOL_SRF2SOL	7-53
Create solid object from surfaces.	
CDK_SOL_WF2DATUM	7-53
Translate wireframe entities to solid datum.	

CDK_SRF_TRACE	7-53
Perform a Ray Trace on a group of surfaces.	

Chapter 8 External User Package

EUREAD	8-2
Check and read a part file.	
EUSAVE	8-2
Save a part file.	

Index

A

ACTUCS
 ACTivate an existing UCS 4-143

ADD_PATH_CHG
 Add old & new pathnames 3-51

ANALYZE
 See Routines

AR2CRV
 ARc by 2 CuRVes 4-4

AR3PTD
 ARc by 3 PoiNTs-Double 4-4

AR3PTS
 ARc by 3 PoiNTS 4-5

ARCNRA
 ARc by CeNter and RADIUS 4-5

ARCNRAD
 ARc by CeNter and RADIUS-Double 4-6

AREX2C
 ARc by EXtended 2 Curves 4-6

Arithmetic
 Elementary Arithmetic 3-60

ASSEMBLY
 See Routines

ATDEID
 ATtach/DEtach record ID 3-3

ATGECO
 Retrieve entities or records 3-4

ATSUMU
 Sum of geometric entity values 3-4

ATTDET
 ATTach/DETach 3-5

ATTINI
 ATTribute INItialization 3-5

ATTRIB
 See Routines

AXI_TYPE_OPEN
 Get type of active part 3-108

B

BEST_CRC
 Find best circle from given points 4-7

C

CDK_ARC_2PNT
 ARC/circle by 2PoiNTs 4-8

CDK_ARC_3CRV
 ARC/circle tangent to 3 CuRVes 4-7

CDK_ARC_3PNT
 ARC/ circle by 3 PoiNTs 4-8

CDK_ARC_CENT_RAD
 ARC/circle by CENTer & RAD 4-9

CDK_ARC_PNT2CRV
 ARC/circle tangent to 2CRVs & PNT
 4-9

CDK_ASSM_INIT
 Load a solid assembly 7-35

CDK_ASSM_PART_NEXT
 Get data for the next part file 7-36

CDK_ASSM_PART_NUM
 Get the number of part files 7-35

CDK_ASSM_TERM
 Terminate a solid assembly 7-35

CDK_ATTACH_SCREW
 Attach a screw, nut, ejector to a solid
 entity 7-23

CDK_BALCON
 Get contours with specific entity 4-13

CDK_BB_LIMITS_AT_UCS
 Calc bounding box of solid object in given
 UCS 7-23

CDK_BLEND_REGION
 Create a trimmed surface given outer loop
 as contour 4-87

Index

CDK_BLEND_SECTIONS	
Create a blend section surface	4-88
CDK_COMCRV_GET	
GET COMponents of CuRve	4-13
CDK_CONIC_3PNT	
Create CONIC by 3PoiNTs	4-10
CDK_CONIC_WP	
Create CONIC parallel to Wrk Pln	4-11
CDK_CONTOUR_MULTY	
Create 2D multi-contour buffer	4-14
CDK_CRV_APPROX	
APPROXimation CuRVes by arcs & lines	4-72
CDK_CRV_DST_CRV	
Find the closest points between two curves	4-73
CDK_CRV_TRMLOOP	
Prepare planar curve to chain of curves	4-73
CDK_CRV2POLY	
Create polygon by stepping along curve	4-71
CDK_CURIN	
Receive Input from Mouse or Keyboard	3-92
CDK_CURVE_FAIR	
Fairing on a curve	4-74
CDK_CYLINDER_RAY_TRACE	
Find cutting curves defined by a cylinder on group surfaces	7-51
CDK_DRAW_NEW	
Define a new drawing	5-65
CDK_DSPTOL	
Display tol of curves, surfs, planar	3-52
CDK_EJECTOR_CUT	
Cut ejector with surface out of curves	7-52
CDK_EJECTOR_CUT_CLEAN	
Delete temporary entities after ejector cut	7-53
CDK_EJECTOR_CUT2	
Cut ejector with surface out of curves	7-52
CDK_ELLIPSE	
Create ELLIPSE parallel to Wrk Pln	4-11
CDK_EXPNOTE	
EXPlode a NOTe	5-45
CDK_EXTRACT_SUB_ASSY	
Extract as EXTRACT >> EXTRACT SUB ASSEMBLY	3-108
CDK_FILE_BROWSE	
Interactive file browser	3-23
CDK_FONT_ASCII_TO_UNICODE	
Convert a string from ASCII to Unicode	5-60
CDK_FONT_CLOSE	
Close the font list.	5-45
CDK_FONT_GET_ACT	
Get active font name.	5-45
CDK_FONT_GET_ACT_UNI	
Get active font name in UNICODE	5-46
CDK_FONT_GETNUM	
Get the number of fonts.	5-46
CDK_FONT_IND2NAME	
Get font name by index.	5-47
CDK_FONT_INIT	
Initialize the font list.	5-46
CDK_FONT_LIST	
Get the list of fonts.	5-47
CDK_FONT_SET_ACTI	
Set active font by index.	5-47
CDK_FONT_SET_ACTN	
Set active font name.	5-48
CDK_FONT_UNICODE_TO_ASCII	
Convert a string from Unicode to ASCII	5-60
CDK_GEOTOL_CREATE	
Create a geotol.	5-29
CDK_GEOTOL_GET	
Obtain geotol data.	5-30
CDK_GEOTOL_SET	
Set variables for geotol data.	5-31

CDK_HOT_VIEW		
Change the view picture	3-23	
CDK_IDNUM_GSC		
Create/Set/Get the ID_NUM.	5-33	
CDK_INIT_NEW		
Initialize a new sketcher from CDK	7-4	
CDK_LABEL_CONNECTION		
Connection	5-34	
CDK_LABEL_CREATE		
Create a label.	5-35	
CDK_LABEL_LEADER		
Edit label leader.	5-36	
CDK_LABEL_LEADER2		
Get labels leader and plane	5-37	
CDK_LABEL_NUMLEADERPOINTS		
Obtain number of leader points	5-37	
CDK_LABEL_POSITION		
Edit label frame position.	5-38	
CDK_LABEL_SET		
Set label params.	5-38	
CDK_LABEL_TEXT		
Edit label text.	5-39	
CDK_LABEL_UPDATE		
Update label with plane definition	5-39	
CDK_LEV_GSNAME		
Get/set the name of a level	3-43	
CDK_LEVEL_ACTIVATE		
Activate a predefined level	3-23	
CDK_LEVELS_DSP_MASK		
Obtain displayed level mask of a view port	3-24	
CDK_LEVELS_ENT_MASK		
Obtain level mask of all entities	3-24	
CDK_LEVELS_LIST		
Get list of level names, numbers	3-43	
CDK_LINE_OFFSET		
Create LINE OFFSET	4-28	
CDK_MENU_POPUP		
Call the popup menu from a user application	3-93	
CDK_MENU_POPUP_DEF		
Invoke a popup menu with line and column definition options	3-24	
CDK_MODAL_SET_COLORS		
Set colors for modal table	3-25	
CDK_MODAL_SET_SIZE		
Set number of modals and size	3-25	
CDK_MOUSE_INPUT		
Receive input from the mouse	3-93	
CDK_NOTE_CREATE		
Create a note.	5-48	
CDK_NOTE_DIMENSIONS		
Get box dimensions for a given text.	5-48	
CDK_NOTE_POSITION		
Edit the frame position of notes.	5-49	
CDK_NOTE_SET		
Set note creating parameters	5-49	
CDK_NOTE_TEXT		
Get / set note text	5-50	
CDK_OFFSET		
Create entity by OFFSET	4-36	
CDK_OFFSET_CONT		
OFFSET of CONTOUR	4-37	
CDK_PICK_WINDOW		
Pick a window	3-94	
CDK_PLF_BOXES		
Get boxes for planar face	4-46	
CDK_PLF_CONCRV		
Get # curves in single contour	4-46	
CDK_PLF_CREATE		
Create a planar face	4-47	
CDK_PLF_EXPLODE		
Explode a planar face	4-47	
CDK_PLF_GETCRV		
Get curve ID from planar face def	4-47	
CDK_PLF_TOTCRV		
Get # contours & total # curves	4-48	

Index

CDK_PLN_INTERSEC	
Calculate intersec of plane and curve	4-88
CDK_PLOT_BOX	
Immediate plot	3-104
CDK_PLOT_DEVICE_LIST	
Get the plot device list	3-104
CDK_PLOT_GET_DEVICE_NAME	
Get the name of the device	3-105
CDK_PLOT_OFFSET	
Set the plotting offsets	3-105
CDK_PLOT_SET_COLORMAP	
Set color map	3-105
CDK_PLOT_SET_ROTATE	
Set plotting rotate flag	3-106
CDK_PLOT_SET_SPEED	
Set the plotting speed	3-106
CDK_PNT_BSPLINE	
Create points of a B-spline	4-50
CDK_PNT_COOR	
PoiNT defined by COORdinates	4-50
CDK_POINT_EQUAL	
Check if 2 3D points are equal	3-64
CDK_PRG_INFO	
Return string with ver & build #	3-109
CDK_PRM_TYPE	
Check the part file type	7-35
CDK_PROJ_CNTR_PLANE	
Project a curve onto the active work plane	4-60
CDK_PROJ_CNTR_SRF	
Project a contour onto a surface	4-61
CDK_PROJ_CNTR_SRF_FREE	
Free memory allocated in CDK_PROJ_CNTR_SRF	4-61
CDK_PROJ_CURVE_PLANE	
Project a curve onto the active work plane	4-62
CDK_PROJ_CURVE_SRF	
Project a curve onto a surface	4-62
CDK_PROJ_CURVE_SRF_FREE	
Free memory allocated in CDK_PROJ_CURVE_SRF	4-63
CDK_PROJ_POINT_PLANE	
Project a point onto a plane	4-63
CDK_PROJ_POINT_SRF_FREE	
Free memory allocated in CDK_PROJ_POINT_SRF	4-64
CDK_PROJ_POINT_SRF	
Project a point onto a surface	4-64
CDK_PROJ_WPDIR	
PROJect to Work Plane in DIRection	4-65
CDK_PROJ_WPNORM	
PROJect to Work Plane in NORMAl dir.	4-65
CDK_PROJ_XYDIR	
PROJect to XY plane in DIRection	4-66
CDK_PROJ_XYNORM	
PROJect to XY plane in NORMAl dir.	4-66
CDK_QE_REPORT_CREATE	
Create an electrode report	3-107
CDK_RUBBER_CIRCLE	
Draws a rubber circle	3-15
CDK_RUBBER_LINE	
Draws a rubber line	3-15
CDK_SCALE	
Create a SCALEd entity	4-38
CDK_SDK_EXECUTE0	
Execute sketcher	7-7
CDK_SK_ADD_CONSTRAINT	
Add sketcher constraints	7-7
CDK_SK_CRMODE_SET	
Set creation mode of sketcher	7-8
CDK_SK_DIM_ADDX	
Add an external dimension to the sketcher	7-6
CDK_SK_END	
End of the sketcher from CDK	7-4
CDK_SK_GET_IDS	
Get entity and instance IDs.	7-5

- CDK_SK_GM_ADDX_CRV
Add a curve as a wire frame geometry to the sketcher 7-5
- CDK_SK_GM_ADDX_SYM
Add symmetry lines to the sketcher 7-6
- CDK_SK_SAVE
Solve and save the sketcher CDK in an SKF file 7-5
- CDK_SK_SOLVE
Solve sketcher 7-4
- CDK_SK_SYMBOL_CREATE
Create a symbol for given sketcher dimension 7-8
- CDK_SLA_CREATE
Create SLA surface 4-89
- CDK_SLA_EXIT
Close CDK_SLA, clear allocated mem. 4-89
- CDK_SLA_FACE_ADD
Create SLA face 4-89
- CDK_SLA_INI
Activate the CDK_SLA package 4-89
- CDK_SLA_NODE_ADD
Create SLA node 4-89
- CDK_SOL_ARRAY
Copy object / feature as COPY >> ARRAY 7-11
- CDK_SOL_ATTEN_HOLE_OFF
Turn hole attention mode OFF 7-42
- CDK_SOL_ATTEN_HOLE_ON
Turn hole attention mode ON 7-42
- CDK_SOL_ATTENTION_OFF
Turn attention mode OFF 7-42
- CDK_SOL_ATTENTION_ON
Turn attention mode ON 7-42
- CDK_SOL_AXIS
Create axis 7-11
- CDK_SOL_AXIS_DATA
Get axis data 7-15
- CDK_SOL_AXIS_FIRST
Get the first axis 7-15
- CDK_SOL_AXIS_NEXT
Get the next axis 7-15
- CDK_SOL_BB_CENTER_AT_UCS
Calc center of solid object in given UCS 7-24
- CDK_SOL_BLANK
Blank a solid entity 7-37
- CDK_SOL_BLANK_ATTRIB
Get attributes of a blanked solid 7-37
- CDK_SOL_BOOLEAN
Perform Boolean operations 7-11
- CDK_SOL_COL_GET
Get solid color 7-37
- CDK_SOL_COL_SET
Set solid color 7-38
- CDK_SOL_DEL_PROC
Delete a procedure (feature) 7-47
- CDK_SOL_DISP_RESUME
Unsuspend the display 7-43
- CDK_SOL_DISP_SUSPEND
Temporarily suspend the display 7-43
- CDK_SOL_DISPLAY_OFF
Delete a solid entity from the display file 7-43
- CDK_SOL_DISPLAY_ON
Create a solid entity in the display file 7-43
- CDK_SOL_DISPLAY_UPDATE
Update the display of a solid entity 7-43
- CDK_SOL_DISPLAY_UPDATE_FULL
Update the solid display 7-44
- CDK_SOL_DRIVE
Execute DRIVE operation 7-11
- CDK_SOL_EDGE_CRVLM
Get the limits of the edge curve 7-15
- CDK_SOL_EDGE_DATA
Get edge data 7-16
- CDK_SOL_EDGE_PARI
Get the surface parameters of the both faces of an edge 7-16

Index

CDK_SOL_EDGE_PARP	Get the surf. param. of both faces of an edge 7-16	CDK_SOL_FACE2PLF	Create a planar face from a solid face 7-17
CDK_SOL_EDGE_SEQ	Get the next edge 7-17	CDK_SOL_FACE2SRF	Create a planar face from a solid face 7-17
CDK_SOL_EDSK_END	Close an open Sketcher without resetting 7-47	CDK_SOL_FACE2TRS	Create trimmed surface out of solid face 7-17
CDK_SOL_EDSK_GET_NUM	Get the number of Sketchers within a procedure 7-47	CDK_SOL_FACE2WFM	Create a WF entity from a solid face 7-18
CDK_SOL_EDSK_REGEN	Regenerate solid after resketch 7-48	CDK_SOL_FND_SCREW_EDGE	Find ID of circ. edge coinciding with screw end profile 7-25
CDK_SOL_EDSK_SET	Reset a Sketcher of a procedure by an index 7-48	CDK_SOL_GET_PART	Get parametric point 7-31
CDK_SOL_ENT_CTOR	CimaDEK Solid entity constructor 7-24	CDK_SOL_HOLE	Execute HOLE operation 7-12
CDK_SOL_EXPLODE_GS	Get the number of faces for exploding a given solid 7-48	CDK_SOL_IMPORT	Execute import operation 7-12
CDK_SOL_EXPLODE_OBTAIN	Get list of faces for exploding a given solid 7-49	CDK_SOL_IMPORT_DEFAULT	Set default parameters to import data 7-12
CDK_SOL_EXPLODE_SOLID	Explode a solid to a set # of surfaces 7-49	CDK_SOL_LATT_GET	Get solid line attributes 7-38
CDK_SOL_EXPLODE_SOLID_GS	Get the number of faces for exploding a given solid 7-50	CDK_SOL_LATT_SET	Set solid line attributes 7-38
CDK_SOL_EXTRUDE	Execute EXTRUDE operation 7-12	CDK_SOL_LEVEL_CHANGE	Move the solid object to the desired level 7-39
CDK_SOL_FACE_BOX_WRK	Calculate the enclosing box of an object 7-18	CDK_SOL_LEVEL_GET	Get the solid level 7-39
CDK_SOL_FACE_DATA	Get face data 7-18	CDK_SOL_LEVEL_SET	Set the solid level 7-39
CDK_SOL_FACE_FIRST	Get the first face entity 7-18	CDK_SOL_LOOP_FIRST	Get the first loop 7-19
CDK_SOL_FACE_NEXT	Get the next face entity 7-19	CDK_SOL_LOOP_NEXT	Get the next loop 7-20
CDK_SOL_FACE_SK_PLANE	Get the Sketcher plane of a face 7-19	CDK_SOL_MODIFY_DIMENSION	Edit procedure parameters 7-50
		CDK_SOL_OBJ_DATA	Get solid object data 7-20

CDK_SOL_OBJ_FIRST		
Get the first solid object	7-20	
CDK_SOL_OBJ_NEXT		
Get the next solid object	7-20	
CDK_SOL_PICK		
Pick a solid entity	7-31	
CDK_SOL_PICK_ANY_FACE		
Pick open face or regular face	7-31	
CDK_SOL_PICK_ANY_LOOP		
Pick open loop or regular loop	7-32	
CDK_SOL_PICK_AXIS		
Pick a reference axis within given solid object	7-32	
CDK_SOL_PICK_EDGE		
Pick an edge	7-32	
CDK_SOL_PICK_ENABLE		
Enable picking of solid elements	7-32	
CDK_SOL_PICK_FACE		
Pick a face	7-33	
CDK_SOL_PICK_FEATURE		
Pick a feature	7-33	
CDK_SOL_PICK_HOLE		
Pick a hole (as part of feature hole)	7-33	
CDK_SOL_PICK_LOOP		
Pick a loop	7-34	
CDK_SOL_PICK_OBJ		
Pick a solid object	7-34	
CDK_SOL_PICK_PLANE		
Pick a plane (planar face or ref. plane)	7-34	
CDK_SOL_PICK_REF_GEOM		
Pick any reference geometry solid entity	7-34	
CDK_SOL_PIERCE_OBJ		
Trace piercing points for all objects by a ray	7-26	
CDK_SOL_PIERCE_OBJ_SORT		
Find all pierce points of a ray and an object	7-40	
CDK_SOL_PIERCE_OBJ1		
Trace piercing points for all objects by a ray	7-26	
CDK_SOL_PLANE		
Create a plane entity	7-13	
CDK_SOL_PLANE_DATA		
Get plane data	7-21	
CDK_SOL_PLANE_FIRST		
Get the first plane	7-21	
CDK_SOL_PLANE_NEXT		
Get the next plane	7-21	
CDK_SOL_PREVIEW_CLOSE		
Close the display and the imported file	7-44	
CDK_SOL_PREVIEW_FREE		
Free all allocated memory	7-44	
CDK_SOL_PREVIEW_INIT		
Initialize and display a preview of an imported file	7-45	
CDK_SOL_PREVIEW_LEVELS		
Returns a list of levels	7-45	
CDK_SOL_PREVIEW_LEVELS_FREE		
Free the memory of allocated P_CDK_IMPORT_LEVELS	7-45	
CDK_SOL_PREVIEW_MOVE		
Move the display list of objects	7-45	
CDK_SOL_PREVIEW_PICK		
Set all objects as pickable / unpickable	7-46	
CDK_SOL_PROC_ATTEN_OFF		
Turn feature attention mode OFF	7-44	
CDK_SOL_PROC_ATTEN_ON		
Turn feature attention mode ON	7-44	
CDK_SOL_PROC_ELEM		
Read maxNum items with given entity and feat. type	7-28	
CDK_SOL_PROC_ELEM_NUM		
Get the number of items	7-28	
CDK_SOL_PROC_ELEM_PROC		
Retrieve the procedure for a given solid element	7-28	
CDK_SOL_PROC_END		
Free Procedure Data	7-28	

Index

CDK_SOL_PROC_FIRST		
Get the first procedure	7-29	
CDK_SOL_PROC_FREE_DIMENS		
Free the dimension buffer	7-29	
CDK_SOL_PROC_GET_DIMENS		
Get feature dimensions	7-29	
CDK_SOL_PROC_INIT		
Init Procedure Data	7-29	
CDK_SOL_PROC_INIT_DIMENS		
Allocate and initialize dimension buffer	7-30	
CDK_SOL_PROC_LAST		
Get last procedure	7-30	
CDK_SOL_PROC_NAME		
Get the name of a procedure	7-30	
CDK_SOL_PROC_NEXT		
Get the next procedure	7-30	
CDK_SOL_REFRCRV		
Execute REFERENCE CURVE operation	7-13	
CDK_SOL_REFRCRV_PRIM_IDS		
Obtain WF ID's of primitive curves of a reference curve	7-21	
CDK_SOL_REFRCRV_PRIM_NUM		
Get the number of primitive curves of a reference curve	7-22	
CDK_SOL_REFGEOM_AXIS		
Add reference geometry axis	7-10	
CDK_SOL_REFGEOM_ENTITY		
Add reference entities	7-10	
CDK_SOL_REFPT_COOR		
Get reference point coordinates	7-22	
CDK_SOL_REVOLVE		
Execute REVOLVE operation	7-13	
CDK_SOL_RIB		
Execute RIB operation	7-13	
CDK_SOL_ROTATE		
Rotate object	7-14	
CDK_SOL_ROUND		
Execute Round/Chamfer operation	7-14	
CDK_SOL_SAME		
Check if solid entities are the same	7-26	
CDK_SOL_SK_DONE		
Execute sketcher within solid operation	7-9	
CDK_SOL_SK_INIT		
Initialize sketcher within solid operation	7-9	
CDK_SOL_SLA_INIT		
Initialize SLA package	7-40	
CDK_SOL_SLA_TRIANG_NEXT		
Get triangle data for the next face	7-40	
CDK_SOL_SOLID2CURVES_GS		
Get the number of edges on a solid object	7-22	
CDK_SOL_SOLID2CURVES_OBTAIN		
Obtain curves of edges of a given object	7-22	
CDK_SOL_SRF2SOL		
Create solid object from surfaces	7-53	
CDK_SOL_SYMMETRY		
Create symmetry object	7-14	
CDK_SOL_TRACE_FREE		
Free allocated trace buffer	7-27	
CDK_SOL_TRACE_FREE1		
Free trace buffer.	7-27	
CDK_SOL_TRANSLATE		
Move (translate) a solid object	7-14	
CDK_SOL_TRIM		
Delete all procedures after "Proc" from the solid tree	7-41	
CDK_SOL_UNBLANK		
Unblank a solid entity	7-41	
CDK_SOL_UPDATE		
Update solid entity handle	7-27	
CDK_SOL_USEGEOM		
Add reference entities to the sketch	7-10	
CDK_SOL_WF2DATUM		
Translate wireframe entities to solid datum	7-53	
CDK_SPLINE_ADJOIN		
Adjoin curves	4-74	

CDK_SPLINE_CP		
Create B SPLINE, defining Control Points	4-75	
CDK_SPLINE_CREATE		
Create a NURBS spline	4-76	
CDK_SPLINE_CUBIC		
Create a cubic spline.	4-77	
CDK_SPLINE_FP		
Create B SPLINE, defining Fairing Points	4-78	
CDK_SPLINE_NURBS_DATA		
Get control points and knots of a B-spline	4-78	
CDK_SPLINE_NURBS_SIZE		
Get parameters of a NURBS spline for allocation purposes	4-79	
CDK_SPLINE_TP		
Create B SPLINE, defining Thru Points	4-79	
CDK_SRF_BOX		
Calculate surface enclosing box	4-90	
CDK_SRF_CRPIP		
Create nurbs surface in a DRIVE option	4-90	
CDK_SRF_DRPRL		
Create a drive surface	4-91	
CDK_SRF_FAIR		
SuRF FAIRing	4-91	
CDK_SRF_INTERSEC		
Calc. intersection points bet. surf/curve	4-92	
CDK_SRF_INTERSECTION		
Calc. global/local intersection between 2 surfaces	4-92	
CDK_SRF_MODIFY_LIM		
Modify surface limits	4-93	
CDK_SRF_MULTY_DRIVE		
Create a multy drive surface	4-93	
CDK_SRF_NORMAL		
Normal of surface at point	4-94	
CDK_SRF_REFINE		
Get/set surface refine factor	4-94	
CDK_SRF_RULCS		
Create RULEd SuRFace	4-95	
CDK_SRF_SECWPL		
Create spline intersections.	4-95	
CDK_SRF_TRACE		
Perform a Ray Trace on a group of surfaces	7-53	
CDK_SRF_TRM_CON		
Trim a surface by a contour	4-96	
CDK_SRF_TRM_PAR		
Trim a surface by a parameter	4-97	
CDK_SRF_TRM_PCON		
Trim a surface by a projected contour	4-98	
CDK_SRF_TRM_SRF		
Trim a surface by a surface	4-99	
CDK_SRF_TRM_SRF1		
Trim a base surface by multi surfaces	4-100	
CDK_SRF_TRM_WPL		
Trim a surface by the work plane	4-101	
CDK_STYLE_DELETE		
Delete a style from the list	5-61	
CDK_STYLE_EDIT		
Edit style attributes	5-61	
CDK_STYLE_GET_ACTIVE		
Get the name of the active style	5-61	
CDK_STYLE_GET_BY_NAME		
Set style attributes by name	5-62	
CDK_STYLE_GET_DEFAULT		
Return the default style values	5-62	
CDK_STYLE_LIST		
Get styles list	5-62	
CDK_STYLE_MODIFY_ENTITIES		
Modify the style of an existing entity	5-63	
CDK_STYLE_MODIFY_ENTITIES_IA		
Modify styles interactively	5-63	
CDK_STYLE_NEW		
Create a new style	5-63	
CDK_STYLE_SET_ACTIVE		
Set the active style	5-64	

Index

CDK_STYLE_TEXT_INIT	
Set style attributes for TEXT and NOTE	
5-64	
CDK_SUBS_DAT	
Get subsystem name, type & ID	5-65
CDK_SUBS_TOT	
Get # of subsystems	5-66
CDK_SURFACE_CREATE	
Create a NURBS surface in the database	
4-102	
CDK_SURFACE_NURBS_DATA	
Get control points and knots of a NURBS surface	4-103
CDK_SURFACE_NURBS_SIZE	
Get parameters of a NURBS surface for allocation purposes	4-103
CDK_SWEEP_ANGLE	
SWEEP entities by ANGLE	4-128
CDK_SWEEP_DELTA	
SWEEP entities by DELTA	4-128
CDK_TEXT_EDIT_PARAMS	
Get/Set text parameters of text entity.	5-51
CDK_TEXT_EDITOR	
Open the text editor window	5-51
CDK_TEXT_PARAMS	
Set text parameters.	5-52
CDK_TRS_BYPLN	
Trim a surface/trimmed surface by work plane	
4-138	
cdk_type.h	5-1
CDK_UCLOSE	
Close the current pfm opened in read only mode	7-27
CDK_UCS_CREATE	
Create UCS by transformatio matrix	4-143
CDK_UCS_CREATE_PTS	
Create UCS by three points	4-144
CDK_UCS_DISPLAY	
Display UCS	4-144
CDK_UCS_LIST	
Create a list of UCS names	4-144
CDK_UCS_MAP	
Map a point from UCS to MODEL or WORK system	4-145
CDK_UCS_MAP_FROM	
Map a point from UCS to MODEL or WORK system	4-145
CDK_UCS_MAP_TO	
Map a point from MODEL or WORK system to UCS	4-145
CDK_UCS_TRANS	
Transform three points into rotation matrix	4-146
CDK_UCS_ZAXIS	
Create UCS from Z axis only	4-146
CDK_UMDDEL	
Delete a modal group	4-22
CDK_UOPEN	
Open a pfm file in read only mode	7-27
CDK_VERIFY_ENTITY	
Obtain class and type of entity	4-152
CDK_VIEW_ACT	
Activate a view/drawing	5-66
CDK_VIEW_DEL	
Delete a view	5-66
CDK_VIEW_NEW	
Define a new view	5-67
CDK_VIEW_SVR	
Create a ref subview for PFM view	5-67
CDK_WINDOW_ACTIVE	
Activate an existing window	3-15
CDK_WINDOW_DELETE	
Delete an existing window	3-16
CDK_WINDOW_MULTY	
Recall multi-window layout	3-16
CDK_WINDOW_NEW_DIVISION	
Define new windows by a division point	3-16
CDK_WINDOW_NEW_NUMERIC	
Define new windows by numeric divide	3-17

- CDK_WINDOW_SINGLE
 Change to a single window 3-17
- CDK_WINDOW_SUB_DIVIDE
 Subdivide a window by a point 3-17
- CDK_WORKPLANE
 Get / set active work plane parameters 4-146
- CDK_WPGETD
 Obtain coeff. of wrk pln.- Double 4-159
- CDK_WRKPLN_3PNT
 WoRK plane define 3 PoiNTs-Double 4-159
- CDK_WRKPLN_3PNT_ROT
 Set a new work plane. 3-85
- CEN2LN
 CENter 2 LiNes 5-11
- Center of Gravity 4-2
- Chamfer 4-18
- CHAMFER
 See Routines
- CHCOMP
 CHaracter COMParE 3-118
- CIRCLE
 See Routines
- CKPT2D
 Check tol. of points on plane 3-85
- Classes 1-1
- Classes and Types 1-1
- CLOSCV
 CLOSeD / periodic CurVe 4-80
- CLOSED TOOLPATH
 See Routines
- CLRLVB
 CLeaR LeVnum Bit 3-44
- CONIC
 See Routines
- CONTOUR
 See Routines
- Conversion Routines
 See Routines - MATHEMATICS
- CONVLO
 CONVert to LOwercase 3-118
- CONVUP
 COnVert to UPpercase 3-119
- Coordinate Systems 1-1
- Copy
 Attributes 3-49
 Entities 1-2
- CORNER
 See Routines
- CPTPCS
 Closest pt to iso-parametric curve surf.
 4-104
- CREUCS
 CREate a new UCS 4-147
- CRUCS3
 CReate UCS using 3 points 4-147
- CRVASO
 dimens CuRVe, create geom entity 5-25
- CRVASP
 dimens CuRVe, create geom ent. inter.
 5-25
- CURRENT TOOL
 See Routines
- CVDELP
 CurVe DElete Part 4-138
- CVI2T
 ConVert Integer 2 (to) Text 3-60
- CVR2CV
 ConVert Real 2 (to) Char Var 3-60
- CVR2TS
 ConVert Real 2 (to) Text String 3-61
- CVT2I
 ConVert Text 2 (to) Integer 3-61
- CVT2R
 ConVert Text 2 (to) Real 3-61

D

DBCLEA		
DataBase CLEA	3-52	
DBDSBX		
Get/set enclosing box of surface	4-104	
DBFGET		
DBFlag GET	3-52	
DBFLOF		
DBFLag Off	3-52	
DBFLON		
DBFLag ON	3-53	
DBLNK		
Display BLanNKs	3-25	
DEFADD		
DEFent ADD	3-53	
DEFPLN		
DEFine a PLaNe of a UCS	4-148	
DEGRD		
Convert radians to DEGREEs (Double)	3-62	
DEGREE		
Convert radians to DEGREEs	3-62	
Delete		
Master	4-24	
Record	3-13	
DELUCS		
DELeTe a UCS	4-148	
DHATCH		
Do HATCH	5-32	
DIAN1A		
DImens ANgle / 1 line and axis - Assoc.	5-3	
DIAN2A		
DImens ANgle by 2 lines - Assoc.	5-4	
DIAN3A		
DImens ANgle by 3 points - Assoc.	5-5	
DIANEX		
DImens ANgle EXtract	5-6	
DIANG1		
DImens ANgle by 1 line and axis	5-7	

DIANG2		
DImens ANgle by 2 lines	5-8	
DIANG3		
DImens ANgle by 3 points	5-8	
DIANPR		
DImens ANgle PaRameters	5-9	
DIARPA		
DImens AngulaR PaRameters	5-10	
DICHEX		
DImention CHAmfer EXtract	5-12	
DICHMA		
DImention CHaMfer	5-12	
DICHPA		
DImens CHAmfer sPecific modAls	5-13	
DIEXTR		
DImention EXTRact	5-14	
DIGLEX		
DImention ent. GLobal modal EXtract	5-15	
DIGLMD		
Accept DImentions GLobal MoDals	5-16	
DIGLPA		
DImention ent. GLobal PaRameters set	5-17	
DIGLPR		
Set DImentions GLobal PaRameters	5-22	
DIGSNA		
DImentions Get Standards Name	5-22	
DILINA		
DImens LiNeAr entity	5-40	
DILNEX		
DImens LiNear EXtract	5-41	
DILNPA		
DImens LiNear PaRameters	5-41	
DILNPN		
DImens LiNear Parameters	5-42	
DILNPR		
DImens LiNear PaRameters	5-43	
DIMENSION PARAMETERS		
See Routines		

DIMLIN		
DIMensions LINear	5-43	
DIMODI		
Dimension MODIfy	5-23	
DIMORD		
Dimensions ORDinate	5-53	
DIMRAD		
DIMensions RADial	5-57	
DIORDA		
Dimension ORDina	5-54	
DIOREX		
Dimension ORDina modal EXtract	5-54	
DIORPA		
Dimension ORDina sPecific modAls	5-55	
DIORPR		
Dimensions ORDinate PaRameters	5-56	
DIRDEX		
Dimension lineaR: modal EXtract	5-58	
DIRDPA		
Dimension ciRcular: sPecific modAl	5-58	
DIRDPR		
Dimension RaDial PaRameters	5-59	
Directories	1-4	
DISAXP		
DISTance between AXis and Point	3-86	
DISPOF		
DISPlay OFF	3-26	
DISPON		
DISPlay ON	3-26	
DIST2		
DISTance in 2D	3-86	
DIST3		
DISTance in 3D	3-86	
DISTD		
DISTance (Double)	3-87	
DIVCRV		
DIVide CuRVe	4-139	
DLDEDO		
Delete entity from database	3-53	
DLDELE		
Database List DELEte	3-54	
DLDELE1		
Database List DELEte	3-54	
DLDISP		
Dispay an entity	3-54	
DLDISPW		
Display entity ID in MODE	3-55	
DLFORC		
DeLete FORCe	3-55	
DMATT		
Dynamic Menu ATTention	3-26	
DMDEF		
Dynamic Menu DEFinition	3-26	
DMDEFN		
Dynamic Menu DEFinition with optioN	3-27	
DMINP		
Dynamic Menu Immed. fuNcs POSSible	3-27	
DMINPT		
Dynamic Menu Immed. fuNct.	3-27	
DMINZ		
Dynamic Menu INitialiZation	3-28	
DMINZ5		
Dynamic Menu InitialiZation for 5 lines	3-28	
DMSHOW		
Dynamic Menu SHOW	3-28	
DOCHPN		
Change pathnames	3-55	
DOCHPN1		
Change wireframe pathnames	3-55	
DOCHPN2		
Change solid pathnames	3-56	
DPIXEL		
Display a PIXEL	3-28	
DRPARL		
DRive PARaLell	4-105	

Index

DSPGET
 DiSPlay GET 3-56

DSPLEV
 DiSPlay all entities of LEVeL 3-44

DTEXT
 Display TEXT 3-29

E

EDMSLink 9-1

EGATTR
 Entity, Get ATTRibute 3-18

EGBLNK
 Entity, Get BLaNK 3-18

EGLEVL
 Entity, Get LEVeL 3-19

EGNGD
 Entity, Get NGD 3-19

EGTRAN
 Entity, Get the matrix TRANsformation
 3-19

Elementary Arithmetic
 See Routines - MATHEMATICS

ENA3DC
 ENAbLe picking of 3D Curves 3-94

ENAALL
 ENAbLe ALL 3-94

ENAARC
 ENAbLe ARC picking 3-94

ENABLE
 Enable class/type list selection 3-95

ENACRV
 ENAbLe CuRVe picking 3-95

ENAINS
 ENAbLe INSTANCES picking 3-95

ENALIN
 ENAbLe LiNe picking 3-95

ENAPLF
 ENAbLe PLanar Face picking 3-95

ENAPT
 ENAbLe PoinT picking 3-96

ENASAT
 ENAbLe Surface picking 3-96

ENAUCS
 ENAbLe UCS picking 3-96

ENCHAI
 ENTity CHange Attributes 1st 3-48

ENCHAT
 ENTity CHange ATtributes 3-49

ENINST
 ENTity INSTance-Double 4-39

ENMILID
 ENTity inst. MIRROR about Line-Double 4-39

ENMIPLD
 ENTity inst. MIRROR about PLane-Double
 4-39

ENMIPOD
 ENTity inst. Mirror about POint-Double 4-40

ENMUIND
 ENTity - MUltiple INSTANCES-Double 4-40

ENOFFS
 ENTity instance by OFFSet 4-41

ENSCAID
 ENTity inst. SCALing-Double 4-42

ENSHIFD
 ENTity instance by SHIFting-Double 4-42

ENSWLR
 ENTity: SWEEP LineAR 4-129

ENSWRT
 ENTity: SWEEP RoTate 4-129

ENTATT
 See Routines

Entities
 Deletion 3-51
 Duplication of 1-1
 ID 1-1
 Permanent 1-3,3-51
 Temporary 1-3,3-51

EQMAT
 Equal Matrices in 3D 3-64

EQMATD
 Equal MATrices in 3D (Double) 3-65

EQPT2
 Equal PoinTs in 2D 3-65

EQPT3
 Equal PoinTs in 3D 3-65

EQPTD
 Equal PoinTs (Double) 3-66

EQPTDD
 Equal PoinTs Double (Double) 3-66

EQV0DD
 Equal Value = 0 Double (Double) 3-67

EQVAL
 Equal VALues 3-67

EQVAL0
 Equal VALues = 0 3-67

EQVALD
 Equal VALues (Double) 3-68

EQVL0D
 Equal VALues = 0 (Double) 3-68

EQVLDD
 Equal VaLues Double (Double) 3-69

EUREAD
 External User READ a part file 8-2

EUSAVE
 External User SAVE a part file 8-2

EUSYS
 External User Package 8-1

EVISM
 Get/set visibility mask from/to entity ID 3-20

EXPLODE
 See Routines

External User Package
 EUSYS 8-1

F

Fillet 4-20

Function 1-3
 USER 1-4

G

GACUCS
 Get ACtive UCS 4-148

Geometrical Routines
 See Routines - MATHEMATICS

GEOMETRY
 See Routines

GETLVB
 GET LeVnum Bit 3-44

GETPD
 GET coordinates of Point (Double)
 3-97

GETPT
 GET coordinates of PoinT 3-98

GETTPTEXT
 Get the toolpath text 6-46

GFCHMF
 Geometrical Function make CHaMFer
 4-18

GFCORN
 Geometrical Function make CORNer
 4-19

GFDIVD
 Geometr. Funct. DIViDe 4-139

GFDVPT
 Geometr. Funct. DiVide by PoinT
 4-140

GFFILT
 Geometrical Function make FILleT
 4-20

GFSPIS
 Gen. Funct. Sect. Property ISland 4-2

GFSPOC
 Gen. Funct. Sect. Property Outer Cont.
 4-2

Index

GFSPUT	
Gen. Funct. Sect. Property outpUT	4-3
GFTRIM	
Geometr. Funct. TRIM	4-140
GFTRPL	
Geometr. Funct. TRim PLane	4-141
GFTRPT	
Geometr. Funct. TRim PoinT	4-141
GINPTD	
Get coordINates of PoinT Displayed	3-98
GLLVD1	
GLobal LeVel Define, not active	3-45
GLLVDF	
GLobal LeVeL DeFine, active	3-45
GLLVSA	
GLobal LeveL Set Active	3-45
Global Parameters	5-14,5-40,5-53,5-57
GLOBAL PARAMS	
See Routines	
GRCLCR	
GRoup Collection List CReate	4-22
GRCLOS	
GRoup CLOSe collecting entities	4-23
GRCOPN	
GRoup Collect entities OPeN	4-23
GRDLUN	
GRoup DeLete UNamed master	4-24
GREXPL	
GRoup EXPLode	4-21
GRIDEM	
GRoup Find ID Ext. Master	4-24
GRIDES	
GRoup Find ID of Ex. Sub-view	4-25
GRIDIM	
GRoup Find ID of Int. Master	4-25
GRIDIS	
GRoup Find ID of Int. Sub-view	4-26
GRIDSA	
GRoup Find ID of int. Sub-Assembly	4-26
GRIN3P	
GRoup INSTance by 3 Points	4-43
GRINRO	
GRoup INSTance by ROtate	4-44
GRINST	
GRoup INSTance	4-44
GROUP	
See Routines	
GSCLR	
Get/Set CLeaR status	3-29
GSMSLA	
Get/Set Level MaSk	3-49
GTABID	
Get TABle ID	3-5
GTABNA	
Get TABle No. and nAme	3-6
GTATTR	
Get ATTRIBUTES	3-6
GTCRPT	
GeT CoorDinates of a PoinT	4-80
GTCRPTD	
GeT CoorDinates of a PoinT	4-81
GTFLNA	
GeT FieLd NAMES	3-7
GTFLV1	
GeT FieLd VaLues 1	3-7
GTFLVL	
GeT FieLd VaLues	3-8
GTGEOM	
Get GEOMetric entity	3-8
GTLANG	
GeT translated version	3-109
GTMACH	
GeT MACHine type	3-109
GTPLMA	
GeT PLace MATrix	4-130
GTRCNU	
GeT ReCord NUMber	3-9

GTTYPE
 GeT field TYPE 3-9
 GUPDAT
 Read User Input during exec. 3-29

H
 HATCH
 See Routines

I
 ID
 Entity 1-1
 ID2NAM
 UCS ID 2(to) NAME 4-149
 IDNUM
 See Routines
 IGTABID
 Get TABLE ID 3-10
 IGTATTR
 Get TABLE No. and nAme 3-10
 IGTGEOM
 Retrieve ID numbers of geom entities 3-11
 IMMAWA
 set IMMEDIATE Auto Window 3-56
 IND2C
 calc. INTERsection point of 2 Circles 3-87
 init_path_chg
 Initialize no_of_changes parameter 3-56
 INPOLD
 point IN POLYgon - closed 3-88
 Instances 4-43
 INT2C
 INTERsect 2 Circles 3-88
 INTERACTION
 See Routines

Intersection 3-88,4-19,4-55

K

Keyboard Input 3-39

L

LABEL

 See Routines

LASTID

 LAST used ID 3-110

LENACT

 LENGth of string - ACTual 3-119

LENINT

 LENGth of INTeger 3-119

LEVELS

 See Routines

LEVSET

 LEVEL SET 3-46

LINATT

 See Routines

LINE

 See Routines

Line Attributes 3-48

LN2ENT

 LiNe by 2 ENTities 4-28

LN2ENTD

 LiNe by 2 ENTities-Double 4-29

LN2PNT

 LiNe by 2 PoiNTs 4-29

LN2PNTD

 LiNe by 2 PoiNTs-Double 4-30

LNCVCV

 LiNe by CurVe and CurVe 4-30

LNIPVE

 LiNe by ID Point and VECTOR 4-31

LNIPVED

 LiNe by ID Point and VECTOR-Double
4-31

Index

LNLINC
 LiNe by LiNe and Curve 4-32

LNOFFS
 LiNe as OFFSet 4-32

LNPOEN
 LiNe POLygon by ENtities 4-33

LNPOEND
 LiNe POLygon by ENtities-Double 4-33

LNPTCV
 LiNe by PoinT and CurVe 4-34

LNPTVE
 LiNe by PoinT and VECtor 4-34

LNPTVED
 LiNe by PoinT and VECtor-Double 4-35

LSQRPD
 Find the least square plane 3-89

M

M3CROS
 Matrix in 3D CROsS product 3-69

M3CROT
 Matrix 3D CROs Transposed 3-69

M3CRSD
 Matrix 3D CROsS Double 3-70

M3CRTD
 Matrix 3D CRoss Transposed Double 3-70

M3DET
 Matrix 3D DETerminant 3-70

M3DETD
 Matrix 3D DETerminant Double 3-71

M3INV
 Matrix 3x3 INVerse 3-71

M3INVD
 Matrix 3x3 INVerse Double 3-71

M3LNED
 Matrix 3 LiNear Eq. Double 3-72

M3LNEQ
 Matrix 3 LiNear EQUations 3-72

M3VEC
 Multiply 3d VECtor by matrix 3-73

M3VECD
 Multiply 3d VECtor by matrix Double 3-73

MACHINE PARAMS
 See Routines

MACSYS
 See Routines

MADM2W
 MAP Model 2 (to) Work-Double 4-130

MADU2M
 Map point from UCS to MODEL system
 4-149

MADUCS
 Map point from UCS to MODEL or WORK
 system 4-149

MADW2M
 MAP Work 2 (to) Model-Double 4-131

MANAGEMENT
 See Routines

MAPM2U
 MAP point from Model 2(to) UCS 4-150

MAPM2W
 MAP Model 2 (to) Work 4-131

MAPU2M
 MAP point from UCS 2(to) Model 4-150

MAPU2W
 MAP point from UCS 2(to) Work 4-150

MAPW2M
 MAP Work 2 (to) Model 4-131

MAPW2U
 MAP Work 2(to) UCS 4-151

MARKERS
 See Routines

MATHEMATICS
 See Routines

Matrix 3-69,3-72,3-76
 Transformation 4-36,4-130

MENU
 Display MENU 3-30

Menu List 1-4

MESSAGE
See Routines

MODCOM
MODE of COMplement of modal 3-30

MODDIM
MODal table DIMensions 3-30

Model Coordinate System 1-1

MODFX
MODals Fixed value 3-31

MODFX2
New MODal parameter 3-31

MODHIX
Length of the MODals area 3-32

MODI
MODals Integer value 3-32

MODI2
New MODal parameter 3-32

MODIF1
MODal Interaction 3-33

MODIFY
MODals modIFY 3-33

MODINT
MODal parameter INTeraction 3-33

MODINZ
MODals INItialIZatIon 3-34

MODISP
MODal parameter DISPlay 3-34

MODM
MODals Menu (scrolling items) 3-34

MODM2
MODals Menu (scrolling items) 3-35

MODMP2
MODals Menu (scrolling items) 3-35

MODMPL
MODals Menu (scrolling items) 3-36

MODR
MODals Real value (USER units) 3-36

MODR2
MODals Real value (USER units) 3-37

Moment of Inertia 4-2

Mouse Response 1-3

MOVE
See Routines

MPICKP
Multiple PICKing 3-99

N

NAM2ID
UCS NAME 2(to) ID 4-151

NEW MACHINE PARAMS
See Routines

NGCPAT
NGd reCord ATtach 3-11

NOTE
See Routines

O

OB2DMD
OBtain 2D/3D MoDe, 3-110

OBACAT
OBtAin Current ATtributes, 3-110

OBACME
OBtAin Current MEasurement unit, 3-110

OBACRT
OBtAin Current RooT directory, 3-111

OBATTR
OBtain ATTRibute 3-20

OBCRCL
OBtain arc/CiRCLe, 4-152

OBDRVW
OBtain DRawing VieWs 5-68

OBDVID
OBtain Drawing/View ID 5-68

Index

OBDVLC
 Obtain Drawing/View/Level Current 5-69

OBDVLL
 Obtain Drawings/Views/Levels 5-69

OBDVLN
 Obtain Drawing/View/Level Number 5-70

OBGROU
 Obtain GROUp data 4-153

OBINST
 Obtain INSTance, 4-153

OBLINE
 Obtain LINE data 4-154

OBPNT
 Obtain PoiNT data 4-154

OBTPNM
 Obtain ToolPath Name 6-45

OBTUCS
 ObTain information on UCS, 4-154

OBVWLS
 Obtain VieWs LiSt 5-70

OBVWTR
 Obtain VieW # TRansformation 5-71

OPEN TOOLPATH
 See Routines

OUTCB
 OUTput Character screen 3-37

OUTCGB
 OUTput Char strinG on screen 3-38

P

PARABO
 PARABola 4-12

Permanent Entity 1-3,3-51

PICK
 Put picked entity into Select list 3-99
 See Routines

PIFACT
 PI multiply by FACTor 3-73

PKATRB
 Pack/Unpack ATTRIButes 3-50

PLACE
 See Routines

PLANAR FACE
 See Routines

PLCOF
 Plane COefficients 3-89

PLCOFD
 Plane COefficients (Double) 3-89

PLEXGR
 PLace EXternal GROup 4-45

PNTEX3
 PoiNT at intersection EXtensions 4-51

PNTIN3
 PoiNT at INtersection 4-51

POINT
 See Routines

PRCRSR
 PRoject a CuRve on a SuRface 4-67

PRDRSR
 PRoject point at DiRection on SuRface
 4-105

PREREC
 rePREsent RECord 3-12

PRJDR1
 PRoJection XY pl. by DiRection 4-67

PRJPD1
 PRoJect PerpenDicular XY of UCS 4-68

PRNRSR
 PRoject point direct. Normal to SuR. 4-106

Procedure
 Compiling & Linking 2-2,2-3

PROCEDURE MANAGEMENT
 See Routines

PROJDR
 PROJection by DiRection 4-69

PROJECT
 See Routines

PROJPD
 PROJection PerpenDic. to work pl. 4-70

PROMPT
 Display text in PROMPT field 3-38

Prompt Area 3-38

PTASCE
 geom PoinT for CEnter option 5-25

PTASEN
 geom PoinT for ENd option 5-26

PTASID
 geom PoinT Interactive 5-26

PTASIN
 geom PoinT for INtersection option 5-27

PTASMI
 geom PoinT for MIddle option 5-27

PTASPI
 geom PoinT for PIch option 5-28

PTCDIS
 PoinT on Curve by DIStance 4-52

PTCENT
 PoinT at CENTer of a curve 4-52

PTCOMO
 Point by COordinates - - Model 4-53

PTCOWO
 PoinT by COordinates - - WOrk 4-53

PTEND
 PoinT at END of a curve 4-53

PTEXIN
 PoinT at EXtended INtersection 4-54

PTGET
 PoinT GET 4-54

PTINCR
 PoinTs at INtersection of 2 CuRves 4-55

PTINTR
 PoinT at INTeRsection 4-55

PTM2PT
 PoinT at Middle of 2 PoinTs 4-56

PTMIDD
 PoinT at MIDDle of a curve 4-57

PUTLVB
 PUT LeVnum Bit 3-46

R

RADD
 Convert degrees to RADians (Double) 3-62

RADIAN
 Convert degrees to RADIANS 3-63

RECCRE
 RECORD CREate 3-12

RECDEL
 RECORD DELeTe 3-13

RECUPD
 RECORD UPDate 3-13

REDRAW
 REbuild Display file, Repaint display Window 3-57

REFERENCE GEOMETRY
 See Routines

REPLY
 Display the prompt REPLY (YES or NO) 3-38

RETYIN
 Retrieve total number of records in table 3-14

REVOLS
 REVOLution Surface 4-106

Routines
 ANALYZE - Modeling 4-2
 ANGULAR DIMENSIONS - Drafting 5-2
 ASSEMBLY - Solid 7-35
 ATTRIB - General 3-2
 CENTER LINES - Drafting 5-11
 CHAMFER - Drafting 5-12
 CIRCLE - Modeling 4-4
 CLOSED TOOLPATH - NC 6-45
 CONIC - Modeling 4-10
 CONTOUR - Modeling 4-13

Index

CORNER - Modeling	4-18	RADIAL DIMENSIONS - Drafting	5-57
CURRENT TOOL - NC	6-17	REFERENCE GEOMETRY - SOLID	7-10
DIMENSION PARAMETERS - Drafting	5-14	REPORTS - General	3-107
DISPLAY - General	3-15	SKETCHER - SOLID	7-4
ENTATT - General	3-18	SOLID ATTRIBUTES	7-37
EXPLODE - Modeling	4-21	SOLID DATA BASE ACCESS - SOLID	7-15
GEOMETRY - Drafting	5-24	SOLID DISPLAY	7-42
GEOTOL - Drafting	5-29	SOLID EDIT	7-47
GLOBAL PARAMS - NC	6-5	SOLID OPERATIONS - SOLID	7-11
GROUP - Modeling	4-22	SOLID PICK FUNCTIONS	7-31
HATCH - Drafting	5-32	SOLID PROCEDURE DATA ACCESS -	
IDNUM - Drafting	5-33	SOILID	7-28
INTERACTION - General	3-22	SPLINE - Modeling	4-81
LABEL - Drafting	5-34	START PROGRAM - NC	6-3
LEVELS - General	3-43	STATUS - General	3-108
LINATT - General	3-48	STRING - General	3-118
LINE - Modeling	4-28	STYLE - Drafting	5-60
LINEAR DIMENSIONS - Drafting	5-40	SURFACE - Modeling	4-88
MACHINE PARAMS - NC	6-10	SWEEP - Modeling	4-128
MACSYS - NC	6-50	TEMPLATES - NC	6-54
MANAGEMENT - General	3-51	TOOL - NC	6-40
MARKERS - NC	6-37	TOOL MOTIONS - NC	6-30
MATHEMATICS - General	3-60	TOOL POSITION - NC	6-9
MATHS Conversion	3-60	TOOLPATH - NC	6-46
MATHS Elementary Arithmetic	3-64	TRANSFORMATION - Modeling	4-130,4-131,4-132,4-133,4-134,4-135,4-136,4-137
MATHS Geometrical	3-85	TRANSLATE WF -> SOLID	7-53
MESSAGE - NC	6-39	TRIM - Modeling	4-138
MOVE - Modeling	4-36	UCS - Modeling	4-143
NEW MACHINE PARAMS - NC	6-33	UTILITIES - SOLID	7-24
NOTE - Drafting	5-44	VERIFY - Modeling	4-152
OPEN TOOLPATH - NC	6-4	VIEWS - Drafting	5-65
ORDINATE DIMENSIONS - Drafting	5-53	WORKPL - Modeling	4-159
PICK - General	3-92		
PLACE - Modeling	4-43	RTEXT	
PLANAR FACE - Modeling	4-46	Read TEXT from keyboard	3-39
PLOT - General	3-104	RULEDS	
POINT - Modeling	4-51	RULED Surface	4-107
PROCEDURE MANAGEMENT - NC	6-51		
PROJECT - Modeling	4-64		

S

Scale

- Factor 4-136
- Vector 3-74,3-77,3-80

Screen areas

- Character String 3-37,3-38
- Interaction 3-22
- Prompt 3-22,3-38

SEGABSET

- Kill user prog. upon exit 3-57

SELARR

- SElect displayed ARRow direction 3-99

Selection Mask (SM) 3-92

SETCR

- SET Carriage Return 3-57

SETGC

- SET strinG start pos on sScreen 3-39

SETIR

- SET Immediate Return 3-39

SETLVB

- SET LeVnum Bit 3-46

SETMF

- SET Master Flag 3-100

SETNCR

- SET Not Carriage Return 3-57

SGDCLR

- Set Given pixel Display CoLoR 3-40

SKETCHER

- See Routines

SLCLR

- SeLection list CLear 3-100

SLLN

- Return # of entities in selection list 3-100

SLPOP

- SeLection list POP entry 3-100

SLPOP3

- POP selection list entry, return XYZ Co 3-101

SLPSH3

- SeLection list PuSH entry Into 3-101

SLPSHI

- SeLection list PuSH entry Into 3-101

SLRJCT

- SeLection list ReJeCT entry 3-101

SLTOP3

- Selection list top entry, return XYZ Co 3-102

SOLID ATTRIBUTES

- See Routines

SOLID DATA BASE ACCESS

- See Routines

SOLID DISPLAY

- See Routines

SOLID EDIT

- See Routines

SOLID OPERATIONS

- See Routines

SOLID PICK FUNCTIONS

- See Routines

SOLID PROCEDURE DATA ACCESS

- See Routines

SPLINE

- See Routines

SPLNCR

- SPLine CReate 4-81

SRFPTD

- Surf./trimmed surf. point ind. params. 4-108

STACLA

- SeT ACtive Line Attributes 3-50

START PROGRAM

- See Routines

STATUS

- See Routines

STCOMP

- STring COMPare 3-120

STDNID

- STanDard Number ID 5-23

STRING

- See Routines

STYLE

See Routines

SURFACE

See Routines

SWEEP

See Routines

T

TEMPLATES

See Routines

Temporary Entity 1-3

TOOL

See Routines

TOOL MOTIONS

See Routines

Tool Path

6-3,6-4,6-5,6-9,6-10,6-17,6-30,6-33,6-37,6-39,6-40,6-45,6-46,6-50,6-51,6-54

TOOL POSITION

See Routines

TOOLPATH

See Routines

TOPOPT

TOP OPTion pull down menu 3-40

TPLCLR

TemPorary List CLear 3-58

TR3PTS

TRansformation by 3 PoinTs 4-132

TRANSFORMATION

See Routines

Transformation Matrix 4-36,4-130

TRANSLATE WF -> SOLID

See Routines

TRCONC

TRansformation by CONCatenation 4-132

TRCONM

TRansformation by CONCatenation 4-132

TRCONMD

TRansformation by CONCatenation-Double 4-133

TRIM

See Routines

TRMCRV

TRiM a CuRVe 4-142

TRMILI

TRansform. MIrror about LIne 4-133

TRMILID

TRansform. MIrror about LIne-Double 4-133

TRMIPL

TRansform.Mirror about PLane 4-134

TRMIPLD

TRansform.Mirror about PLane-Double 4-134

TRMIPO

TRansform. MIrror about POfnt 4-134

TRMIPOD

TRansform. MIrror about POfnt-Double 4-135

TRROAX

TRansform. ROTation about AXis 4-135

TRROAXD

TRansform. ROTation about AXis-Double 4-135

TRSCA1

TRansformation by SCALing 4-136

TRSCA1D

TRansformation by SCALing-Double 4-136

TRSGID

Get trimmed surface ID 4-109

TRSGSZ

TRimmed Surface Get SiZe 4-109

TRSHIF

TRansformation by SHIFting 4-136

TRSHIFD

TRansformation by SHIFting-Double 4-137

TSD_BOX

Get/set parametric/geometric enclosing box 4-109

Typographical Conventions 1-3

U

U2DMOD

User 2D MODe 3-111

UAXI_INDEX_DATA

Get assembly tree components 3-111

UAXI_NUM_ASM_COMPS

Get # of assembled tree components 3-111

UBLANK

User BLANK unblank entity 3-58

UBSPLN

User B-SPLiNe create 4-82

UBSSPW

Get/set list of control Pts & Weights 4-110

UBSURF

User create B-SURFace 4-111

UBZSRF

User create BeZier SuRFace 4-111

UCOMCR

User create COMposite CuRve 4-14

UCOMSP

User approx COMposite SPline 4-15

UCR2PS

User CReate 2 sPineS 4-113

UCRBCM

User CReate BiCubic Mesh 4-112

UCRBSP

User CReate Bezier SPline 4-83

UCRELP

User CReate ELlipse 4-12

UCRPIP

User CReate bez. drive/sPine, sp.&Pl 4-114

UCRVLN

calcUlate CuRve LeNgth 4-84

UCS

See Routines

UCSMAT

UCS MATrix 4-151

UDBLEV

Look in DB for entities in given level
3-47

UDFDC

User DeFineD Closed contour buf 4-15

UDFDO

User DeFineD Open contour buf 4-16

UDRAG

User DRAG an entity 4-42

UEVCRV

calcUlatE co/deriV. of pt. on CuRve
4-84

UEVISM

User gEt VISibility Mask 3-21

UEVSRD

calcUlatE co/direV. of SuRf pt - D.P.
4-115

UEXPLD

User EXPLoDe 4-21

UFILLC

User create FILLet with Constant rad.
4-116

UFILLV

User create FILLet with Variable rad.
4-117

UGETPNT

User PICK 3-102

UGNRBS

Get properties of NURBS 4-117

UGPARM

User Get curve PARaMeter 4-85

UGPARS

User create a PARametric Spline 4-118

UGSBZP

User Get/Set points on BeZier mesh srf.
4-118

UGSPRM

Retreive surface parameters 4-119

UGSSDC

Get/Set no. curves to Display Surf.
4-119

Index

UGSSRF	
User - Get/Set Spline Refine Factor	4-85
UGTALE	
User GeT Active LElvel	3-47
UGTAPL	
User GeT APpLication number,	3-112
UGTDTM	
User GeT current Date and TiMe	3-112
UGTERF	
User GeT list of External Refs	3-112
UGTMNM	
User GeT Master NaMe,	4-155
UGTNID	
User GeT Node ID	3-113
UGTNMERF	
Get no. of ext refs in .pfm file	3-58
UGTNTP	
User GeT New ToolPath id	6-45
UGTPFN	
User GeT Part File Name	3-113
UGTPNM	
User GeT Path NaMe	3-114
UGTRPO	
User Get TRimmed surf. POlygon data	4-120
UGTSNM	
Get name of system file/dir	3-114
UGTVNM	
User GeT current View NuMber	3-115
UGVNAM	
User Get View NAME and type	3-115
UHLIGHT	
Highlight instance	3-40
ULMCRD	
User - LiMits of CuRve from Database	4-85
ULMSRF	
User - LiMits of SuRFace	4-120
UMADEL	
User MAster DElete	4-27
UMDCLR	
User get Menu Default CoLoR	3-40
UMDGET	
User MoDal parameter GET	3-58
UMDPUT	
User MoDal parameter PUT	3-59
UMODS	
User - define MODal parameter String	3-41
UMPICK	
User PICK	3-102
UNC_APPEND_PCT2TPT	
Appends a PCT file to a TPT file	6-54
UNC_BYVC	
User NC-set to BY VC or spin	6-33
UNC_CIR5	
User NC-create a CIRcular 5 axis motion	6-30
UNC_CIRC	
User NC-create a CIRCular motion	6-31
UNC_CIRS	
User NC-create a CIRCular 3 axiS motion	6-31
UNC_CLER	
User NC-current CLEaR value	6-5
UNC_CMPNS	
Send the compensation block to the machine	6-5
UNC_COOL	
User NC-set new COOL value	6-33
UNC_FAST	
User NC-set to FAST motion	6-34
UNC_FEED	
User NC-set to new FEED value	6-34
UNC_HOME	
User NC-toolpath HOME pos	6-4
UNC_ID_TDRL	
Get tool params by ID - drill tool	6-40
UNC_ID_TGRV	
Get tool params by ID - groove tool	6-40

UNC_ID_TLAT	
Get tool params by ID - lathe tool	6-41
UNC_ID_TMIL	
Get tool params by ID - mill tool	6-41
UNC_ID_TPUN	
Get tool params by ID - punch tool	6-42
UNC_ID_TTHR	
Get tool params by ID - thread tool	6-42
UNC_LATH	
User NC-current mach params LATHe tp	6-10
UNC_LIN5	
User NC-create a LINear 5 axis motion	6-32
UNC_LINE	
User NC-create a LINEar motion	6-32
UNC_LINS	
User NC-create a LINear 3 axiS motion	6-32
UNC_MACS_ACT	
Activate the defined MACSYS	6-50
UNC_MACS_DEL	
Delete a MACSYS	6-50
UNC_MACS_NEW	
Define a new MACSYS	6-50
UNC_MAPP	
User NC-Machining APProach marker	6-37
UNC_MILL	
User NC-current mach params MILL tp	6-10
UNC_MLAY	
User NC-Machining LAYer marker	6-37
UNC_MORG	
User NC-current ORiGin	6-5
UNC_MPAS	
User NC-Machining PASs marker	6-38
UNC_MRTR	
User NC-Machining ReTRact marker	6-38
UNC_MSG	
User NC-enter a MeSsaGe	6-39
UNC_MTyp	
User NC-current Machine TYPE	6-4
UNC_PONF	
User NC-set to Punch ON/off	6-34
UNC_PRC_BOTT	
Update the bottom menu	6-11
UNC_PRC_CFD	
Get/set corner feed value for Mill/Lathe tp	6-11
UNC_PRC_CHORG	
Change origin	6-6
UNC_PRC_CLON	
Get/set clear ON/OFF value	6-6
UNC_PRC_CLOSE	
Close a procedure	6-51
UNC_PRC_COOL	
Get/set cool value for Mill/Lathe tp	6-12
UNC_PRC_DEL	
Delete a procedure	6-51
UNC_PRC_DIA_COMP	
Get/Set the Diameter Compensation ON/OFF value	6-6
UNC_PRC_DSTL	
Get/set the tool display flag	6-7
UNC_PRC_DWFD	
Get/set down feed value for Mill/Lathe	6-12
UNC_PRC_FST	
Get/set feed type value for Mill/Lathe tp	6-13
UNC_PRC_GCLA	
Get/set absolute internal clear value	6-7
UNC_PRC_GCLI	
Get/set incremental internal clear value	6-8
UNC_PRC_GSCL	
Get/set the clear value	6-8
UNC_PRC_INAB	
Get/set internal clear mode	6-8
UNC_PRC_INIT	
Initialize a procedure	6-51

Index

UNC_PRC_INTER	
Do procedure after running UNC_PRC_INT	
6-52	
UNC_PRC_NIB	
Get/set NIBBLING val. for Punch tp	6-13
UNC_PRC_NUM	
Obtain # of procedures in TP	6-52
UNC_PRC_OPEN	
Open a procedure	6-52
UNC_PRC_PFD	
Get/set plunge feed value for Mill/Lathe tp	
6-14	
UNC_PRC_PUN	
Get/set PUNCH value for Punch tp	6-14
UNC_PRC_RFD	
Get/set feed value for Mill/Lathe tp	6-14
UNC_PRC_SDFD	
Get/set side feed value for Mill/Lathe tp	
6-15	
UNC_PRC_SPD	
Get/set spindle value	6-15
UNC_PRC_SPN	
Get/set spin value	6-15
UNC_PRC_TXT	
Get/set the text of a procedure	6-53
UNC_PRC_VCS	
Get/set Vc/Spin value for Lathe tp	6-16
UNC_PRC_VCV	
Get/set Vc value	6-16
UNC_PUNC	
User NC-current mach params PUNCH tp	
6-11	
UNC_RFED	
User NC-set to FEED motion	6-35
UNC_SPIN	
User NC-set to new SPIN value	6-35
UNC_SPND	
User NC-set to new SPINDle direction	6-35
UNC_STR	
User NC-STaRt	6-3
UNC_TAGI	
User NC-params of Tool: AGIe	6-17
UNC_TCHR	
User NC-params of Tool: CHaRmille	6-18
UNC_TDRL	
User NC-params of Tool: DRiLl	6-18
UNC_TGRV	
User NC-params of Tool: GRooVe	6-19
UNC_THLD	
User NC-Tool HoLDer number	6-19
UNC_TL_ATT	
Return tool attribs for given ID	6-20
UNC_TL_CAGI	
Create/update AGIE tool	6-21
UNC_TL_CDRL	
Create/update drill tool	6-21
UNC_TL_CGRV	
Create/update groove tool	6-22
UNC_TL_CLAT	
Create/update lathe tool	6-22
UNC_TL_CLDR	
Create/update lathe drill tool	6-23
UNC_TL_CMIL	
Create/update mill tool	6-23
UNC_TL_CTHR	
Create/update thread tool	6-24
UNC_TL_CUR	
Set the current tool	6-24
UNC_TL_DEL	
Delete the tool entity	6-25
UNC_TL_ID	
Return tool ID for given type & index	6-25
UNC_TL_NUM	
Return # of tools of a specific type	6-26
UNC_TL_SYM	
Set/unset user defined symbol	6-26
UNC_TLAT	
Get current tool params for lathe	6-20

UNC_TMIL		
User NC-params of Tool: MILl	6-27	
UNC_TMKN		
User NC-params of Tool: MaKiNo	6-27	
UNC_TMPL_RW		
Read/write params to/from PCT template file	6-54	
UNC_TMPL_TPT_RW		
Read/write param value in TPT file	6-55	
UNC_TP_CHK		
Check if the tool path exists	6-46	
UNC_TP_CLOSE		
Close a tool path	6-47	
UNC_TP_CREA		
Create a tool path	6-47	
UNC_TP_FLAGS		
Read the flags of a procedure in a Toolpath	6-48	
UNC_TP_IDCUR		
ID of currently open tool path	6-48	
UNC_TP_NUM		
# of tool paths in current MACSYS	6-48	
UNC_TP_PAR		
Tool path parameter list	6-49	
UNC_TP_REN		
Rename a tool path	6-49	
UNC_TP_REOP		
Reopen a tool path	6-49	
UNC_TPOS		
User NC-current Tool POSition	6-9	
UNC_TPT_PROC_NUM		
Returns # of PCT sections in TPT file	6-55	
UNC_TPTA		
User NC Tool Params, calc Tool tAn.	6-43	
UNC_TPTP		
User NC Tool Params, calc Tool tiP.	6-43	
UNC_TPUN		
User NC-params of Tool: PUNch	6-28	
UNC_TTAN		
User NC Tool tip, calc TAN pt.	6-44	
UNC_TTHR		
User NC-params of Tool: THRead	6-28	
UNC_TTIP		
User NC-given Tool tan, calc tool TIP	6-44	
UNC_TTYP		
User NC-current Tool TYPE	6-29	
UNC_TVEC		
User NC-current Tool axis	6-9	
UNC_VCVL		
User NC-set to new VC VaLue	6-36	
UNPBZS		
User No. of Patches in BeZier Surface	4-120	
UODCIR		
User Obt Double p. CIRle data	4-155	
UODCON		
User Obtain Double p. CONic data	4-156	
UODHLX		
User Obtain Double p. HeLiX data	4-156	
UODLIN		
User Obtain Double p. LiNe data	4-157	
UODPNT		
User Obtain Double p. PoiNT data	4-157	
UODPRJ		
User Obtain Double p. PRoJected	4-157	
UODSPL		
User Obtain Double p. SPLine data	4-158	
UPICK		
User PICK	3-103	
UPICKP		
User PICK and make Point	3-103	
UPLFTS		
User conv. PLanar Face to Trim Surf.	4-48	
UPNTCENT		
PoiNT at CENTer of curve	4-57	

Index

UPNTCM		
PoiNT by Coord. of Model	4-57	
UPNTCW		
PoiNT by Coord. of Work	4-58	
UPNTEND		
PoiNT by END of curve	4-58	
UPNTGET		
PoiNT interactive	4-59	
UPNTMID		
PoiNT by MID of curve	4-59	
UREVOLS		
Create surface of REVOLution	4-121	
USAVE		
User SAVE current file	3-116	
User Program	1-4	
USMKCB		
User MaKe Contour Buffer	4-17	
USPLNP		
User - SPLiNe control Points	4-86	
USRERR		
USeR ERRor message	3-41	
USRFBD		
User - get SuRFace BounDary	4-122	
USRFIN		
User Spline along suRFace INtersection	4-122	
USRFPT		
User parameters of SuRFace/PoinT	4-123	
USRMES		
USeR MESsage on screen	3-42	
USRMSG		
USeR MeSsaGe on screen & pad	3-42	
USRNOR		
User - pt coord SuRface NORmal	4-123	
USRSCL		
USeR SCaLe a surface	4-124	
USRSEC		
User spline on SuRface interSEction	4-124	
UTILITIES		
See Routines		
UTPFWP		
User - Trim Planar Face by Work Plane	4-49	
UTR3PNT		
calc. TRansformation matrix	4-137	
UTRMPF		
User - TRiM Planar Face by contours	4-49	
UTRSCO		
User TRim Surface by COntour	4-125	
UTRSPA		
User TRim a Surface by PArAmeter	4-125	
UTRSPC		
User TRim a Surf. by Projected Cont.	4-126	
UTRSPL		
User TRim a Surface by work PLane	4-126	
UTRSSR		
User TRim a Surface by a SuRface	4-127	
UVRBSR		
U/V contRol points of Bezier meSh	4-127	
UWP2CRV		
Work Plane define by 2 CuRVes	4-159	
UWP3PTID		
Work plane define 3 PoinTs	4-160	
UWPCOEF		
Work Plane define by 4 COEFs	4-160	
UWPDSPL		
Work Plane define ParalleL to screen	4-160	
UWPLNPT		
Work PLaNe define perpen. to line	4-161	
V		
V2ADD		
Vectors in 2D ADD	3-74	
V2ANGL		
Vectors in 2D ANGLE in between	3-90	
V2CROS		
Vectors in 2D CROSSs product	3-74	

V2CRSD		
Vectors in 2D CROSSs prod. (Double)	3-74	
V2DOT		
Vectors in 2D DOT product	3-75	
V2LNED		
Vector in 2D solve LiNear EQ. D.P.	3-75	
V2LNEQ		
Vectors in 2D solve LiNear EQ.	3-76	
V2PRP		
Vector in 2D PeRPendicular	3-90	
V2SCAL		
Vector in 2D SCALe	3-76	
V2SIZD		
Vector in 2D SIZE (Double)	3-76	
V2SIZE		
Vector in 2D SIZE	3-77	
V2SUB		
Vectors in 2D SUBtract	3-77	
V2UNIT		
Vectors in 2D UNIT value	3-77	
V2ZERD		
Vectors in 2D check if ZERO (Double)	3-78	
V2ZERO		
Vectors in 2D check if ZERO	3-78	
V3ADD		
Vectors in 3D ADD	3-78	
V3ANGL		
Vectors in 3D ANGLe in between	3-90	
V3CROS		
Vectors in 3D CROSSs product	3-79	
V3CRSD		
Vectors in 3D CROSSs product (Double)	3-79	
V3DOT		
Vectors in 3D DOT product	3-79	
V3LIN		
Vectors in 3D check if coLiNear	3-91	
V3SCAL		
Vectors in 3D SCALe	3-80	
V3SIZD		
Vectors in 3D SIZE (Double)	3-80	
V3SIZE		
Vectors in 3D SIZE	3-80	
V3SUB		
Vectors in 3D SUBtract	3-81	
V3UNIT		
Vectors in 3D UNIT value	3-81	
V3ZERD		
Vectors in 3D check if ZERO (Double)	3-81	
V3ZERO		
Vectors in 3D check if ZERO	3-82	
Variables		
SELECT	1-2	
STATUS	1-2	
VASDDD		
Add Scaled Vector to another Vector	3-82	
VDOTD		
Vectors DOT product (Double)	3-83	
VECSCL		
VECTor SCALe	3-83	
Vector	3-74,3-81	
VERIFY		
See Routines		
VIEWLM		
VIEW Level Mask	3-47	
VIEWS		
See Routines		
VPDRWN		
Set flag for View Port	3-116	
VPERSD		
Get flag for View Port	3-116	
VPGSDX		
Get/Set transformation data	3-117	
VPGVPN		
Get current view port number	3-117	
VPLDIS		
Get list of all displayed windows	3-117	

Index

VSUBD

Vector SUBtract (Double) 3-84

VUNITD

Vector to UNit vector (Double) 3-84

VWACTV

View ACTive 5-71

VWDFCD

View DeFine as Current Display 5-72

VWDFMP

View DeFine as Model Projection 5-72

VWEXIT

View EXIT 5-73

W

Work Coordinate System 1-1

Work Plane 1-1,4-141

WORKDF

WORK plane DeFine 4-161

WORKPL

See Routines

WPCRVS

Work Plane by CuRVeS, 4-162

WPGET

Obtain coeff. of wrk pln. - Float 4-162

WPLNPT

Work Plane by LiNe and PoinT 4-162

WSNMX1

Export master from PFM to another file
4-27

WVCHK

Work View CHecK 3-91