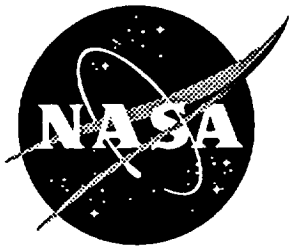
NASA Contractor Report 195075



A Generic Interface Element for COMET-AR

Susan L. McCleary
Lockheed Engineering & Sciences Company, Hampton, Virginia

Mohammad A. Aminpour
Analytical Services & Materials, Inc., Hampton, Virginia

Contracts NAS1-19000 and NAS1-19700

March 1995

National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

Preface

This report documents the implementation of an interface element capability within the COMET-AR software system. The report is intended for use by both users of currently implemented interface elements and developers of new interface element formulations. For guidance on the use of COMET-AR the reader should refer to Ref. 1-1. A glossary is provided as an Appendix to this report for readers unfamiliar with the jargon of COMET-AR. A summary of the currently implemented interface element formulation is presented in Section 7.3 of this report. For detailed information on the formulation of this interface element, the reader is referred to Refs. 1-8 through 1-10.

Table of Contents

Part I.
INTRODUCTION

THIS PAGE INTENTIONALLY BLANK

1. Introduction

1.1. Overview

This report describes the implementation of an interface element capability within the COMET-AR software system (Ref. 1-1) and contains a summary of the implementation, a simple analysis example for the new user, a description of the user interface (including generic procedures which may be used to access interface elements), a description of the developer interface, and a description of new data structures. The report has been designed for both users of existing interface elements and developers of new interface elements and is organized as follows:

I.	**Introduction.** Answers the questions: • What is an interface element? • What does an interface element do and why is it needed? • How does an analysis change when interface elements are used? • What are the limitations and assumptions of the element implementation?
II.	**A Simple Analysis Example.** Provides a simple example of an analysis using an interface element.
III.	**Procedures.** Describes new and modified procedures including: • Generic control procedures • Interface element cover procedures • Modified finite element analysis procedures • Master model analysis procedures
IV.	**Processors.** Describes the use of two new processors: • The Generic Interface Element Processor (EI) • The Master Model Processor (MSTR)
V.	**Developer Interface.** Describes programming details of the new processors including: • Generic interface element processor shell • Generic interface element processor cover • Master model generation in processor MSTR
VI.	**Data Objects.** Describes new data objects in the object oriented database including: • New nodal objects • New element objects
A.	**Glossary.** Defines terms used throughout the document.

New **users** should find Parts I through IV the most useful. **Developers** of new interface elements are directed to Parts I, V, and VI for programming information and Parts II and III for assistance in using the software. ***Both users and developers should be familiar with the COMET-AR system as described in the COMET-AR Users' Manual (Ref. 1-1).***

1.2. What is an "Interface Element?"

An interface element is a special type of finite element which connects independently modeled finite element substructures along their common interface. The connected finite element models need not have a one-to-one correspondence between the nodes across the common boundary (*i.e.*, they need not be nodally compatible). The interface element is therefore particularly useful for global/local analyses and for analyses involving component substructuring.

In the past, applications of coupled global/local analysis and component substructuring have required at least partial, and often full, nodal compatibility across global/local and substructure boundaries. Quite often, the transition across substructure boundaries is performed through some form of mesh transitioning. One technique uses either distorted quadrilateral or triangular finite elements to make the transition (called "htq" and "htt" refinement respectively, in COMET-AR). Some have developed special elements which typically connect two elements to one along a single boundary and are known by various names such as variable order elements (Refs. 1-2, 1-3) and transition elements (Ref. 1-4). All of these special elements require some degree of nodal compatibility (usually only two new elements may be connected to one original element). One of the most common means of transitioning between different quadrilateral discretizations is through the use of multipoint constraints which may be applied as constraints (*e.g.*, "hc" refinement in COMET-AR) or through a modification in the finite element formulation of the affected elements (Ref. 1-5). Both of these constraint techniques require nodal compatibility similar to the compatibility required of the special elements.

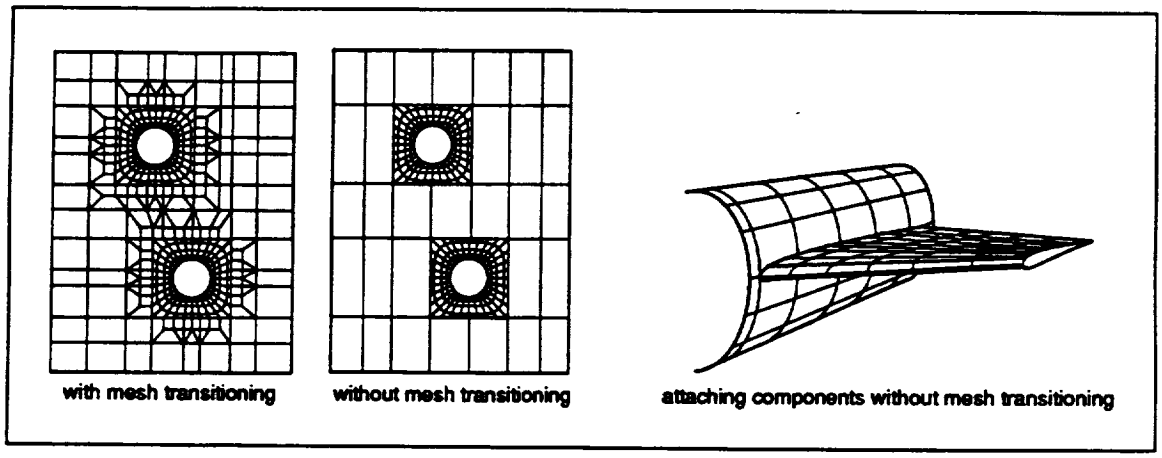


Figure 1.1. Examples of Mesh Transitions

Each of these transitioning techniques potentially introduces additional error into the solution due to constraints or distortion and each also requires at least some degree of nodal compatibility. Several coupling methods which do not require nodal compatibility (*e.g.*, see Figure 1.1) on substructure or element boundaries have been developed (Refs. 1-6, 1-7). However, these methods have also typically had difficulty in maintaining solution accuracy, particularly near the common substructure boundaries. Recent work has focused on the development of a means of connecting independently modeled substructures which maintains solution accuracy (Refs. 1-8, 1-9). Several techniques for tying together two substructures (*i.e.*, collocation, least-squares, and hybrid variational) have been examined. It was concluded that the use of an independent function to connect two independently modeled substructures through a hybrid variational formulation was an effective method of connecting such substructures. The method preserves solution accuracy (of displacements *and* stresses) across the common substructure boundaries. The interface element reported on in Ref. 1-10 represents a generalization of this previous work.

Software implementation of the interface element concept was driven by three requirements: (1) the need for a general implementation to accommodate potentially several different types of interface formulations; (2) the need to extend, in the future, the hybrid variational formulation to include nonlinear and dynamic effects and to permit its application in adaptive refinement; and (3) the need for a user-friendly environment in which the interface technique could be used to solve realistic, potentially large, structures problems. Original prototype software served well during the "proof-of-concept" phase but required very large amounts of disk space and large amounts of machine memory. It was also severely limited in its application in that it could not process multiple interfaces, more than two substructures, or generally curved interfaces. By recasting the interface formulation in the form of an element (much like a finite element) and creating a new software framework for the element implementation, all of the requirements are met. Developers of new interface formulations have a platform of support software readily available and may insert new software kernels without understanding the requirements of accessing the database. Extensions to the existing hybrid variational formulation may be implemented by adding new kernel modules (subroutines). Because it was implemented within a general-purpose software system, COMET-AR, the interface element can be used to solve practical applications. This report provides a detailed description of the interface element implementation.

A few words on notation:

Consistent notation is used throughout this report.

- file names and path names are denoted by bold italics (*e.g.*, ***initialize.clp***, ***/ar/el***);
- processor names are denoted by uppercase bold (*e.g.*, **MSTR**, **EI1**);
- procedure names are denoted by mixed case bold (*e.g.*, **EI_Define**, **Form_EI_Stiffness**);
- an asterisk in a processor or procedure name indicates a wild card (*e.g.*, **EI*** indicates any EI processor, EI1, EI2, etc.);
- attributes are denoted by roman bold (*e.g.*, **Nnode**);
- actual runstream examples are listed in a courier font(*e.g.*, `run MSTR`);
- templates are listed in the text font (*e.g.*, run MSTR);
- Processor commands, subcommands, qualifiers, and procedure argument names are in upper case text (*e.g.*, RESET, DEFINE ELEMENTS);
- argument values are denoted by text italics (*e.g.*, *auto_dof_sup_flag*);
- macrosymbols are denoted by mixed case bold (*e.g.*, **Auto_dof_sup**).

1.3. Overview of the Implementation Strategy

COMET-AR (Ref. 1-1) is a modular software system composed of the standard finite element modules (*e.g.*, model definition, assembly) along with modules which perform error estimation and mesh refinement. These modules are semi-independent FORTRAN executables called processors. The system allows for extensions through the addition of both new processors and new command language procedures which provide high level control, may operate on data using the command language CLAMP, and typically call processors to perform the more compute-intensive tasks associated with a structural analysis.

The implementation of the interface element was accomplished by adding both new processors and new procedures to COMET-AR. The flowchart in Figure 1.2 describes the solution process when using interface elements. Initially, the user must define each substructure completely (*i.e.*, node locations, element connectivity, loads, boundary conditions, material and section properties). The substructure definitions serve as input to the interface element definition which is accomplished through a new generic interface element (EI) processor (shown as the shaded boxes in the Figure). Interface elements are defined by the substructures to which they are connected and may internally generate new displacement nodes (herein called pseudo-nodes) and/or traction nodes (herein called alpha-nodes). Once the interface elements have been defined, all element stiffness matrices are formed and unstiffened degrees-of-freedom (*e.g.*, drilling degrees-of-freedom) are suppressed. The various substructures are then merged into a single, global, master model for the purposes of assembly and solution. The new master model processor, **MSTR**, (shown as the large box in the Figure) combines all input substructures by renumbering nodes sequentially and then copying and modifying the data needed to effect a solution (*i.e.*, element connectivity, active degree-of-freedom tables, element matrices and vectors, nodal vectors). With all data in a single library file, the standard assemblers and solvers may be used on the global master model. The **MSTR** processor may be used after the solution has been obtained in order to extract substructure results from the master model. Note that while n substructures are depicted in Figure 1.2, a single model may be used with interface elements connecting various parts of the one defined model.

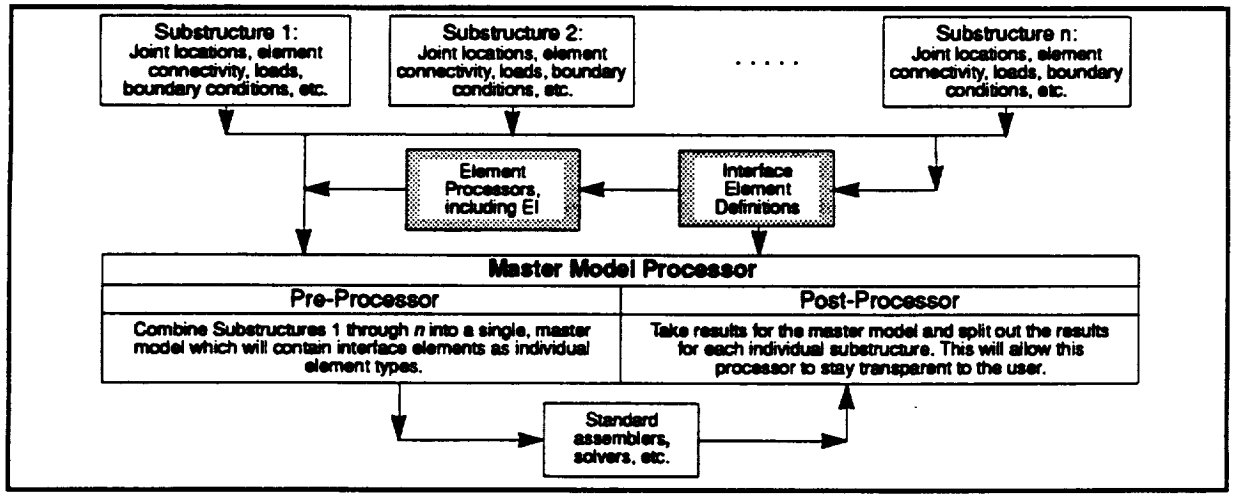


Figure 1.2. Coupled Analysis Solution Strategy

The generic nature of the EI processor facilitates the implementation of additional interface formulations within the general-purpose framework thereby enabling future research in interfacing techniques. The processor is designed so that an interface element developer is isolated from all user and database interaction. The user interface for the interface element is composed of both processors and procedures. While all interaction may be through processors (the EI generic interface element processor and the **MSTR** master model generator), cover procedures have been written which simplify the user interaction.

1.3.1. New Procedures

Several procedures which hide the actual new processor execution have been written. Macrosymbols are used to define such things as file names and procedure names. A script file template for execution (***SS_control.com***) has also been provided and may be adapted to execute most applications. The template calls a procedure named **SS_control** (discussed fully in Section 3.2) that coordinates (automatically) the flow of the analysis.

The procedures required to run an analysis with interface elements are summarized in the Table 1.1. The experienced COMET-AR user should note the absence of familiar procedures (*e.g.*, **L_STATIC_1**, **STIFFNESS**). These "normal" analysis procedures have been split into functional pieces and incorporated into the procedures listed in Table 1.1 in order to conform to the new analysis flow depicted in Figure 1.2. Of the procedures listed in the Table, only three must be user defined: **Initialize**, **EI_Define**, and **Merge_SS**. The remaining procedures rely on macrosymbols defined by the user in the **Initialize** procedure and are transparent to the user since they are invoked automatically by the control procedure, **SS_control**. All required user action is discussed in the Sections listed.

Table 1.1. Summary of New Procedures

Procedure Name	Function	Section
Control Procedures		**3**
SS_control	Controls the analysis. No user interaction is required other than modification of the ***ss_control.com*** template file.	3.2
Initialize	Initializes required macrosymbols. Requires modification by user.	3.3
Post_FE_Stress	Controls stress resultant recovery.	3.4
EI Processor Cover Procedures:		**4**
EI_Define	Template for interface element definitions. Requires user modification.	4.2
Defn_EI_Freedoms	Defines the active degrees of freedom for each node (specifically each pseudo-node and alpha-node) in the interface element substructure. Called automatically by **SS_control**; requires no user action.	4.3
Form_EI_Stiffness	Forms interface element stiffness matrices. Called automatically by **SS_control**; requires no user action.	4.4
Modified Finite Element Procedures:		**5**
Initialize_FE	Perform finite element substructure initialization. Called automatically by **SS_control**; requires no user action.	5.2
Defn_FE_Freedoms	Define the active degrees of freedom for each node in the finite element substructure. Called automatically by **SS_control**; requires no user action.	5.3
Form_FE_Force	Forms consistent load vector. Called automatically by **SS_control**; requires no user action.	5.4
Form_FE_Stiffness	Form element stiffness matrices for finite element substructures. Called automatically by **SS_control**; requires no user action.	5.5
Comp_FE_Stress	Compute finite element stress resultants. Called through **Post_FE_Stress**.	5.6
Comp_Nodal_Stress	Compute smoothed nodal stress resultants for substructures and master model. Called through **Post_FE_Stress**.	5.7
Master Model Analysis Procedures		**6**
Merge_SS	Merge finite element and interface element substructures into a single master model. May require modification by user.	6.2
Assemble_Master	Perform assembly of master model system stiffness matrix and load vector. Called automatically by **SS_control**; requires no user action.	6.3
Solve_Master	Execute the appropriate solver for the assembled master model. Called automatically by **SS_control**; requires no user action.	6.4

1.3.2. New Processors

The software framework developed for the interface element has been used to implement a hybrid variational interface element; this same framework may also be used by developers to implement additional interface formulations as new interface element types. The generic interface element implementation is based on the same philosophy used in the generic element processor (GEP) implementation of structural elements (Ref. 1-11). Just as specific structural elements are implemented via new ES (Element-Structural) processors, additional interface elements may be implemented via new EI (Element-Interface) processors. While the GEP served as a model, the requirements of the interface element are such that substantial effort was invested in creating a GEP tailored for the interface elements. One of the new features is a provision for "traction nodes," that is, nodes for which the unknowns are tractions rather than displacements or rotations. These traction nodes currently have no meaningful physical location (*i.e.*, their nodal coordinates are arbitrarily assigned) but rather, exist along the edges of finite elements connected at a given interface. Nodes introduced along the interface (*i.e.*, not attached to a finite element but attached only to the interface element) which have displacement and/or rotational degrees of freedom are denoted "pseudo-nodes." Traction nodes are denoted "alpha-nodes."

The EI processor (depicted in Figure 1.3) has a generic software shell (which provides for uniform user input and database interaction) and a software cover (which communicates between the shell and the developer supplied kernels). Each developer of new interface elements must supply the software kernels which form the interface element stiffness matrix. All interaction with the database is accomplished for the developer through the software shell using High level DataBase (HDB) utilities (Ref. 1-12).

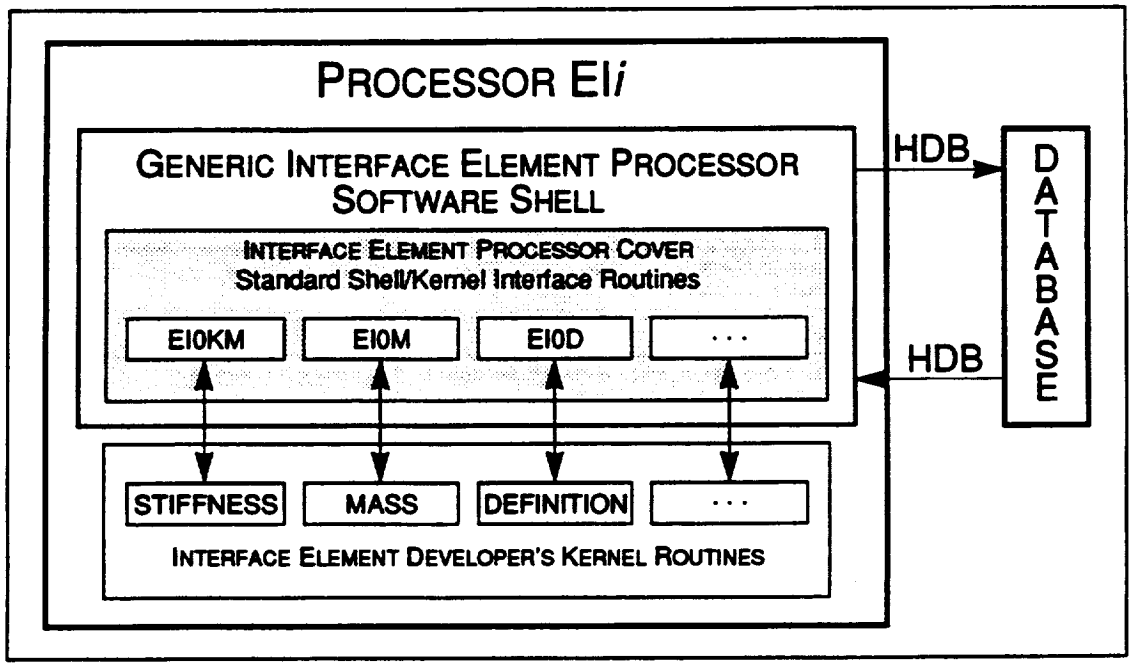


Figure 1.3. Interface Element Processor Design

The EI processor permits both the automatic and user-specified definition of the interface element pseudo-nodes. For example, the currently implemented hybrid variational interface element (processor EI1) will select automatically a proper number of pseudo-nodes or will permit the user to specify the number of pseudo-nodes. Thus, the number of pseudo-nodes may be determined either within a developer-written kernel or through user input but must fall within a range which ensures that the resultant global system will be

nonsingular. Whether user-specified or developer determined, the **EI** processor will generate the pseudo-nodes as actual nodes in the database. If tractions exist as unknowns (as they do in the **EI1** version of the hybrid variational interface element), the processor will generate alpha-nodes as actual nodes in the database. An interface element connectivity is written to the database and consists of the finite element nodes of each connected substructure along with the node numbers of the pseudo-nodes and the alpha-nodes. Thus, the interface element stiffness matrices may be assembled as any other element matrix (*i.e.*, the assembler simply uses the element connectivity).

The **EI** processor shell calculates the geometry of the interface element so that it is independent of the specific element formulation. This element geometry may be determined in one of three ways. The user may define a function (currently limited to a linear function) that represents the exact geometry of the interface. In this case, the **EI** shell will identify the substructure nodes lying along the function. The user may alternatively specify the nodes through which a function (piecewise linear, quadratic spline, or cubic spline) is passed. In this case, the **EI** shell will read the nodes, retrieve their coordinates, and construct the interface element path. The third option for definition of the element geometry is available only for interface elements with linear geometry. Using this option, the user may specify only the nodes at the end points of the interface. In this case, the **EI** processor will internally construct a line between the two nodes, identify the substructure finite element nodes lying along the line, and construct the interface element.

A second processor which merges the substructures into a single, master finite element model is also provided. The Master Model Processor, **MSTR**, renumbers all of the input nodes (including pseudo-nodes and alpha-nodes) sequentially, renumbers the elements, rewrites the element connectivities, and copies all the data required for the solution into a single library file. The resulting master model then contains both finite elements (possibly several different types) and interface elements. The element stiffness matrices may then be assembled using the available assembly processor (*e.g.*, processor **ASM**) and the resulting global system of equations may be solved using a conventional solver (*e.g.*, processor **PVSNP**).

1.4. Organization

The files required to run an analysis using interface elements have been consolidated into a directory structure which all users and developers may access (to read). This directory structure is outlined in Table 1.2. The environment variable ***$AR_ROOT***, as well as the environment variables listed in the Table, will be set up automatically upon initialization of the COMET-AR system (see Section 2.2).

Table 1.2. Directory Structure for Interface Elements

Directory	Environment Variable	Function
$AR_ROOT/ei	***$AR_EI***	Top level interface element directory
$AR_ROOT/ei/mods	***$AR_EIMODS***	Top level for software developers
$AR_ROOT/ei/mods/inc	***$AR_EIINC***	Include files for EI and MSTR processors
$AR_ROOT/ei/mods/ei	***$AR_EISRC***	Source and object files for the EI shell. Includes a template for ***EI*cover.ams*** and a ***makefile***.
$AR_ROOT/ei/mods/ei/ei1	***$AR_EI1***	Source, object, and executable files for processor EI1. Includes source and object for ***EI1_cover.ams*** and a ***makefile***.
$AR_ROOT/ei/mods/mstr	***$AR_MSTR***	Source, object, and executable files for the MSTR processor.
$AR_ROOT/ei/bin	***$AR_EIBIN***	Executables for EI1 and MSTR.
$AR_ROOT/ei/prc	***$AR_EIPRC***	Top level for users. Contains the ***proclib.gal*** procedure library and templates for user written procedures and scripts.
$AR_ROOT/ei/prc/control	None	Control procedures
$AR_ROOT/ei/prc/utility	None	Utility procedures
$AR_ROOT/ei/demo	***$AR_EIDEMO***	Demonstration and analysis example files
$AR_ROOT/ei/applications	None	Applications procedure files

A template file for execution of an analysis, called ***ss_control.com***, is located in ***$AR_ROOT/ei/prc/***. An explanation of this file appears in Chapter 2.

1.5. Limitations, Implicit Assumptions, Conventions

There are currently limitations on the range of application of the interface element. Certain assumptions have been made which place additional limitations on the interface element's use. In the future, as the implementation is broadened to include additional functionality, many of these may be addressed.

1.5.1. Limitations

- All interface elements must be in a single library and only interface elements are assumed to be in that library. As long as all interface elements are defined in a single execution of the EI processor, this will remain so. Note that this means that the current implementation will not permit the mixing of interface element processors or types.
- Only Finite Element and Interface Element substructures are explicitly provided for at present. While the user input has hooks for Rayleigh-Ritz and Boundary Element Substructures, these types of substructures do not currently exist in COMET-AR.
- Interface elements may only be applied in linear static applications.
- Each finite element along the interface is of uniform order on each element edge (*i.e.*, finite elements must have the same number of nodes on each element edge or must be implemented so that they appear to be this way).
- For each substructure, all finite elements along the interface are of the same order. The order of the finite elements need not be the same for each attached substructure.
- Stresses or stress resultants cannot currently be computed on the Master Model. The displacement solution must be split out for each substructure (using the **MSTR** processor) and stresses calculated at the substructure level. However, utilities exist which allow the user to combine substructure stresses into master model stresses.
- The choices for the geometry and displacement interpolation functions are limited to: a piecewise linear function, quadratic spline, or cubic spline. The geometry and displacement interpolation functions may be different functions (*e.g.*, piecewise linear geometry and cubic spline displacement).
- Only 8 data libraries may be open at one time within the COMET-AR system. Therefore, there can be no more than 5 active substructure libraries. This restriction assumes that one library is used for the interface elements, one for the master model, and one for the procedure library, thereby leaving 5 libraries for use by the substructures.

1.5.2. Implicit Assumptions

- The user *must* understand how to use COMET-AR to perform an analysis.
- Each interface element processor contains only one interface element type.
- Interface elements may intersect each other only at end points.

1.5.3. Conventions

- Each substructure is assigned a unique identification number which remains with the substructure throughout the analysis (*i.e.*, substructure 1 remains substructure 1 from start to finish).
- Pseudo-node numbering begins at 1 in the interface element substructure. This happens automatically provided all interface elements are defined in one execution of the interface element processor.
- For each interface element, displacement nodes (*i.e.*, pseudo-nodes) are numbered first, traction nodes (*i.e.*, alpha-nodes) are numbered second.
- The Master model orders all of the finite element nodes first, all pseudo-nodes second, and all of the alpha-nodes last.
- Pseudo-nodes are evenly spaced along a given interface element.

1.6. Reference Frames

COMET-AR permits the use of several different reference frames: computational (the frame attached to each node in which the solution is obtained) denoted by the subscript "c," element (the frame attached to each finite element) denoted by the subscript "e," global (the frame in which the nodal coordinates are defined) denoted by the subscript "g," and material or stress (the frame that defines the principal material direction) denoted by the subscript "m." The interface element introduces two additional reference frames. The edge frame defines the finite element edge along the interface (the computational frame for the alpha-nodes) and is denoted by the subscript "d." The interface frame defines the interface path (the computational frame for the pseudo-nodes) and is denoted by the subscript "s." Figure 1.4 depicts these various reference frames. Finite element nodes are denoted by filled circles; pseudo-nodes are denoted by filled squares.

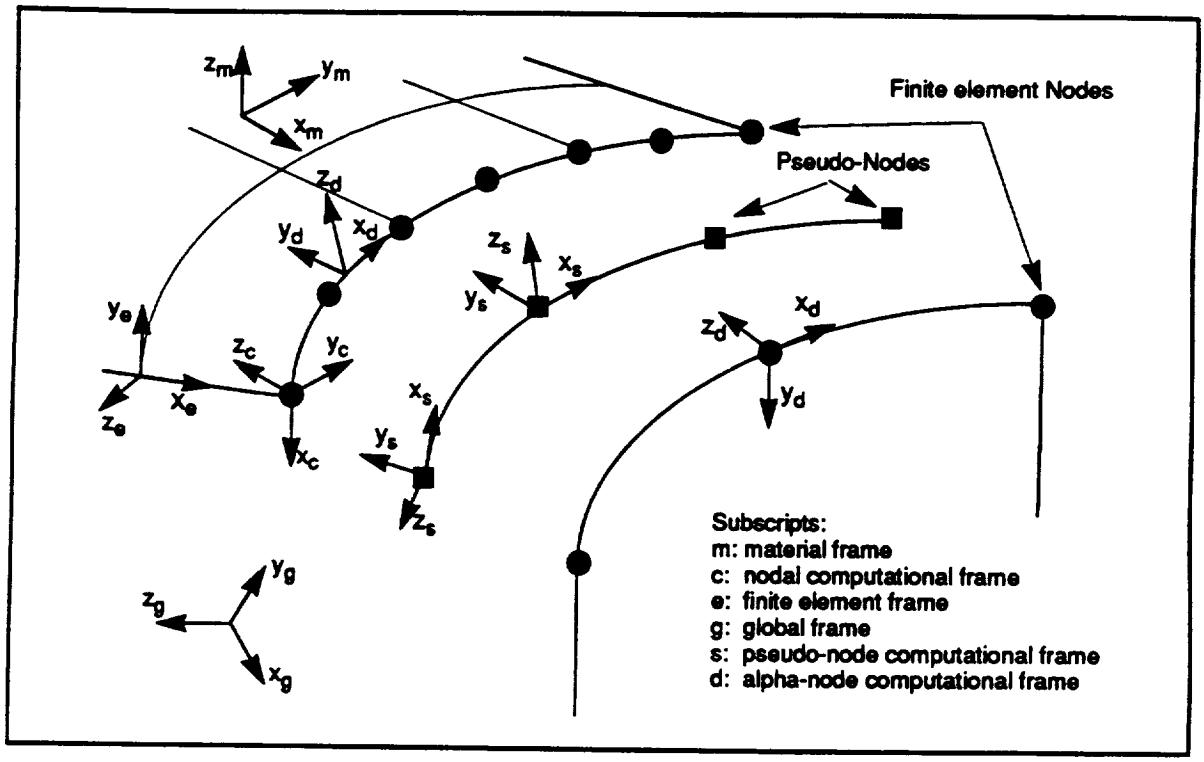


Figure 1.4. Interface Element Reference Frames

1.7. References

1-1 Stanley, G.M., Hurtbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

1-2 Bathe, K.J., *Finite Element Procedures in Engineering Analysis*, Prentice Hall, New Jersey, 1982.

1-3 Choi, C.K., and Park, Y.M., "Transition Plate Bending Elements with Variable Nodes," *Numerical Techniques for Engineering Analysis and Design - Proceedings of the International Conference on Numerical Methods in Engineering: Theory and Applications, NUMETA '87*, edited by G.N. Pande and J. Middleton, Martinus Nijhoff Publishers, Boston, 1987, pp. D31/1-D31/8.

1-4 Subbaraj, K. and Dokainish, M.A., "Side-Node Transition Quadrilateral Finite Element for Mesh-Grading," *Computers and Structures*, Vol. 30, No. 5, 1988, pp. 1175-1183.

1-5 McDill, J. M., Goldak, J. A. Oddy, A. S., and Bibby, M. J., "Isoparametric Quadrilaterals and Hexahedrons for Mesh-Grading Algorithms," *Communication in Applied Numerical Methods*, Vol. 3, 1987, pp.155-163.

1-6 Maday, Y., Mavriplis, D., and Patera, A., "Nonconforming Mortar Element Methods: Application to Spectral Discretizations," NASA CR-181729, ICASE Report No. 88-59, October 1988.

1-7 Shaeffer, H.G., *MSC/NASTRAN Primer, Static and Normal Modes Analysis*, Shaeffer Analysis, Inc., Mont Vernon, New Hampshire, 1979, pp. 262-265.

1-8 Aminpour, M. A., Ransom, J. B., and McCleary, S. L., "Coupled Analysis of Independently Modeled Finite Element Subdomains," AIAA Paper Number 92-2235, 1992.

1-9 Aminpour, M.A., McCleary, S.L., and Ransom, J.B., "A Global/Local Analysis Method for Treating Details in Structural Design," *Proceedings of the Third NASA Advanced Composites Technology Conference*, compiled by J.G. Davis, Jr. and H.L. Bohon, NASA CP-3178, Vol. 1, Part 2, 1992, pp. 967-986.

1-10 Ransom, J. B., McCleary, S. L., and Aminpour, M. A., "A New Interface Element for Connecting Independently Modeled Substructures," AIAA Paper Number 93-1503, 1993.

1-11 Stanley, G. M. and Nour-Omid, S., *The Computational Structural Mechanics Testbed Generic Structural-Element Processor Manual*, NASA Contractor Report 181728, March 1990.

1-12 Stanley, G. M. and Swenson, L., *HDB Object-Oriented Database Utilities for COMET-AR*, NASA CSM Contract Report, August, 1992.

THIS PAGE INTENTIONALLY BLANK

Part II.
ANALYSIS EXAMPLE

THIS PAGE INTENTIONALLY BLANK

2. A Simple Analysis Example

2.1. Overview

This Chapter contains a simple example of an analysis using a single interface element. It is assumed that the user is familiar with COMET-AR. The example application is a cantilever beam with a variable end load. User-written procedures and a script for executing the analysis are provided. The Chapter contains the following sections:

Table 2.1. Outline of Chapter 2: A Simple Analysis Example

Section	Topic	Function
2	**Application: Cantilever Beam**	Explains the use of the new software for a simple, single interface analysis

Section 2 contains example model generation and analysis procedures. Each procedure is accompanied by an explanation of the required user action. ***This Chapter is not a tutorial*** in the sense that it does not provide step-by-step instructions on how to use COMET-AR. Rather, the user is assumed to have knowledge of COMET-AR, its procedures and how to read them, and how to perform an analysis. The Chapter focuses on providing the user with the procedures required for an example application, highlighting the additional requirements of the interface element.

The COMET-AR initialization procedure has been updated to reflect the interface element software. New environment variables have been included and are automatically defined when the COMET-AR ***login*** file is executed. Running an analysis using interface elements requires several steps which may be summarized as follows:

1. Create a new directory for the new application.
2. Copy the files:
   - ***$AR_EIPRC/SS_control.com***
   - ***$AR_EIPRC/ei_define.clp***
   - ***$AR_EIPRC/merge_ss.clp***
   - ***$AR_EIPRC/initialize.clp***

   into the new application directory.
3. Create model definition files for the given application. Note that models may be created through PATRAN (or some other model generation software) or through command language procedures.
4. Modify the procedure files:
   - ***ei_define.clp***
   - ***merge_ss.clp***
   - ***initialize.clp***

   to reflect the current application.
5. Modify the ***SS_control.com*** script file to reflect the current application.
6. Run the analysis.
7. Post-process the results as required.

Steps 1 through 6 are described in the following Sections. Where appropriate, user actions are highlighted and summarized at the bottom of each page. Post-processing may occur at either the substructure or the master model level and may be performed with the usual post-processing facilities (e.g,.PATRAN).

THIS PAGE INTENTIONALLY BLANK

2.2. Application: End-Loaded Cantilever Beam

2.2.1. General Description

The application described in Figure 2.1 is a simple example of an analysis using a single interface element. The cantilever beam may be loaded in tension, in-plane or out-of-plane shear, or bending at the beam tip.

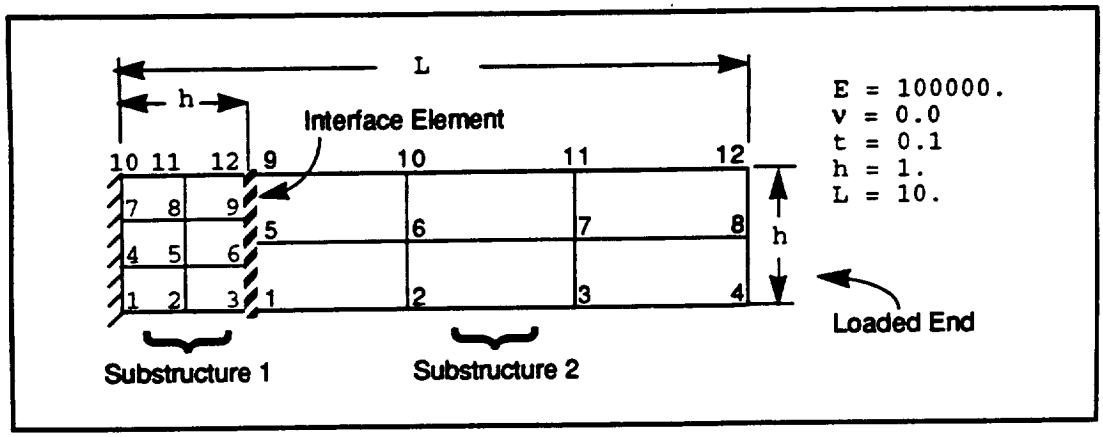


Figure 2.1. Cantilever Beam with Various End Loads

While not required, the user should begin by creating a new directory within which the analysis will take place. By keeping each analysis in a separate directory, there is less chance for confusion since procedure files will have to be added for each different application. For this example, a directory named ***beam*** could be created and the files:

- ***$AR_EIPRC/SS_control.com***
- ***$AR_EIPRC/el_define.clp***
- ***$AR_EIPRC/merge_ss.clp***
- ***$AR_EIPRC/initialize.clp***

copied into this directory. Note that the environment variable ***$AR_EIPRC*** is defined during the COMET-AR initialization (i.e execution of the ***cometar.login*** file). Once all the necessary files are in place, the user must create the model definition procedures.†

INITIALIZATION USER ACTION
• Ensure proper COMET-AR initialization • Create an application directory named ***beam*** • Copy the files: ***$AR_EIPRC/SS_control.com*** ***$AR_EIPRC/el_define.clp*** ***$AR_EIPRC/merge_ss.clp*** ***$AR_EIPRC/initialize.clp*** to the ***beam*** directory. • Proceed to the model definition (next Section)

† Note that the purpose of this Chapter is to assist the user in running an analysis with interface elements; it is not to teach a new user how to perform an analysis with COMET-AR. **Those unfamiliar with COMET-AR should consult the COMET-AR User's Manual and Tutorial documents as needed.**

2.2.2. Model Definitions

The model definition procedures must fully define each of the substructures. Full substructure definition includes the definition of: nodal coordinates, element connectivity, boundary conditions, applied loading, and material and section properties. The configuration of the application shown in Figure 2.1 lends itself to the use of a generic rectangular grid generation procedure for the definition of the models of both finite element substructures. This generic procedure along with procedures for defining the substructures identified as Substructure 1 and Substructure 2 in Figure 2.1 are provided in the following Sections.

The model definitions are initiated by first copying the file ***$AR_EIDEMO/beam/beam_util.prc*** to the current working directory. This file contains the generic model generation procedure and its subordinate procedures. Each specific model generation procedure (for each of Substructures 1 and 2) will call the top level generic procedure contained in this file and named **BEAM_MODEL**. The model generation procedures use several user-defined macrosymbols. These macrosymbols are accessed by copying the procedure file ***$AR_EIDEMO/beam/macros.clp*** into the current working directory.

MODEL DEFINTION USER ACTION
• Copy ***$AR_EIDEMO/beam/beam_util.prc*** to the application directory (***beam***).
• Copy ***$AR_EIDEMO/beam/macros.clp*** to the application directory (***beam***).
• Define user macrosymbols, if any, in a procedure file as in Section 2.2.2.2.
• Create a procedure to define Substructure 1 as in Section 2.2.2.3.
• Create a procedure to define Substructure 2 as in Section 2.2.2.4.
• Proceed to the definition of the required macrosymbols as in Section 2.2.3.

2.2.2.1 Generic Rectangular Mesh Generation Procedure

The generic modeling procedure **BEAM_MODEL** creates a regular, rectangular finite element model which may be loaded and/or constrained on any edge. It is fully parameterized and uses various arguments to determine the dimensions and location of the rectangular region, along with the specification of loading and boundary conditions. Within the file ***$AR_EIDEMO/beam/beam_util.prc***, is a set of utility procedures which may be used repeatedly for the model definitions of any combination of regular, rectangular regions in the x-y plane (minor modifications are required for regions which have a nonzero or varying z coordinate). Once this file has been added to the current procedure library, the user need only call **BEAM_MODEL** with the proper arguments; the subordinate procedures will be called automatically but will remain invisible to the user. A listing of the **BEAM_MODEL** procedure follows:

```
*procedure BEAM_MODEL (  es_proc  ;es_type;    --  . ES processor name and element type
                         es_nen   ;            --  . Number of nodes for this element type
                         nelx     ; nely   ;   --  . Number of elements in x and y directions
                         load_dir ;            --  . Direction of applied load (if any)
                         consedge ; cons   ;   --  . Edge # of constr. edge and const. dofs
                         loadedge ; load   ;   --  . Edge # of loaded edge and load values
                         x0 ; y0  ;            --  . Coordinates of first node in region
                         Lx ; Ly  ;            --  . Length in x and y of the region
                         E  ; PR  ; THICK  )       . Young's mod., Poisson's ratio, thickness
*remark ******************
*remark Defining Beam Model
*remark ******************
. ----------------------------------------
. Define Nodal Coordinates and Transformations (and Model Summary)
. ----------------------------------------
*call DEF_NODES (  nen  =[es_nen]; nelx =[nelx]; nely =[nely]; --
                   x0=[x0]; y0=[y0]; Lx=[Lx]; Ly=[Ly] )
. ----------------------------------------
. Define Material and Fabrication Data
. ----------------------------------------
*call DEF_FABS (   E = [E] ; PR = [PR] ; THICK = [THICK] )
. ----------------------------------------
. Define Element Connectivity (Nodal, Fabrication and Solid Model)
. ----------------------------------------
*call DEF_ELTS (   es_proc=[es_proc]; es_type=[es_type]; es_nen =[es_nen] )
. ----------------------------------------
. Define Loads and Boundary Conditions
. ----------------------------------------
*call DEF_LBC (    consedge = [consedge] ; cons    = [cons]    ; --
                   loadedge = [loadedge] ; load    = [load]    ; --
                   nelx     = [nelx]     ; nely    = [nely]    ; --
                   es_proc  = [es_proc]  ; es_type = [es_type] ; load_dir = [load_dir] )
*end
```

2.2.2.2 Model Definition Macrosymbols

The model definition for this example is facilitated through the use of a number of macrosymbols contained, in this case, in a separate procedure which resides in the file ***$AR_EIDEMO/beam/macros.clp***. This procedure contains macrosymbol definitions which are used in subsequent calls to the model definition procedures for the two substructures. By defining these macrosymbols either within a procedure or within the script file, the models may be modified while the model definition procedures remain unaltered. A listing of the macrosymbol definition procedure follows:

```
*procedure MODEL_PARAMS
. Define model parameters using macrosymbol arrays. The # of items in each array is
. determined by the # of substructures (one item in each array for each substructure)
    *def/i num_models    == 2              . # of substructures (SS)
    *def/i nelx          == 2,3            . # of elements in x direction for each SS
    *def/i nely          == 3,2            . # of elements in y direction for each SS
    *def/e x0            == 0.0,1.0        . x-coordinate of the first node for each SS
    *def/e y0            == 0.0,0.0        . y-coordinate of the first node for each SS
    *def/e Lx            == 1.0,4.0        . x-dimension for each SS
    *def/e Ly            == 1.0,1.0        . y-dimension for each SS
    *def/i consedge      == 4,0            . Edge # of constrained edge for each SS
    *def/a cons          == 'fixed','none' . Constraints for each SS (all nodes on edge)
    *def/i loadedge      == 0,2            . Edge # of loaded edge for each SS
    *def/e load          == 0.0,1.0        . Loading for each SS (applies to <loadedge[i]>)
    *def/e load_dir      == 0,1            . Direction of applied loading (0 => no load)
    *def/a ES_PROC       == ES1            . ES processor name
    *def/a ES_TYPE       == Ex47           . ES element type name
    *def/i es_nen        == 4              . # of nodes per element of <es_type>
*end
```

2.2.2.3 Substructure 1 Model Definition

With the generic modeling procedure and its subordinate procedures and the macrosymbol definitions in place, it remains to define procedures for each of the substructures. The following procedure, located in a file named ***$AR_EIDEMO/beam/model1.clp***, is an example of a procedure which will fully define the model for Substructure 1 provided the generic model and macrosymbol definition files previously discussed are used.

```
*procedure Model1_Def
. Call the generic model procedure using the macrosymbols which define substructure 1
*call BEAM_MODEL (  es_proc   = <es_proc>      -- . ES processor name
                    es_type   = <es_type>      -- . ES element type
                    es_nen    = <es_nen>       -- . # of nodes for this ES type
                    nelx      = <nelx[1]>      -- . # of elements in x direction
                    nely      = <nelx[1]>      -- . # of elements in y direction
                    consedge  = <consedge[1]>  -- . Edge # of constrained edge
                    cons      = <cons[1]>      -- . Constrained dofs
                    loadedge  = <loadedge[1]>  -- . Edge # of loaded edge
                    load      = <load[1]>      -- . Load values
                    load_dir  = <load_dir[1]>  -- . Load direction
                    x0        = <x0[1]>        -- . x coordinate of first node
                    y0        = <y0[1]>        -- . y coordinate of first node
                    Lx        = <Lx[1]>        -- . Length in x
                    Ly        = <Ly[1]>        -- . Length in y
                    E         = 1.0E5          -- . Young's modulus
                    PR        = 0.0            -- . Poisson's ratio
                    THICK     = 0.01           )  . thickness
*end
```

2.2.2.4 Substructure 2 Model Definition

The procedure defining Substructure 2 is nearly identical to the procedure of the previous section (which defined Substructure 1). The only differences between the two are in the procedure name (which reflects the substructure number) and in the macrosymbols used (the second item in the list is now used rather than the first). The following procedure, located in a file named ***$AR_EIDEMO/beam/model2.clp***, is an example of a procedure which will fully define the model for Substructure 2 provided the generic procedure and macrosymbol definition files previously discussed are used.

```
*procedure Model2_Def
. Call the generic model procedure using the macrosymbols which define substructure 2
*call BEAM_MODEL (  es_proc   = <es_proc>      -- . ES processor name
                    es_type   = <es_type>      -- . ES element type
                    es_nen    = <es_nen>       -- . # of nodes for this ES type
                    nelx      = <nelx[2]>      -- . # of elements in x direction
                    nely      = <nelx[2]>      -- . # of elements in y direction
                    consedge  = <consedge[2]>  -- . Edge # of constrained edge
                    cons      = <cons[2]>      -- . Constrained dofs
                    loadedge  = <loadedge[2]>  -- . Edge # of loaded edge
                    load      = <load[2]>      -- . Load values
                    load_dir  = <load_dir[2]>  -- . Load direction
                    x0        = <x0[2]>        -- . x coordinate of first node
                    y0        = <y0[2]>        -- . y coordinate of first node
                    Lx        = <Lx[2]>        -- . Length in x
                    Ly        = <Ly[2]>        -- . Length in y
                    E         = 1.0E5          -- . Young's modulus
                    PR        = 0.0            -- . Poisson's ratio
                    THICK     = 0.01           )  . thickness
*end
```

2.2.3. Definition of Required Macrosymbols

Prior to defining the interface elements, a customized version of the file ***Initialize.cip*** (which has already been copied into the working directory) should be created. This file contains a procedure which defines the global macrosymbols used by various utility procedures. While these macrosymbols do not have to be defined through this procedure, they must be defined prior to calling the control procedure, **SS_control**. It is highly recommended that the user adjust the template file rather than attempt to incorporate the definitions into other procedures or files elsewhere.

The following example of the **Initialize** procedure has been customized for this beam application. The new user should note that each substructure is saved in its own database which has been assigned a unique logical device index (ldi) or library number. Furthermore, the interface element and master model database file names and ldis are also unique. While the substructure models may be combined into a single database file, this is not recommended due to the absence of node and element label capabilities. The interface element and master model files and logical device indices must always be unique. That is, the interface elements must always be kept in a separate library (they are created in a new library).

```
*procedure Initialize
.
  *def/i Num_SS                    == 2                      . # of SS
  *def/i SS_List[1:<Num_SS>]       == 1,2                    . Id's for SS
  *def/a SS_Lib_Name[1]            == MODEL<SS_List[1]>.DBC  . Library file name SS1
  *def/a SS_Lib_Name[2]            == MODEL<SS_List[2]>.DBC  . Library file name SS2
  *def/a SS_Define_Prc[1]          == MODEL1_DEF             . Model definition procedure SS1
  *def/a SS_Define_Prc[2]          == MODEL2_DEF             . Model definition procedure SS1
  *def/i SS_ldi[1:<Num_SS>]        == 1,2                    . ldi for SS1 and SS2
  *def/i SS_step[1:<Num_SS>]       == 0,0                    . load step # for SS1 and SS2
  *def/i SS_load_set[1:<Num_SS>]   == 1,1                    . load set # for SS1 and SS2
  *def/i SS_con_set[1:<Num_SS>]    == 1,1                    . constr. set # for SS1 and SS2
  *def/i SS_mesh[1:<Num_SS>]       == 0,0                    . mesh id # for SS1 and SS2
  *def/a EI_Proc                   == EI1                    . IE processor name
  *def/a EI_Lib_Name               == 'interface.dbc'        . IE library file name
  *def/a EI_Define_Prc             == 'EI_Define'            . IE definition procedure
  *def/i EI_ldi                    == 4                      . IE logical device index
  *def/i EI_step                   == 0                      . Load step # for IE's
  *def/i EI_Load_set               == 1                      . Load set # for IE's
  *def/i EI_Con_set                == 1                      . Constraint set # of IE's
  *def/i EI_mesh                   == 0                      . Mesh # for IE's
  *def/a MM_Name                   == 'master.model'         . Master model (MM) library file
  *def/a Merge_SS_Prc              == 'Merge_SS'             . MM generation procedure
  *def/i MM_ldi                    == 3                      . MM logical device index#
  *def/i MM_step                   == 0                      . MM step #
  *def/i MM_Load_set               == 1                      . MM load set #
  *def/i MM_Con_set                == 1                      . MM constraint set #
  *def/i MM_mesh                   == 0                      . MM mesh number
  *def/i auto_dof_sup              == <true>                 . Auto dof suppression flag
  *def/i auto_drill                == <false>                . Artificial drill stiffness flag
  *def/i auto_triad                == <false>                . Auto nodal normal triads flag
  *def/a Post_Prc                  == 'Post_Test             . Post-processing procedure
*end
```

REQUIRED MACROSYMBOL DEFINITION USER ACTION

- Modify the ***Initialize.cip*** file to reflect the current application

2.2.4. Interface Element Definition

Once the substructure models have been generated, the user should proceed to the definition of the interface element(s). The file ***el_define.clp*** (which was copied earlier into the current working directory) should be modified to reflect the current application. The following procedure is a version of this file which has been customized for the beam application. Note that the interface element is defined by specifying substructures 1 and 2 and various parameters associated with the substructures. Referring to Figure 2.1, the user may verify that the nodes along the interface for finite element substructure 1 are nodes 3, 6, 9, and 12 and for finite element substructure 2 are nodes 1, 5, and 9 as shown in the input list. This model does not require that constraints be applied to the interface (either pseudo-nodes or alpha-nodes) as there are only two substructures and they are coplanar. The drilling degree of freedom will therefore be suppressed automatically. A detailed discussion of user input to the EI processor may be found in Chapter 7.

```
*procedure EI_Define
. Define Interface Elements
    run EI1
    . Processor Resets
        reset ldi       = <EI_ldi>
        reset mesh      = <EI_mesh>
        reset step      = <EI_step>
        reset load_set  = <EI_load_set>
        reset cons_set  = <EI_con_set>
    . Element Definitions
        DEFINE ELEMENTS
        ELEMENT 1
              SS 1 /LDI=1 /FE /CONS=1
                 NODES = 3:12:3
              SS 2 /LDI=2 /FE /CONS=1
                 NODES = 1:9:4
        END_DEFINE
*end
```

INTERFACE ELEMENT DEFINITION USER ACTION
• Edit the file ***el_define.clp*** to reflect the current application

2.2.5. Merging the Substructures into a Master Model

The introduction of the interface element into the analysis creates new requirements on both the analysis and the user. One of these requirements is the creation of a master model. With each substructure in a potentially different database file and the interface elements in yet another database file, a merge operation must be performed in order to take advantage of the current COMET-AR assemblers and solvers.

This merge operation combines specified substructures and the interface elements into a single master model. There is a utility procedure called **Merge_SS** (within the file ***merge_ss.clp***) which performs this merge for all of the substructures identified as active in the ***Initialize.clp*** file. If a selective merge is desired (i.e only some of these substructures are to be merged for a given analysis) the **Merge_SS** procedure should be modified to reflect the selected substructures. If all of the defined substructures are to be merged, the user need do nothing to the **Merge_SS** procedure. The following is a version of the procedure **Merge_SS** which is described in detail in Section 6.2 and which may be used unaltered for this application.

```
*procedure Merge_SS
. Merge User-specified substructures into a single library
    Run  MSTR
    . Define Substructures that will be merged
        DEFINE SUBSTRUCTURES
        *do $j = 1, <Num_SS>
        . Finite Element Substructures
            SUBstructure <SS_List[<$j>]> /fe
            Library          = <SS_ldi[<$j>]>       . SS library numbers
            Mesh             = <SS_mesh[<$j>]>      . SS mesh numbers
            Load_set         = <SS_load_set[<$j>]>  . SS load set numbers
            Constraint_case  = <SS_con_set[<$j>]>   . SS constraint case numbers
            Load_step        = <SS_step[<$j>]>      . SS load step numbers
        *enddo
        . Interface Element Substructure
            SUBstructure <<SS_List[<$j>]>+1> /ie
            Library          = <EI_ldi>             . Interface Element library
            Mesh             = <EI_mesh>            . Interface Element mesh
            Load_set         = <EI_load_set>        . Interface Element load set
            Constraint_case  = <EI_con_set>         . Interface Element constraint case
            Load_step        = <EI_step>            . Interface Element load step
        END_DEFINE
    . Perform the Merge operation
        MERGE <SS_List[1:<Num_SS>]>,<<Num_SS>+1>
            File             = <MM_name>            . Master model library file name
            Library          = <MM_ldi>             . Master model ldi number
            Mesh             = <MM_mesh>            . Master model mesh
            Load_set         = <MM_load_set>        . Master model load set
            Constraint_case  = <MM_con_set>         . Master model constraint case
            Load_step        = <MM_step>            . Master model load step
        END_MERGE
    STOP
*end
```

MERGING SUBSTRUCTURES USER ACTION

- Edit (as needed) the file ***merge_ss.clp*** to reflect the current application

2.2.6. Running the Analysis

At this point, the models for the substructures, the interface elements, and the master model will have been defined, and it remains only to prepare a script and to run the analysis. If procedure files have been used for the model definitions and the macrosymbol definitions (both user desired and required), the script file may look much like the following file. A script file template has been provided in ***$AR_EIPRC*** and is called ***SS_control.com***. As its name implies, this file is a template which contains the commands necessary to control the analysis. The user will have already copied this script file into the current working directory and should modify it as needed for this application. The following is a version of the script ***SS_control.com*** which has been modified for the current application. Typical user modifications may include changing file names, changing procedure names, and setting arguments to limit the scope of the execution. The reader should note the order of the calls to procedures. All macrosymbol definition procedures must be called prior to calling the procedure named **SS_control** (see Section 3.2 for a complete discussion). This control procedure decides what to do and how to do it based on the macrosymbols defined in the **Initialize** procedure and the arguments passed through the call. The arguments are all logical (.e either <true> or <false>) and turn on (<true>) or off (<false>) the named functions. For example, if the argument DEFINE_SS is set to<true>, then all substructures indicated by the **SS*** macrosymbols will be defined. If DEFINE_SS is set to<false>, then the control procedure will assume that the substructure definitions have already been completed and that data libraries exist which fully define the substructures.

```
rm proclib.gal DBdebug.dat
cp $AR_EIPRC/proclib.gal .
cometar << \endinput
*set echo off
.  ADD proper files; set up the procedure library
*set plib = 28
*open 28 proclib.gal /old
*add macros.clp                                    . User macros
*add model1.clp                                    . Model 1
*add model2.clp                                    . Model 2
*add initialize.clp                                . Required macros
*add beammodel.clp                                 . Generic model definition
*add ei_define.clp                                 . Interface element def'n.
*add merge_ss.clp                                  . Master model merge
. Define Macrosymbols needed for model generation
*call MODEL_PARAM
. Define Macrosymbols required by the interface element procedures
*call Initialize
*call SS_control (  Define_SS     = <true> ; --  . Define substructures?
                    Define_EI     = <true> ; --  . Define Interface elements?
                    Merge_SS      = <true> ; --  . Merge substructures?
                    Assemble      = <true> ; --  . Assemble master model?
                    Solve         = <true> ; --  . Solve master model system?
                    Post_Process  = <false> )    . Post process?
Run Exit
\endinput
```

RUNNING AN ANALYSIS USER ACTION

- Edit the file ***SS_control.com*** to reflect the current application
- Run the script
- Post-process the results as desired

THIS PAGE INTENTIONALLY BLANK

Part III.
PROCEDURES

THIS PAGE INTENTIONALLY BLANK

3. New Control Procedures

3.1. Overview

This Chapter describes new COMET-AR command language procedures for controlling an analysis which employs interface elements. A Section is dedicated to each of the procedures listed in Table 3.1.

Table 3.1. Outline of Chapter 3: New Control Procedures

Section	Procedure	Function
2	**SS_control**	Controls linear static analysis using interface elements
3	**Initialize**	Initializes required macrosymbols
4	**Post_FE_Stress**	Controls stress recovery for substructures

Currently there is only one control procedure for analyses which employ interface elements, named **SS_control**, and it is limited to linear static analysis. This procedure invokes various additional procedures, some of which must be written by the user. Subordinate procedures are described in subsequent Chapters; examples of user-written procedures are provided as well. The procedure **Initialize** is considered a control procedure in that it defines the macrosymbols which are used to control the analysis. The procedure **Post_FE_Stress** controls the stress recovery operation and calls both master model and finite element procedures.

THIS PAGE INTENTIONALLY BLANK

3.2. Analysis Control - Procedure SS_control

3.2.1. General Description

The procedure named **SS_control** which controls the analysis flow was introduced in Section 2.2.6. For most users and applications, only a call to the control procedure, **SS_control** is needed to perform an analysis. Procedure **SS_control** performs a sequence of calls to other procedures as shown in the Figure 3.1. In the Figure, **iSS** refers to the current substructure and **nSS** refers to the total number of substructures. Only those boxes marked with shaded ends are user-written (or user-modified) procedures; all others are utilities which will be executed automatically.

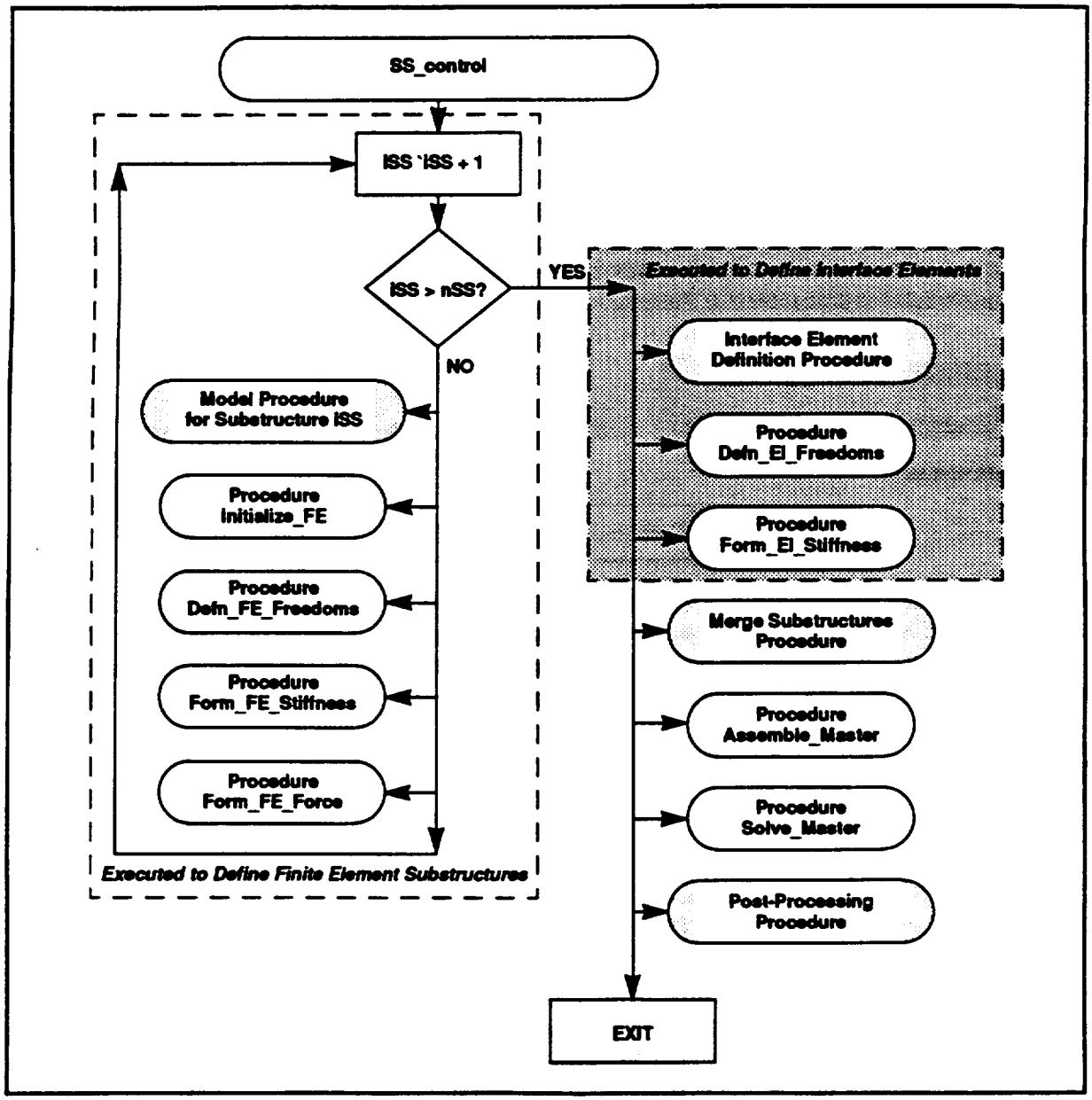


Figure 3.1. Schematic of SS_control: Analysis Control Procedure

3.2.2. Argument Summary

Procedure **SS_control** may be invoked with the COMET-AR *call directive, employing the arguments summarized in Table 3.2, which are described in detail subsequently.

Table 3.2. Procedure SS_control Input Arguments (Logical order)

Argument	Default Value	Description
DEFINE_SS	<false>	Define substructures flag
DEFINE_EI	<false>	Define interface elements flag
MERGE_SS	<false>	Merge substructure flag
ASSEMBLE	<false>	Assemble master system of equations flag
SOLVE	<false>	Solve master system of equations flag
POST_PROCESS	<false>	Post-processing flag

3.2.3. Argument Definitions

In this subsection, the procedure arguments summarized in Table 3.2 are defined in detail. Note that arguments are listed in logical order (*i.e.*, the order of the analysis) rather than alphabetical order.

3.2.3.1 DEFINE_SS Argument

Define Substructures Flag. This flag turns on or off the model definition for all substructures.

Argument syntax:

DEFINE_SS = *define_SS_flag*

where *define_SS_flag* may be set to either <true> (if substructure model definition procedures are to be executed) or <false> (if existing libraries are to be used for the substructure model definitions). When this flag is set to <true>, procedures (named by the macrosymbol **SS_Define_Prc[1:nSS]**) which define the substructures must be provided by the user. (Default value: <false>)

3.2.3.2 DEFINE_EI Argument

Define Interface Elements Flag. This flag turns on or off the definition of all interface elements.

Argument syntax:

DEFINE_EI = *define_EI_flag*

where *define_EI_flag* may be set to either <true> (if interface element definition procedures are to be executed) or <false> (if an existing library is to be used for the interface element definitions). When this flag is set to <true>, a procedure (named by the macrosymbol **EI_Define_Prc**) which defines the interface elements must be provided by the user. (Default value: <false>)

3.2.3.3 MERGE_SS Argument

Merge Substructures Flag. This flag turns on or offthe merging of selected substructures and interface elements into a single, master model.

Argument syntax:

MERGE_SS = *merge_SS_flag*

where *merge_SS_flag* may be set to either<true> (if the merge procedure is to be executed) or <false> (if an existing library is to be used for the merged master model). When this flag is set to<true>, a procedure (named by the macrosymbol **Merge_SS_Prc**) which merges the substructures into a single master model must be provided by the user. (Default value: <false>)

3.2.3.4 ASSEMBLE Argument

Assemble Global System Matrix and Vector Flag. This flag turns off or on assembly of the system stiffness matrix and applied force vector.

Argument syntax:

ASSEMBLE = *assemble_flag*

where *assemble_flag* may be set to either <true> (if an existing assembly utility procedure is to be executed) or <false> (if an existing library contains the assembled stiffness matrix and load vector). This flag will trigger the execution of an existing utility procedure; no additional user action is required. (Default value: <false>)

3.2.3.5 SOLVE Argument

Solve Global System of Equations Flag. This flag turns off or on the solution of the global system of equations which has been reduced in size by the number of constraints applied to the system during assembly. Once a solution for the reduced system has been obtained, the solution vector is expanded to include the constrained degrees of freedom.

Argument syntax:

SOLVE = *solve_flag*

where *solve_flag* may be set to either<true> (if the existing solution utility procedure is to be executed) or <false> (if an existing solution vector is to be used). This flag will trigger the execution of an existing utility procedure; no additional user action is required. (Default value: <false>)

3.2.3.6 POST_PROCESS Argument

Post-processing Flag. This flag turns off or on the post-processing of selected substructures and/or the master model.

Argument syntax:

POST_PROCESS = *post_process_flag*

where *post_process_flag* may be set to either<true> (if the post-processing procedure is to be executed) or <false> (if no post-processing is desired during the current execution). When this flag is set to <true>, a procedure (named by the macrosymbol **Post_Prc**) which provides the post-processing commands must be provided by the user. (Default value: <false>)

3.2.4. Database Input/Output Summary

Procedure **SS_control** can perform a complete analysis, from model definitions through solution post-processing. As such, there are no input datasets for the initial execution of the procedure. In general however, the input and output datasets depend on the arguments (*i.e.*, depend on which portion of the analysis is being performed during the current execution). A summary of the input and output datasets for each phase of the analysis is included in the following Sections. In each of the following Tables, "SS" signifies "SubStructure," "IE" signifies "Interface Element," and "MM" signifies "Master Model." In addition, the variables *mesh*, *ldset*, and *concase*, are defined as mesh number, load set number and constraint case number, respectively.

3.2.4.1 Input Datasets

Table 3.3 contains a list of the datasets required as input for each phase of the analysis. A check mark indicates that the dataset must (or may in some cases) exist. Note that some datasets must appear in more than one database file (*i.e.*, for each substructure). The column labeled "SS_control argument" indicates that the listed argument is set to `<true>` while all others remain`<false>`.

Table 3.3. Input Datasets Required by Procedure SS_control

SS_control argument	Dataset	files		Description
		SS	IE	
DEFINE_SS	None			
DEFINE_EI	CSM.SUMMARY...*mesh*	Y		Model summary for input SS
	NODAL.COORDINATE...*mesh*	Y		SS nodal coordinates
	NODAL.DOF..*concase.mesh*	Y		SS constraints
	NODAL.SPEC_DISP.*ldset..mesh*	Y		SS specified displacements
	NODAL.TRANSFORMATION...*mesh*	Y		Nodal global-to-local transformations
	NODAL.TYPE...*mesh*		Y	Node types
	EltName.DEFINITION...*mesh*	Y	Y	Element definition for input SS
	EltName.ELTYPE...*mesh*		Y	Finite element types along each IE
	EltName.NODSS...*mesh*		Y	SS connected to each node of each IE
	EltName.NORMALS...*mesh*	Y	Y	IE and FE element nodal normals
	EltName.PARAMS...*mesh*		Y	IE parameters
	EltName.SCALE...*mesh*		Y	Scale factor for each IE
	EltName.SCOORD...*mesh*		Y	Path coordinates for nodes on IE
	EltName.SSID...*mesh*		Y	List of SS connected to each IE
	EltName.TANGENT_S...*mesh*		Y	IE path tangent vectors
	EltName.TANGENT_T...*mesh*		Y	IE surface tangent vectors
	EltName.TGC...*mesh*		Y	Computational-to-global transformations
MERGE_SS	CSM.SUMMARY...*mesh*	Y	Y	Model summary
	NODAL.COORDINATE...*mesh*	Y	Y	Nodal coordinates
	NODAL.DOF..*concase.mesh*	Y	Y	Constraints
	NODAL.EXT_FORCE.*ldset..mesh*	Y		Applied nodal forces
	NODAL.SPEC_DISP.*ldset.concase.mesh*	Y		Specified displacements
	NODAL.TRANSFORMATION...*mesh*	Y	Y	Nodal global-to-local transformations
	NODAL.TYPE...*mesh*		Y	Node types

Table 3.3. Input Datasets Required by Procedure SS_control (Continued)

	EltName.DEFINITION...*mesh*	Y	Y	Element definitions
	EltName.PARAMS...*mesh*		Y	IE parameters
	EltName.MATRIX...*mesh*	Y	Y	Element stiffness matrices
ASSEMBLE	*Operation on Master Model; see COMET-AR User's Manual Assembly Processors*			
SOLVE	*Operation on Master Model; see COMET-AR User's Manual Solution Processors*			
POST_PROCESS	*Operations user-defined*			

3.2.4.2 Output Datasets

Table 3.4 contains a list of datasets that may be created or updated by procedure **SS_control**. A check mark indicates that the dataset must (or may in some cases) exist. Note that while the input datasets come from various different database files, each phase of the analysis only writes to a single database file. The column labeled **SS_control argument** indicates that the listed argument is set to `<true>` while all others remain `<false>`.

Table 3.4. Datasets Output From by Procedure SS_control

SS_control argument	Dataset	Files			Description
		SS	IE	MM	
DEFINE_SS	CSM.SUMMARY...*mesh*	Y			Model summary for input SS
	NODAL.COORDINATE...*mesh*	Y			SS nodal coordinates
	NODAL.DOF..*concase.mesh*	Y			SS constraints
	NODAL.EXT_FORCE.*ldset..mesh*	Y			SS applied nodal forces
	NODAL.SPEC_DISP.*ldset..mesh*	Y			SS specified displacements
	NODAL.TRANSFORMATION...*mesh*	Y			SS nodal global-to-local transformations
	EltName.DEFINITION...*mesh*	Y			Element definition for input SS
	EltName.MATRIX...*mesh*	Y			SS Element stiffness matrices
	EltName.NORMALS...*mesh*	Y			SS Element nodal normals
DEFINE_EI	CSM.SUMMARY...*mesh*		Y		Model summary for input SS
	NODAL.COORDINATE...*mesh*		Y		SS nodal coordinates
	NODAL.DOF..*concase.mesh*		Y		SS constraints
	NODAL.TRANSFORMATION...*mesh*		Y		IE nodal global-to-local transformations
	NODAL.TYPE...*mesh*		Y		IE node types
	EltName.DEFINITION...*mesh*		Y		Element definition for each IE
	EltName.ELTYPE...*mesh*		Y		List of finite element types along each IE
	EltName.NODSS...*mesh*		Y		List of SS connected to each IE
	EltName.NORMALS...*mesh*				IE nodal normals
	EltName.PARAMS...*mesh*		Y		IE parameters
	EltName.SCALE...*mesh*		Y		Scale factor for each IE
	EltName.SCOORD...*mesh*		Y		Path coordinates for nodes on IE
	EltName.SSID...*mesh*		Y		List of SS connected to each IE
	EltName.TANGENT_S...*mesh*		Y		IE path tangent vectors
	EltName.TANGENT_T...*mesh*		Y		IE surface tangent vectors
	EltName.TGC...*mesh*		Y		Computational-to-global transformations

Table 3.4. Datasets Output From by Procedure SS_control (Continued)

MERGE_SS	CSM.SUMMARY...*mesh*			Y	Model summary
	NODAL.COORDINATE...*mesh*			Y	Nodal coordinates
	NODAL.DOF..*concase.mesh*			Y	Constraints
	NODAL.EXT_FORCE.*ldset..mesh*			Y	Applied nodal forces
	NODAL.SPEC_DISP.*ldset.concase.mesh*			Y	Specified displacements
	NODAL.TRANSFORMATION...*mesh*			Y	Nodal global-to-local transformations
	EltName.DEFINITION...*mesh*			Y	Element definitions
	EltName.MATRIX...*mesh*			Y	Element stiffness matrices
ASSEMBLE	*Operation on Master Model; see COMET-AR User's Manual Assembly Processors*				
SOLVE	*Operation on Master Model; see COMET-AR User's Manual Solution Processors*				
POST_PROCESS	*Operations user-defined*				

3.2.5. Subordinate Procedures and Processors

3.2.5.1 Subordinate Procedures

A list of procedures invoked directly by procedure **SS_control** is provided in Table 3.5. Documentation of these procedures may be found in the Sections listed.

Table 3.5. Procedures Subordinate to Procedure SS_control

Procedure	Type	Function	Refer to:
Initialize	User-Written	Define required macrosymbols	3.3
SS model generation	User-Written	Generate finite element models for substructures	—
EI_Define	User-Written	Define interface elements	4.2
Defn_EI_Freedoms	Utility	Suppress unstiffened degrees of freedom	4.3
Form_EI_Stiffness	Utility	Form interface element stiffness matrix	4.4
Initialize_FE	Utility	Initialize finite element substructures	5.2
Defn_FE_Freedoms	Utility	Suppress unstiffened degrees of freedom for finite element substructures	5.3
Form_FE_Force	Utility	Form force vector for finite element substructures	5.4
Form_FE_Stiffness	Utility	Form element stiffness matrices for finite element substructures	5.5
Merge_SS	User-Written	Merge finite element substructures and interface element libraries into a single master model	6.2
Assemble_Master	Utility	Assemble single, master system of equations	6.3
Solve_Master	Utility	Solve the master system of equations	6.4

3.2.5.2 Subordinate Processors

Since the **SS_control** procedure may control an analysis from the model generation through post-processing, all COMET-AR processors may be considered subordinate processors.

3.2.6. Current Limitations

SS_control will only perform linear, static, nonadaptive analyses. Additional limitations and assumptions are noted in Section 1.5.

3.2.7. Status and Error Messages

SS_control will not print any status or error messages directly. All messages will be produced by the processors being used in the analysis. For specific error messages, the user should refer to Chapter 7 for the **EI** processors, Chapter 8 for the **MSTR** processor, and the COMET-AR User's Manual (Ref. 3.2-1) for all others.

3.2.8. Examples and Usage Guidelines

3.2.8.1 Example 1: A complete analysis

Listed below is a sample script, including Unix commands, for running a complete analysis, from model definition through post-processing the results. Files contain input runstreams and data as annotated.

```
cp $AR_EIPRC/proclib.gal .
cometar << \endinput
*set echo off
. Set up the procedure library
    *set plib = 28                                      . Set procedure library ldi
    *open 28 proclib.gal /old                           . open procedure library
. Add User files
    *add macros.clp                                     . add user macro definitions
    *add model1.clp                                     . add SS 1 definitions file
    *add model2.clp                                     . add SS 2 definitions file
    *add eidefn.clp                                     . add IE definitions file
    *add util.clp                                       . add special utilities
    *add post.clp                                       . add post-processing file
    *add initialize.clp                                 . add initialization file
. Initialize Macrosymbols
    *call Initialize
. Call Control Procedure
    *call SS_control (    Define_SS      = <true>   ; --  . Define Substructures?
                          Define_EI      = <true>   ; --  . Define Interface Elements?
                          Merge_SS       = <true>   ; --  . Merge Substructures?
                          Assemble       = <true>   ; --  . Assemble global system?
                          Solve          = <true>   ; --  . Solve global system?
                          Post_Process   = <true>   )     . Post-process?
Run Exit
\endinput
```

3.2.9. References

3.2-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

THIS PAGE INTENTIONALLY BLANK

3.3. Macrosymbol Definitions - Procedure Initialize

3.3.1. General Description

Procedure **Initialize** is a procedure template which the user may copy and customize for each application. An example of the procedure is provided at the end of this Section. The macrosymbols defined in procedure **Initialize** are *required* for any analysis using interface elements. Should the user prefer, the macrosymbols may be defined directly in the script file (thus eliminating the need for this procedure).

3.3.2. Macrosymbol Summary

The macrosymbols required by procedure **SS_control** and its subordinate procedures and processors are listed in Table 3.6. It is suggested that the user make use of the procedure template provided, although this is not mandatory. The listed macrosymbols must however, be defined in some manner prior to calling procedure **SS_control**.

Table 3.6. Macrosymbols Required by SS_control and Subordinate Procedures

Macrosymbol	Type	Definition
Num_SS	Integer	Total number of substructures
SS_List[1:NumSS]	Integer array	List of substructure id's (one per substructure)
SS_Lib_Name[1:NumSS]	Character array	List of substructure library (file) names
SS_Define_Prc[1:NumSS]	Character array	List of substructure model definition procedures
SS_ldi[1:NumSS]	Integer array	List of substructure logical device indices
SS_step[1:NumSS]	Integer array	List of substructure load step numbers
SS_con_set[1:NumSS]	Integer array	List of substructure constraint set numbers
SS_load_set[1:NumSS]	Integer array	List of substructure load set numbers
SS_mesh[1:NumSS]	Integer array	List of substructure mesh numbers
EI_Proc	Character	Interface element processor name
EI_Lib_Name	Character	Interface element library (file) name
EI_Define_Prc	Character	Name of procedure for interface element definition
EI_ldi	Integer	Logical device index for interface element library
EI_step	Integer	Load step number
EI_Con_set	Integer	Constraint set number
EI_Load_set	Integer	Load set number
EI_mesh	Integer	Mesh number
MM_Name	Character	Library (file) name for master model
Merge_SS_Prc	Character	Name of procedure for performing the merge
MM_ldi	Integer	Logical device index for master model library
MM_step	Integer	Master model load step number
MM_Con_set	Integer	Master model constraint set number
MM_Load_set	Integer	Master model load set number
MM_mesh	Integer	Master model mesh number
auto_dof_sup	Integer	Automatic drilling freedom suppression flag
auto_drill	Integer	Artificial drilling stiffness flag
auto_triad	Integer	Automatic nodal triad construction flag
Post_Prc	Character	Postprocessing procedure name

3.3.3. Examples and Usage Guidelines

The following example is for an analysis which has a single interface element connecting two substructures. In this case a procedure named **Initialize** is used to define the macrosymbols. The user should referr to the CLAMP manual (Ref. 3.2-1) for an explanation of the*def directive syntax.

```
*procedure Initialize
. Required Macrosymbol Definitions
. Define Substructure parameters:
    *def/i Num_SS            == 2                . Number of substructures
    *def/i SS_List[1:2]      == 1,2              . List of SS id numbers
    *def/p SS_Lib_Name[1]    == model1.dbc       . Library name for SS 1
    *def/p SS_Lib_Name[2]    == model2.dbc       . Library name for SS 2
    *def/p SS_Define_Prc[1] == Model_1           . Model def'n SS 1
    *def/p SS_Define_Prc[2] == Model_2           . Model def'n SS 2
    *def/i SS_ldi[1:2]       == 1,2              . logical device indices
    *def/i SS_step[1:2]      == 0,0              . load step numbers
    *def/i SS_con_set[1:2]   == 1,1              . constraint set numbers
    *def/i SS_load_set[1:2] == 1,1               . load set numbers
    *def/i SS_mesh[1:2]      == 0,0              . mesh numbers
. Define Interface element parameters:
    *def/p EI_Proc           == EI1              . PROCESSOR NAME
    *def/p EI_Lib_Name       == 'interface.dbc'  . Library name
    *def/p EI_Define_Prc     == 'EI_Define'      . I.E. definition procedure
    *def/i EI_ldi            == 3                . logical device index
    *def/i EI_step           == 0                . load step number
    *def/i EI_con_set        == 1                . constraint set number
    *def/i EI_load_set       == 1                . load set number
    *def/i EI_mesh           == 0                . mesh number
. Define Master Model parameters:
    *def/p MM_Name           == 'master.model'   . Library name
    *def/p Merge_SS_Prc      == 'Merge_SS'       . merge procedure name
    *def/i MM_ldi            == 4                . logical device index
    *def/p MM_step           == 0                . load step number
    *def/i MM_con_set        == 1                . constraint set number
    *def/i MM_load_set       == 1                . load set number
    *def/i MM_mesh           == 0                . mesh number
. Drilling freedom suppression flags
    *def/i auto_dof_sup      == <true>           . suppress freedoms?
    *def/i auto_drill        == <false>          . artificial stiffness?
    *def/i auto_triad        == <false>          . automatic nodal triads?
. Miscellaneous macrosymbols:
    *def/p Post_Prc          == 'Post_Test'      . Post processing procedure
*end
```

3.3.4. References

3.3-1 Felippa, Carlos A., *The Computational Structural Mechanics Testbed Architecture: Volume II - Directives.* NASA Contractor Report 178385, February 1989.

3.4. Stress Recovery Control - Procedure Post_FE_Stress

3.4.1. General Description

Procedure **Post_FE_Stress** provides the user the options of recovering substructure element stress resultants (at nodes, integration points, or centroids), substructure smoothed nodal stress resultants (provided element stress resultant data exists in the substructure database), and master model smoothed nodal stress resultants (provided substructure nodal stress resultant data exists in the substructure databases). This control procedure may be executed by the **SS_control** procedure (see Section 3.2) provided the **Post_Prc** macrosymbol has been set (*i.e.*, `*def/p Post_Prc = 'Post_FE_Stress'`).

3.4.2. Argument Summary

Procedure **Post_FE_Stress** may be invoked with the COMET-AR *call directive, employing the arguments summarized in Table 3.2, which are described in detail subsequently.

Table 3.7. Procedure Post_FE_Stress Input Arguments (functional order)

Argument	Default Value	Description
SPLIT_MM	<true>	Flag indicating that substructure results need to be split from the master model.
STRESS_SS	<true>	Flag indicating that substructure stress resultants need to be recovered.
NODAL_STRESS_MM	<true>	Flag indicating that a master model nodal stress object should be formed.

3.4.3. Argument Definitions

In this subsection, the procedure arguments summarized in Table 3.2 are defined in detail. Note that arguments are listed in alphabetical order.

3.4.3.1 SPLIT_MM Argument

Split Substructure data from Master Model Flag. This flag turns on or off the function which takes the solution from the master model and splits out solution vectors for the substructures.

Argument syntax:

SPLIT_MM = *split_mm_flag*

where *split_mm_flag* may be set to either<true> (if the master model solution is to be split into substructure vectors) or <false> (if this step is to be skipped and substructure displacement vectors already exist). This flag will trigger the execution of existing utility procedures; no additional user action is required. (Default value: <true>)

3.4.3.2 STRESS_SS Argument

Calculate Substructure Stress Flag. This flag turns on or off the function which calculates substructure stress resultants based on the solution recovered using the SPLIT_MM argument.

Argument syntax:

STRESS_SS = *stress_ss_flag*

where *stress_ss_flag* may be set to either<true> (if the substructure stress resultants are to be calculated) or <false> (if this step is to be skipped and substructure stress resultants already exist or are not needed). This flag will trigger the execution of existing utility procedures; no additional user action is required. (Default value: <true>)

3.4.3.3 NODAL_STRESS_MM Argument

Calculate Nodal Stress Flag. This flag turns on or off the function which calculates nodal stress resultants based on the element stress resultants recovered using the STRESS_SS argument.

Argument syntax:

NODAL_STRESS_MM = *nodal_stress_mm*

where *nodal_stress_mm* may be set to either <true> (if the smoothed nodal stress resultants are to be calculated) or <false> (if this step is to be skipped and nodal stress resultants already exist or are not needed). When <true>, this flag will create a nodal dataset in the substructure data libraries as well as the master model library. This flag will trigger the execution of existing utility procedures; no additional user action is required. (Default value: <true>)

3.4.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the **MSTR**, **ES**, and **NVST** processors. The **MSTR** processor requirements are documented in Chapter 8 of this document while the **ES** and **NVST** requirements are documented in Ref. 3.2-1.

3.4.5. Subordinate Procedures and Processors

Three procedures may be invoked by **Post_FE_Stress**: **Split_MM**, **Comp_FE_Stress**, and **Comp_Nodal_Stress**. The **Split_MM** procedure calls only the **MSTR** processor. **Comp_FE_Stress** calls only the **ES** processor for the appropriate finite element types. The **Comp_Nodal_Stress** procedure calls the **NVST** and **MSTR** processors.

3.4.6. Current Limitations

Limitations on the procedure usage are, in general, dictated by the limitations on the **MSTR** (see Section 8), **ES** (see Ref. 3.2-1), and **NVST** processors. The user is referred to the documentation appropriate for each processor. The one requirement of the procedure is that the procedure **Initialize** be invoked prior to the call to **Post_FE_Stress** as several of the macrosymbols defined in **Initialize** are used during the calculation of the stress resultants.

3.4.7. Status and Error Messages

Comp_FE_Stress will not print any status or error messages directly. All messages produced by the **MSTR** (see Section 8), **ES** (see Ref. 3.2-1), and **NVST** processors. The user is referred to the documentation appropriate for each processor.

3.4.8. Examples and Usage Guidelines

The **Post_FE_Stress** procedure may be called from within **SS_control**, however, it may also be used in a stand-alone mode. In both cases, the procedure **Initialize** must be called before **Post_FE_Stress** is called. The **Post_FE_Stress** procedure listing follows:

```
*procedure Post_FE_Stress (Split_MM = <true>; Stress_SS = <true>; --
                           Nodal_Stress_MM = <true> )
.
. This procedure is used to control the postprocessing of stress resultants
.
*if <[Split_MM]> /then
  *remark ************
  *remark *** Split Displacements from Master Model to FE Substructures
  *remark ************
  *call Split_MM
*endif
.
 *if <[Stress_SS]> /then
  *remark ************
  *remark *** Compute Stresses for FE Substructures
  *remark ************
  *call Comp_FE_Stress
 *endif
.
 *if <[Nodal_Stress_MM]> /then
  *remark ************
  *remark *** Compute Nodal Stresses for FE Substructures and Merge
  *remark *** Into Master Model
  *remark ************
  *call Comp_Nodal_Stress
 *endif
.
*end
```

3.4.9. References

3.4-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

THIS PAGE INTENTIONALLY BLANK

4. Interface Element Cover Procedures

4.1. Overview

This Chapter describes new COMET-AR command language procedures which control the execution of the interface element processor (processor EI). A Section is dedicated to each of these procedures, which are listed in Table 4.1.

Table 4.1. Outline of Chapter 4 : New Interface Element Cover Procedures

Section	Procedure	Function
2	**EI_Define**	Template for user-written procedure which defines interface elements
3	**Defn_EI_Freedoms**	Automatically suppresses inactive degrees of freedom
4	**Form_EI_Stiffness**	Forms interface element "stiffness" matrix

Cover procedures have been written for each of the functions performed by the **EI** processor. Rather than one procedure which performs all tasks (as has been done with the **ES** processor), several procedures are used, each of which performs an individual task. While the **EI_Define** procedure must be written by the user, the remaining two procedures, **Defn_EI_Freedoms** and **Form_EI_Stiffness**, are utility procedures which are automatically called by the **SS_control** procedure. These two procedures, included here for completeness, require no user action or interaction beyond the definition of the macrosymbols described in Section 3.3.

THIS PAGE INTENTIONALLY BLANK

4.2. Interface Element Definition - Procedure EI_Define

4.2.1. General Description

Procedure **EI_Define** is a procedure template which the user must copy and customize for each application. An example of the procedure **EI_Define**, is listed in Table 4.2.

Table 4.2. Template for User-Defined Procedure EI_Define

```
*procedure EI_Define
. Define Interface Elements
    run  EI1
    . Processor Resets
        reset   ldi          = <EI_ldi>
        reset   mesh         = <EI_mesh>
        reset   step         = <EI_step>
        reset   load_set     = <EI_Load_set>
        reset   cons_set     = <EI_Con_set>
    . Element Definitions
        DEFINE ELEMENTS
        ********************************************************************************
        ELEMENT 1 /DSPLINE=<dspline> /SCALE=<scale>
        *do $i  =1,  <Num_SS>
                *def/i ssid = <SS_id[<$i>]>
                SS  <ssid>   /LDI=<SS_ldi[<ssid>]>/FE/MESH=<SS_mesh[<ssid>]> --
                             /CONS=<SS_con_set[<ssid>]>
                    NODES = <node_list[<ssid>]> /GSPLINE=<SS_geom[<ssid>]>
        *enddo
        ********************************************************************************
*end
```

For the case of multiple interface elements, the lines between the asterisk-filled lines should be repeated for each additional interface element. All of the macrosymbols used above must be defined somewhere in the runstream and must be visible to this procedure (*i.e.*, they must either be global macrosymbols or have been defined in the calling tree for this procedure). If procedure **Initialize** is used, the only additional macrosymbol which must be defined prior to a call to this example of **EI_Define** is **<node_list[1:<Num_SS>]>** which contains as character data a list of the nodes along the interface for each substructure.

The interface element is essentially defined by specifying the substructure edges along which a connection is to be made. This definition may be performed by using the NODES option (as shown in Table 4.2), by specifying a series of coordinates through which a curve may be passed, or by specifying the two nodes at either end of a straight line. In addition, boundary conditions may be applied to either the interface pseudo-nodes or to the alpha-nodes attached to the substructures. The user is referred to Chapter 7 for a complete explanation of the input.

4.2.2. Argument Summary

Users may choose to utilize procedure arguments however, the procedure **SS_control** will then also need to be customized by the user. It is therefore recommended that required input parameters be defined using macrosymbols rather than through procedure arguments.

4.2.3. Argument Definitions

See previous Section.

4.2.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the EI processor being executed. These database requirements are documented in detail in Chapter 7.

4.2.5. Subordinate Processors and Procedures

EI_Define has only one subordinate processor, the EI processor of choice. Normally, there will also be no need for subordinate procedures although the user may wish to define these for particularly complicated models.

4.2.6. Current Limitations

EI_Define is a user-written procedure. Limitations on the procedure usage are dictated by the limitations of the **EI** processor being used in the analysis. These limitations are documented in detail in Chapter 7. Limitations on all **EI** processors are discussed in Section 1.5.

4.2.7. Status and Error Messages

EI_Define will not typically print any status or error messages directly (although the user may choose to insert such messages). Error messages will be produced by the **EI** processor being used in the analysis. The user should refer to Chapter 7 for specific error messages produced by these processors.

4.2.8. Examples and Usage Guidelines

4.2.8.1 Example 1: Define a Single Interface Element connecting Two Substructures.

In this example, substructure 1 resides in library 1 and substructure 2 resides in library 2. Both are finite element substructures. The interface element is written to library 3 and connects nodes 1, 3, 5, and 7 of substructure 1 to nodes 25, 30, 35, 40, 45, and 50 of substructure 2 using cubic spline functions for both the geometry and displacement of a hybrid variational interface element. No constraints have been defined.

```
*procedure EI_Define
. Define Interface Elements
    run  EI1
    . Processor Resets
         reset   ldi        = 3
         reset   mesh       = 0
         reset   step       = 0
         reset   load_set   = 1
         reset   cons_set   = 1
    . Element Definitions
         DEFINE ELEMENTS
         ELEMENT 1 /DSPLINE=3
                 SS  1 /LDI=1 /FE /MESH=0 /CONS=1
                     NODES = 1:7:2 /GSPLINE=3
                 SS  2 /LDI=2 /FE /MESH=0 /CONS=1
                     NODES = 25:50:5 /GSPLINE=3
         END_DEFINE
*end
```

4.2.8.2 Example 2: Define two Interface Elements each connecting Two Substructures.

In this example, substructure 1 resides in library 1, substructure 2 resides in library 2, and substructure 3 resides in library 3. All are finite element substructures. The interface elements are written to library 4. The first hybrid variational interface element connects nodes 1, 3, 5, and 7 of substructure 1 to nodes 25, 30, 35, 40, 45, and 50 of substructure 2 using cubic spline functions for both geometry and displacement. The second element connects nodes 35, 37, 39, 41, 43, and 45 of substructure 1 to nodes 110, 120, 130, 140, 150, and 160 of substructure 3 again using cubic spline functions for both the geometry and displacement of the interface element. No constraints have been defined.

```
*procedure EI_Define
. Define Interface Elements
   run EI1
   . Processor Resets
       reset ldi       = 4
       reset mesh      = 0
       reset step      = 0
       reset load_set  = 1
       reset cons_set  = 1
   . Element Definitions
       DEFINE ELEMENTS
       ELEMENT 1 /DSPLINE=3 /CURVED
              SS 1 /LDI=1 /FE /MESH=0 /CONS=1
                 NODES = 1:7:2 /GSPLINE=3
              SS 2 /LDI=2 /FE /MESH=0 /CONS=1
                 NODES = 25:50:5 /GSPLINE=3
       ELEMENT 2 /DSPLINE=3 /SCALE=10000. /CURVED
              SS 1 /LDI=1 /FE /MESH=0 /CONS=1
                 NODES = 35:45:2 /GSPLINE=3
              SS 3 /LDI=3 /FE /MESH=4 /CONS=2
                 NODES = 110:160:10 /GSPLINE=3
       END_DEFINE
*end
```

4.2.9. References

None.

THIS PAGE INTENTIONALLY BLANK

4.3. Interface Element Drilling Freedom Suppression - Procedure Defn_El_Freedoms

4.3.1. General Description

Procedure **Defn_El_Freedoms** is a utility procedure for performing automatic degree-of-freedom suppression on the new nodes (pseudo-nodes and alpha-nodes) introduced by the interface element(s). It is automatically invoked by the solution control procedure **SS_control**, and requires no user action or interaction beyond the definition of the required macrosymbols (see Section 3.3).

4.3.2. Argument Summary

There are currently no arguments to this procedure. It is assumed that the macrosymbols discussed in Section 3.3 have been defined and exist as macrosymbols visible to the **SS_control** procedure.

4.3.3. Argument Definitions

See previous Section.

4.3.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the EI processor being executed. These dataset requirements are documented in detail in Chapter 7.

4.3.5. Subordinate Processors and Procedures

Defn_El_Freedoms has only one subordinate processor, the EI processor of choice. It has no subordinate procedures.

4.3.6. Current Limitations

Limitations on the procedure usage are dictated by the limitations of the EI processor being used in the analysis. These limitations are documented in detail in Chapter 7. Limitations on all EI processors are discussed in Section 1.5.

4.3.7. Status and Error Messages

Defn_El_Freedoms will not print any status or error messages directly. All messages will be produced by the EI processor being used in the analysis. The user should refer to Chapter 7 for specific error messages produced by these processors.

4.3.8. Examples and Usage Guidelines

The determination of the active degrees-of-freedom for the pseudo-nodes and the alpha-nodes is currently made by the interface element processor during the definition of the elements. In the present implementation, the computational frame for both the pseudo-nodes and the alpha-nodes are defined so that the drilling degree-of-freedom is always the sixth degree-of-freedom. During the element definition, two parameters are set, **Drill_Dof** and **Drill_Sup**, and saved in the **EAT EltName.PARAMS...mesh** (see Section 10.3 for a description of this data object). The parameter **Drill_Dof** is set to six. The parameter **Drill_Sup**, is a flag which indicates whether or not the **Drill_Dof** degree of freedom is to be suppressed.

The decision to suppress the drilling degree-of-freedom is made based on two criteria. First, the suppression need occur only if the interface element connects two substructures, as more than two substructures cannot be coplanar. Second, if the difference between either substructure normal and the average normal is greater than one degree, the drilling degree-of-freedom is not flagged for suppression (*i.e.*, **Drill_Sup** is set to<false>). If the difference between each substructure normal and the average normal is within one degree, the drilling degree-of-freedom is flagged for suppression (*i.e.*, **Drill_Sup** is set to<true>). Note that while the decision to suppress or not suppress degrees-of-freedom is made automatically during the element definition, the procedure **Defn_El_Freedoms** performs the actual suppression of any inactive freedoms.

The **Defn_El_Freedoms** procedure is called automatically. A listing of the procedure has been provided for completeness. The user should refer to Chapter 7 for a full description of the processor input.

```
*procedure Defn_EI_Freedoms
. Suppress inactive degrees of freedom
   run <EI_Proc>
   . Processor Resets
       reset ldi      = <EI_ldi>
       reset mesh     = <EI_mesh>
       reset step     = <EI_step>
       reset load_set = <EI_load_set>
       reset cons_set = <EI_cons_set>
   . Issue command to set active freedoms
       DEFINE FREEDOMS
       STOP
*end
```

4.3.9. References

None.

4.4. Interface Element Stiffness Matrix Generation - Procedure Form_EI_Stiffness

4.4.1. General Description

Procedure **Form_EI_Stiffness** is a utility procedure for forming the interface element stiffness matrices. It is invoked automatically by the solution control procedure **SS_control**, and requires no user action or interaction beyond the definition of the required macrosymbols (see Section 3.3).

4.4.2. Argument Summary

There are currently no arguments to this procedure. It is assumed that the macrosymbols discussed in Section 3.3 have been defined and exist as macrosymbols visible to the **SS_control** procedure.

4.4.3. Argument Definitions

See previous Section.

4.4.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the EI processor being employed. These dataset requirements are documented in detail in Chapter 7.

4.4.5. Subordinate Processors and Procedures

Form_EI_Stiffness has only one subordinate processor, the EI processor of choice. It has no subordinate procedures.

4.4.6. Current Limitations

Limitations on the procedure usage are dictated by the limitations of the EI processor being used in the analysis. These limitations are documented in detail in Chapter 7. Limitations on all EI processors are documented in Section 1.5.

4.4.7. Status and Error Messages

Form_EI_Stiffness will not print any status or error messages directly. All messages will be produced by the EI processor being used in the analysis. The user should refer to Chapter 7 for specific error messages produced by these processors.

4.4.8. Examples and Usage Guidelines

The **Form_EI_Stiffness** procedure, called automatically from within the **SS_control** procedure, will trigger the formation of all element stiffness matrices for elements created by the specified EI processor. As with the **Defn_EI_Freedoms** procedure, the user need only ensure that the macrosymbols defined in procedure **Initialize** (see Section 3.3) are visible to the **SS_control** procedure. A listing of the procedure has been provided for completeness. The user should refer to Chapter 7 for a full description of the processor input.

```
*procedure Form_EI_Stiffness
. Form interface element stiffness matrices
   run <EI_Proc>
   . Processor Resets
       reset ldi      = <EI_ldi>
       reset mesh     = <EI_mesh>
       reset step     = <EI_step>
       reset load_set = <EI_load_set>
       reset cons_set = <EI_cons_set>
   . Issue command to set active freedoms
       FORM STIFFNESS/MATL
       STOP
*end
```

4.4.9. References

None.

5. Finite Element Analysis Procedures

5.1. Overview

This Chapter describes new COMET-AR command language procedures which replace the standard finite element analysis procedures when performing an analysis with interface elements. The use of these procedures is *completely masked from the user* provided procedure **SS_control** is used to perform the analysis. *No user action is required* for these utilities other than that the appropriate macrosymbols be defined.

A Section is dedicated to each of these replacement utility procedures, which are listed in Table 5.1.

Table 5.1. Outline of Chapter 5 : New Finite Element Analysis Procedures

Section	Procedure	Function
2	**Initialize_FE**	Initializes finite element databases
3	**Defn_FE_Freedoms**	Automatically suppresses inactive degrees of freedom
4	**Form_FE_Force**	Forms finite element applied force vector
5	**Form_FE_Stiffness**	Forms finite element stiffness matrices
6	**Comp_FE_Stress**	Computes element stress resultant data
7	**Comp_Nodal_Stress**	Computes smoothed nodal stress resultant data

Most of the procedures discussed in this Chapter use arguments named MESH (which defines the mesh number of the finite element model) and STEP (which defines the nonlinear load step number). While the interface element does not currently have either adaptive or nonlinear capabilities, these two arguments are used to identify data object names within COMET-AR and are included for consistency with existing procedures and processors (*e.g.*, **L_STATIC_1**, **ASM**). Both MESH and STEP will usually be zero (the default values). It should be noted however, that the interface element could be used to couple finite element models for which neither MESH nor STEP are zero provided only a linear analysis is performed. For example, an analyst may wish to perform a coupled linear analysis of two models which have each been through an adaptive analysis resulting in a final nonzero mesh for each model. In this case, the **SS_mesh[1:2]** macrosymbols would be set to nonzero mesh numbers corresponding to the desired mesh numbers in each adaptive analysis.

THIS PAGE INTENTIONALLY BLANK

5.2. Finite Element Initialization - Procedure Initialize_FE

5.2.1. General Description

The initialization process for finite element analysis (in COMET-AR) consists of several phases: initializing data structures; reordering of nodes for optimal bandwidth, fill or profile; generating the proper equation numbers based on the new nodal ordering and constraints; suppressing inactive degrees-of-freedom. With the addition of the interface element capability, the initialization process must be done separately for each substructure and the reordering of nodes or equations must occur after the interface elements have been defined. Thus, the original COMET-AR finite element initialization procedure is no longer adequate and has been split into its components. The data structure initialization is performed by the procedure **Initialize_FE** which executes the ES processors. Other functions are performed later in the analysis using additional new procedures, each of which is documented in later sections. **Initialize_FE** is called automatically by **SS_control** (see Section 3.2) using macrosymbols defined in the **Initialize** procedure (see Section 3.3).

5.2.2. Argument Summary

SS_control invokes procedure **Initialize_FE** with the COMET-AR *call directive, employing the arguments summarized in Table 5.2, which are described in detail subsequently.

Table 5.2. Procedure Initialize_FE Input Arguments

Argument	Default Value	Description
LDI	1	Logical device index
MESH	0	Mesh number of model to be initialized

5.2.3. Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.2 are defined in more detail. Note that arguments are listed in alphabetical order.

5.2.3.1 LDI Argument

Logical Device Index. This argument is the logical unit for the database containing the model data for the substructure being processed.

Argument syntax:

LDI = *ldi*

where the integer *ldi* must be set to an appropriate, active library number. Procedure **SS_control** (see Section 3.2) passes a macrosymbol, **SS_ldi[I]**, through this argument for each substructure I defined by the user. (Default value: 1)

5.2.3.2 MESH Argument

Mesh Number. This argument identifies the number of the finite element mesh to be processed within library *ldi*.

Argument syntax:

MESH = *mesh*

where the integer *mesh* must be set to a valid mesh number. Procedure **SS_control** (see Section 3.2) passes a macrosymbol, **SS_mesh[I]**, through this argument for each substructure I defined by the user. (Default value: None)

5.2.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the ES processor being employed. The dataset requirement for the Initialize command of the ES processors may be found in the COMET-AR User's Manual (Ref. 5.2-1).

5.2.5. Subordinate Processors and Procedures

Initialize_FE has two subordinate procedures, **CSMget** and **ES**. While **Initialize_FE** has no directly subordinate processors, procedure **ES** does execute the ES processor. **CSMget** interacts directly with the database.

5.2.6. Current Limitations

Initialize_FE is a general purpose procedure and the only limitations on its usage are dictated by the limitations of the ES processor being employed. The user should refer to the Element Processor Chapters of the COMET-AR User's Manual (Ref. 5.2-1) for specific processor limitations.

5.2.7. Status and Error Messages

Initialize_FE does not print any status or error messages directly. All messages will be produced by the ES processor being employed. The user should refer to the Element Processor Chapters of the COMET-AR User's Manual (Ref. 5.2-1) for specific processor limitations

5.2.8. Examples and Usage Guidelines

The **Initialize_FE** procedure, called automatically from within the **SS_control** procedure, will initialize all finite element types within a specific substructure. The **SS_control** procedure calls **Initialize_FE** with the appropriate macrosymbols substituted for the two arguments. The user need only ensure that the macrosymbols defined in procedure **Initialize** (see Section 3.3) are visible to the **SS_control** procedure. A listing of the **Initialize_FE** procedure follows.

```
*procedure Initialize_FE ( ldi = 1; mesh = 0 )
. Initialize Finite Element configurations
   . Retrieve element type names and processor names
   *call CSMget ( ldi=[ldi]; mesh=[mesh]; attrib=NET; macro=ES_NET
   *do  $et = 1, <ES_NET>
        *call CSMget ( ldi=[ldi]; mesh=[mesh]; iet=<$et>; --
              attrib=EltTyp; macro=ES_PROC[<$et>])
        *call CSMget ( ldi=[ldi]; mesh=[mesh]; iet=<$et>; --
              attrib=EltPro; macro=ES_TYPE[<$et>])
   *enddo
. Call ES procedure to initialize finite element data objects
   *call ES ( function = 'INITIALIZE'; mesh=[mesh] )
*end
```

5.2.9. References

5.2-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

THIS PAGE INTENTIONALLY BLANK

5.3. Finite Element Drilling Freedom Suppression - Procedure Defn_FE_Freedoms

5.3.1. General Description

The suppression of the drilling freedoms normally occurs in the solution procedure for linear static analysis, procedure **L_STATIC_1** (Ref. 5.2-1). Due to the introduction of the interface element, this solution procedure no longer exists and its functions have been distributed among several procedures. Procedure **Defn_FE_Freedoms** operates on a single finite element substructure and thus is called once for each finite element substructure in the system. This procedure is automatically called from within procedure **SS_control** (see Section 3.2) using macrosymbols defined in procedure **Initialize** (see Section 3.3).

5.3.2. Argument Summary

SS_control invokes procedure **Defn_FE_Freedoms** with the COMET-AR *call directive, employing the arguments summarized in Table 5.2, which are described in detail subsequently.

Table 5.3. Procedure Defn_FE_Freedoms Input Arguments

Argument	Default Value	Description
AUTO_DOF_SUP	<true>	Auto. dof suppression flag
AUTO_DRILL	<false>	Artificial drilling stiffness flag
AUTO_TRIAD	<false>	Auto nodal triads flag
CONSTRAINT_SET	1	Constraint set number
LDI	1	Logical device index
MESH	0	Mesh number of model

5.3.3. Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.2 are defined in detail. Note that arguments are listed in alphabetical order.

5.3.3.1 AUTO_DOF_SUP Argument

Automatic Degree of Freedom Suppression Flag. This argument is a flag which indicates whether or not unstiffened degrees of freedom are to be suppressed automatically.

Argument syntax:

AUTO_DOF_SUP = *auto_dof_sup_flag*

where *auto_dof_sup_flag* may be set to either <true> or <false>. A value of <true> indicates that unstiffened freedoms should be suppressed; a value of <false> indicates that those freedoms should not be suppressed. **SS_control** (see Section 3.2) passes a macrosymbol, **auto_dof_sup**, through this argument. (Default: <true>)

5.3.3.2 AUTO_DRILL Argument

Automatic Drilling Stiffness Flag. This argument is a flag which indicates whether or not artificial stiffness should be added to unstiffened drilling degrees of freedom.

Argument syntax:

AUTO_DRILL = *auto_drill_flag*

where *auto_drill_flag* may be set to either `<true>` or `<false>`. A value of `<true>` indicates that artificial stiffness should be added to unstiffened drilling degrees of freedom. A value of `<false>` indicates that no artificial stiffness should be added. **SS_control** (see Section 3.2) passes a macrosymbol, **auto_drill**, through this argument. (Default: `<false>`)

5.3.3.3 AUTO_TRIAD Argument

Automatic Triad Generation Flag. This argument is a flag which indicates whether or not average nodal normal triads should be generated. Once generated, these triads define the new computational reference frames for the finite element nodes.

Argument syntax:

AUTO_TRIAD = *auto_triad_flag*

where *auto_triad_flag* may be set to either `<true>` or `<false>`. A value of `<true>` indicates that new nodal normal triads should be computed. A value of `<false>` indicates that no new triads should be formed. **SS_control** (see Section 3.2) passes a macrosymbol, **auto_triad**, through this argument. (Default: `<false>`)

5.3.3.4 CONSTRAINT_SET Argument

Constraint set number. This argument identifies the constraint set number for the substructure being processed.

Argument syntax:

CONSTRAINT_SET = *constraint_set*

where the integer *constraint_set* must be set to a valid constraint set number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_con_set[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: `1`)

5.3.3.5 LDI Argument

Logical Device Index. This argument is the logical unit for the database containing the model data for the substructure being processed.

Argument syntax:

LDI = *ldi*

where the integer *ldi* must be set to an appropriate active library number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_ldi[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: 1)

5.3.3.6 MESH Argument

Mesh Number. This argument identifies the number of the mesh to be processed within library *ldi*.

Argument syntax:

MESH = *mesh*

where the integer *mesh* must be set to a valid mesh number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_mesh[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: 0)

5.3.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the subordinate processors and procedures. These dataset requirements are documented in the appropriate sections of the COMET-AR User's Manual (Ref. 5.2-1).

5.3.5. Subordinate Processors and Procedures

Defn_FE_Freedoms calls the utility procedure **ES** and executes the processor **COP**. If the AUTO_TRIAD argument has been set to <true>, then the processor **TRIAD** will also be executed. The subordinate procedure and processors are documented in the COMET-AR User's Manual (Ref. 5.2-1).

5.3.6. Current Limitations

Limitations on the procedure usage are dictated by the limitations of the **ES**, **TRIAD**, and **COP** processors. These limitations are documented in the COMET-AR User's Manual (Ref. 5.2-1).

5.3.7. Status and Error Messages

Defn_FE_Freedoms will not print any status or error messages directly. All messages will be produced by the **ES**, **TRIAD**, and **COP** processors. The user should refer to the COMET-AR User's Manual (Ref. 5.2-1) for specific error messages produced by these processors.

5.3.8. Examples and Usage Guidelines

The macrosymbols **auto_dof_sup**, **auto_drill**, and **auto_triad** (defined within procedure **Initialize**) determine which functions are performed within **Defn_FE_Freedoms**. The **Defn_FE_Freedoms** procedure is called automatically by the **SS_control** procedure. A listing of **Defn_FE_Freedoms** follows.

```
*procedure Defn_FE_Freedoms ( auto_dof_sup=<true>; auto_drill=<false>; --
                              auto_triad=<false>; constraint_set=1; ldi=1; mesh=0 )
. Perform drilling stiffness suppression as specified
. Define nodal flags for drilling stiffness (AUTO_DRILL Option)
   *def/i auto_drill[1:3]       = 0
   *def/i auto_drill[1]         = [auto_drill]
   *def/i auto_drill_o          = <auto_drill[1]>       . Option
   *def/i auto_drill_t          = <auto_drill[2]>       . Tolerance (degrees
   *def/i auto_drill_s          = <auto_drill[3]>       . scale factor
   *if < <auto_drill_o> > /then
        *call ES ( function = 'DEFINE NORMALS'; mesh=[mesh] )
        *call ES ( function = 'DEFINE DRIL_FLAGS'; mesh=[mesh] --
                       drill_tol = <auto_drill_t> )
   *endif
. Replace Current Triads with Avg. Normal-Aligned Triads (AUTO_TRIAD)
   *def/i auto_triad[1:2]       = 0
   *def/i auto_triad[1]         = [auto_triad]
   *def/i auto_triad_o          = <auto_triad[1]>       . Option
   *def/i auto_triad_t          = <auto_triad[2]>       . Tolerance (degrees)
   *if < <auto_triad_o> > /then
        *call ES ( function = 'DEFINE NORMALS'; mesh=[mesh] )
        *call ES ( function = 'DEFINE DRIL_FLAGS'; mesh=[mesh] --
        Run Triad
             LDI  = [ldi]
             MESH = [mesh]
             GO
   *endif
. Suppress Un-stiffened Degrees of Freedom (AUTO_DOF_SUP)
   *def/i auto_dof[1:2]         = 0
   *def/i auto_dof[1]           = [auto_dof_sup]
   *def/i auto_dof_o            = <auto_dof[1]>         . Option
   *def/i auto_dof_t            = <auto_dof[2]>         . Tolerance (degrees)
   *if < <auto_dof_o> > /then
        *call ES ( function = --
        'DEFINE FREEDOMS [ldi], NODAL.ELT_DOF..[constraint_set].[mesh]'; --
        mesh=[mesh]; drill_tol=<auto_dof_t> )
   *endif
. Construct Nodal DOF Table (Number Equations)
   Run  COP
        MODEL [ldi] CSM.SUMMARY...[mesh]
        *if < [auto_dof_sup] > /then . UPDATE
             DOF_SUPPRESS INPUT =[ldi],NODAL.ELT_DOF..[constraint_set].[mesh] --
             DOFDAT=[ldi] [constraint_set] [mesh]
        *endif
        SELECT OLD [ldi] [constraint_set] [mesh] DOFDAT
        CONSTRAIN
        RESET ZERO = NO
        RESET NONZERO = NO
        DONE
        STOP
*end
```

5.3.9. References

5.3-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

5.4. Finite Element Consistent Load Definition - Procedure Form_FE_Force

5.4.1. General Description

Procedure **Form_FE_Force** calculates consistent nodal forces based on input element and nodal forces. The procedure operates on a single finite element substructure and thus is called once for each finite element substructure in the system. This procedure is called automatically from within procedure **SS_Control** (see Section 3.2) using macrosymbols defined in the **Initialize** procedure (see Section 3.3).

5.4.2. Argument Summary

SS_control invokes procedure **Form_FE_Force** with the COMET-AR *call directive, employing the arguments summarized in Table 5.2, which are described in detail subsequently.

Table 5.4. Procedure Form_FE_Force Input Arguments

Argument	Default Value	Description
LDI	None	Logical device index
LOAD_SET	None	Load set number
MESH	None	Mesh number of model
STEP	None	Nonlinear load step number

5.4.3. Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.2 are defined in detail. Note that arguments are listed in alphabetical order.

5.4.3.1 LDI Argument

Logical Device Index. This argument is the logical unit for the database containing the model data for the substructure being processed.

Argument syntax:

LDI = *ldi*

where the integer *ldi* must be set to an appropriate active library number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_ldi[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3.(Default value: None)

5.4.3.2 LOAD_SET Argument

Load set number. This argument identifies the load set number for the substructure being processed.

Argument syntax:

LOAD_SET = *load_set*

where the integer *load_set* must be set to a valid load set number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_load_set[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: None)

5.4.3.3 MESH Argument

Mesh Number. This argument identifies the number of the mesh to be processed within the library *ldi*.

Argument syntax:

MESH = *mesh*

where the integer *mesh* must be set to a valid mesh number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_mesh[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: None)

5.4.3.4 STEP Argument

Nonlinear load step number. This argument identifies the load step number for the substructure being processed.

Argument syntax:

STEP = *load_step*

where the integer *load_step* must be set to a valid load set number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_step[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: None)

5.4.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the subordinate processors and procedures. These dataset requirements are documented in the appropriate sections of the COMET-AR User's Manual (Ref. 5.2-1).

5.4.5. Subordinate Processors and Procedures

Form_FE_Force calls the utility procedure **FORCE** which in turn calls the utility procedure **ES**. The **ES** procedure executes finally the **ES** processor. These procedures and processor are documented in the COMET-AR User's Manual (Ref. 5.2-1).

5.4.6. Current Limitations

Form_FE_Force is a general purpose procedure. Limitations on the procedure usage are dictated by the limitations of the **ES** processors. These limitations are documented in the COMET-AR User's Manual (Ref. 5.2-1).

5.4.7. Status and Error Messages

Form_FE_Force will not print any status or error messages directly. All messages will be produced by the **ES** processors. The user should refer to the COMET-AR User's Manual (Ref. 5.2-1) for specific error messages produced by these processors.

5.4.8. Examples and Usage Guidelines

The **Form_FE_Force** procedure, called automatically from within the **SS_control** procedure, calls a second procedure, **FORCE**, which forms a nodal force vector given input element and nodal loads by executing the **ES** processor. The **SS_control** procedure calls **Form_FE_Force** with the appropriate macrosymbols substituted for the arguments. The user need only ensure that the macrosymbols defined in procedure **Initialize** (see Section 3.3) are visible to the **SS_control** procedure. A listing of the **Form_FE_Force** procedure follows.

```
*procedure Form_FE_Force ( step; load_set; ldi; mesh)
.
  *call FORCE ( type         = EXTERNAL                                    ; --
                ldi          = [ldi]                                       ; --
                input_force  = [ldi],NODAL.SPEC_FORCE.[load_set].0.[mesh] ; --
                output_force = [ldi], NODAL.EXT_FORCE.[load_set].0.[mesh] ; --
                load_set     = [load_set]                                  ; --
                load_factor  = 1,0                                         ; --
                mesh         = [mesh]                                      )
*end
```

5.4.9. References

5.4-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

THIS PAGE INTENTIONALLY BLANK

5.5. Finite Element Stiffness Matrix Formation - Procedure Form_FE_Stiffness

5.5.1. General Description

Procedure **Form_FE_Stiffness** calculates the element stiffness matrices for finite element substructures. The procedure operates on a single finite element substructure and thus is called once for each finite element substructure in the system. The procedure is called automatically from within procedure **SS_Control** (see Section 3.2) using macrosymbols defined in the **Initialize** procedure (see Section 3.3).

5.5.2. Argument Summary

SS_control invokes procedure **Form_FE_Stiffness** with the COMET-AR *call directive, employing the arguments summarized in Table 5.2, which are described in detail subsequently.

Table 5.5. Procedure Form_FE_Stiffness Input Arguments

Argument	Default Value	Description
LDI	None	Logical device index
LOAD_SET	None	Load set number
MESH	None	Mesh number of model
STEP	None	Nonlinear load step number

5.5.3. Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.2 are defined in detail. Note that arguments are listed in alphabetical order.

5.5.3.1 LDI Argument

Logical Device Index. This argument is the logical unit for the database containing the model data for the substructure being processed.

Argument syntax:

LDI = *ldi*

where the integer *ldi* must be set to an appropriate active library number. **SS_control** (see Section 3.2) passes a macrosymbol, SS_ldi[i], through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: None)

5.5.3.2 LOAD_SET Argument

Load set number. This argument identifies the load set number for the substructure being processed.

Argument syntax:

LOAD_SET = *load_set*

where the integer *load_set* must be set to a valid load set number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_load_set[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: None)

5.5.3.3 MESH Argument

Mesh Number. This argument identifies the number of the mesh to be processed within the library *ldi*.

Argument syntax:

MESH = *mesh*

where the integer *mesh* must be set to a valid mesh number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_mesh[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: None)

5.5.3.4 STEP Argument

Nonlinear load step number. This argument identifies the load step number for the substructure being processed.

Argument syntax:

STEP = *load_step*

where the integer *load_step* must be set to a valid load set number. **SS_control** (see Section 3.2) passes a macrosymbol, **SS_step[I]**, through this argument for each substructure I. This macrosymbol is one of the required macrosymbols discussed in Section 3.3. (Default value: None)

5.5.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the subordinate processors and procedures. These dataset requirements are documented in the appropriate sections of the COMET-AR User's Manual (Ref. 5.2-1).

5.5.5. Subordinate Processors and Procedures

Form_FE_Stiffness calls the utility procedure ES which executes the ES processor. The procedures and processor are documented in the COMET-AR User's Manual (Ref. 5.2-1).

5.5.6. Current Limitations

Form_FE_Stiffness is a general purpose procedure. Limitations on the procedure usage are dictated by the limitations of the ES processors. These limitations are documented in the COMET-AR User's Manual (Ref. 5.2-1).

5.5.7. Status and Error Messages

Form_FE_Stiffness will not print any status or error messages directly. All messages will be produced by the ES processors. The user should refer to the COMET-AR User's Manual (Ref. 5.2-1) for specific error messages produced by these processors.

5.5.8. Examples and Usage Guidelines

The **Form_FE_Stiffness** procedure, called automatically from within the **SS_control** procedure (see Section 3.2), calls a second procedure, **ES** (Ref. 5.2-1), which calls the specific finite element processor to compute the element stiffness matrices. The **SS_control** procedure calls **Form_FE_Stiffness** with the appropriate macrosymbols substituted for the arguments. The user must ensure that the macrosymbols defined in procedure **Initialize** (see Section 3.3) are visible to the **SS_control** procedure. A listing of the **Form_FE_Stiffness** procedure follows.

```
*procedure Form_FE_Stiffness ( step; load_set; ldi; mesh)
.
  *call ES ( function     = 'FORM STIFFNESS/MATL'   ; --
             stiffness    = MATL_STIFFNESS          ; --
             ldi          = [ldi]                   ; --
             load_set     = [load_set]              ; --
             step         = [step]                  ; --
             mesh         = [mesh]                  )
*end
```

5.5.9. References

5.5-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

THIS PAGE INTENTIONALLY BLANK

5.6. Finite Element Stress Recovery - Procedure Comp_FE_Stress

5.6.1. General Description

Procedure **Comp_FE_Stress** calculates the finite element stress resultants for each element in each substructure. The procedure may be called directly or may be invoked through a call to the **Post_FE_Stress** procedure (see Section 3.4).

5.6.2. Argument Summary

The procedure **Comp_FE_Stress** may be invoked with the COMET-AR *call directive employing the arguments summarized in Table 5.2 (which are described in detail subsequently), or called through the procedure **Post_FE_Stress** provided the default values for all of the arguments listed are acceptable. If any default value requires modification, **Comp_FE_Stress** should be invoked directly so that the proper argument value may be passed.

Table 5.6. Procedure Comp_FE_Stress Input Arguments

Argument	Default Value	Description
DELETE	<true>	Mark stress resultant object for deletion from database
LOCATION	INTEG_PTS	Location at which element stress resultants are calculated
MESH	0	Mesh number of model
STR_DIRECTION	1	Stress direction

5.6.3. Argument Definitions

In this subsection, the procedure arguments summarized in Table 5.2 are defined in detail. Note that arguments are listed in alphabetical order.

5.6.3.1 DELETE Argument

Delete Existing Dataset Flag. This argument flags existing stress data objects for deletion.

Argument syntax:

DELETE = *delete_flag*

where the integer *delete_flag* must be set to either <true> (if an existing stress object is to be deleted) or <false> (if is an existing stress object is to remain) Section 3.3. (Default value: None)

5.6.3.2 LOCATION Argument

Stress Resultant Location. This argument identifies the location within each element at which the stress resultants are calculated for each substructure.

Argument syntax:

LOCATION = *location*

where the character string *location* must be set to `CENTROIDS`, `NODES`, or `INTEG_PTS`. (Default value: `INTEG_PTS`)

5.6.3.3 MESH Argument

Mesh Number. This argument identifies the number of the mesh to be processed.

Argument syntax:

MESH = *mesh*

where the integer *mesh* must be set to a valid mesh number (*i.e.*, a mesh number which exists in the each of the substructure libraries). (Default value: `0`)

5.6.3.4 STR_DIRECTION Argument

Stress Direction. This argument identifies the direction in which the stress resultants are to be computed.

Argument syntax:

STR_DIRECTION = *direction*

where the *direction* may be either character or integer and must be set to a valid stress direction as defined in Section 7.2.6.24 of Ref. 5.2-1. (Default value: `1` or `GLOBAL X`)

5.6.4. Database Input/Output Summary

All database input and output requirements are imposed by the **ES** processor. These requirements are documented in detail in Ref. 5.2-1.

5.6.5. Subordinate Processors and Procedures

The **Comp_FE_Stress** procedure has two subordinate procedures: **CSMget** and **STRESS**. The procedure **CSMget** accesses the **CSM** data object and the procedure **STRESS** calls the **ES** processor to calculate the stress resultants. The user is referred to Ref. 5.2-1 for details on these procedures and processor.

5.6.6. Current Limitations

Limitations on the procedure usage are dictated by the limitations of the **ES** processor and the **CSMget** and **STRESS** procedures. The user is referred to Ref. 5.2-1 for details on these procedures and processor.

5.6.7. Status and Error Messages

Comp_FE_Stress does not print any error messages directly. All messages will be produced by the ES processor or the **CSMget** and **STRESS** procedures. The user is referred to Ref. 5.2-1 for details on these procedures and processor.

5.6.8. Examples and Usage Guidelines

The **Comp_FE_Stress** procedure may be invoked through a call to the **Post_FE_Stress** procedure or through a direct call. The user need only ensure that the macrosymbols defined in the procedure **Initialize** are visible to the **Comp_FE_Stress** procedure. A listing of the procedure follows.

```
*procedure Comp_FE_Stress ( Location = INTEG_PTS; str_direction = 1; --
                            mesh = 0; Delete = <true> )
*do $k = 1,<Num_ss>
     *def/i ldi = 1
     *open <ldi> <SS_Lib_Name[<$k>]>
.    -------------------------------
.    Get Element Type Information
.    -------------------------------
     *call CSMget ( ldi=<ldi>; mesh=[mesh]; attrib=NET; --
                    macro=ES_NET )
     *do $et = 1, <ES_NET>
         *call CSMget ( ldi=<ldi>; mesh=[mesh] ; iet=<$et>; --
                        attrib=EltPro; macro=ES_PROC[<$et>] )
         *call CSMget ( ldi=<ldi>; mesh=[mesh] ; iet=<$et>; --
                        attrib=EltTyp; macro=ES_TYPE[<$et>] )
     *enddo
     *def/i i = <SS_Load_Set[<$k>]>
     *def/i j = <SS_Con_Set[<$k>]>
.    Delete Existing files
     *if <[Delete]> /then
         *Find Dataset <ldi> E*.STRESS.<i>.<j>.[mesh] /seq=iseq[1]
         *Find Dataset <ldi> E*.STRAIN.<i>.<j>.[mesh] /seq=iseq[2]
         *Find Dataset <ldi> E*.STRAIN_ENERGY.<i>.<j>.[mesh] /seq=iseq[3]
         *do $l = 1,3
            *if <iseq[<$l>]> /ne 0 > /then
               *Delete <ldi> <iseq[<$l>]>
            *endif
         *enddo
     *endif
     *call STRESS ( STRESS = <ldi>, E*.STRESS.<i>.<j>.[mesh] ; --
           STRAIN = <ldi>, E*.STRAIN.<i>.<j>.[mesh]; --
           STRAIN_ENERGY = <ldi>, E*.STRAIN_ENERGY.<i>.<j>.[mesh]; --
           DISPLACEMENT  = <ldi>, NODAL.DISPLACEMENT.<i>.<j>.[mesh]; --
           MESH          = [mesh]; --
           LOCATION = [location]; DIRECTION = [str_direction] )
 *enddo
 *close
*end
```

5.6.9. References

5.6-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

THIS PAGE INTENTIONALLY BLANK

5.7. Compute Smoothed Nodal Stresses - Procedure Comp_Nodal_Stress

5.7.1. General Description

Procedure **Comp_Nodal_Stress** calculates a set of weighted average nodal stress resultants for each node in each substructure and creates a single **NAT** data object in the master model library which contains the master model nodal stress resultants. The procedure may be called directly or may be invoked through a call to the **Post_FE_Stress** procedure (see Section 3.4).

5.7.2. Argument Summary

There are currently no arguments to this procedure. It is assumed that the macrosymbols discussed in Section 3.3 have been defined and exist as macrosymbols visible to the procedure.

5.7.3. Argument Definitions

See previous Section.

5.7.4. Database Input/Output Summary

All database input and output requirements are imposed by the **NVST** processor. These requirements are documented in Ref. 5.2-1.

5.7.5. Subordinate Processors and Procedures

The **Comp_Nodal_Stress** procedure has only one subordinate processor, **NVST**. The user is referred to Ref. 5.2-1 for details on this processor.

5.7.6. Current Limitations

Limitations on the procedure usage are dictated by the limitations of the **NVST** processor. The user is referred to Ref. 5.2-1 for details on this processor.

5.7.7. Status and Error Messages

Comp_Nodal_Stress does not print any error messages directly. All messages will be produced by the **NVST** processor. The user is referred to Ref. 5.2-1 for details on this processor.

5.7.8. Examples and Usage Guidelines

The **Comp_Nodal_Stress** procedure may be invoked either through a call to the **Post_FE_Stress** procedure or through a direct call. The user need only ensure that the macrosymbols defined in the procedure **Initialize** are visible to the **Comp_Nodal_Stress** Procedure. A listing of the procedure follows.

```
*procedure Comp_Nodal_Stress

*open <MM_ldi> <MM_Name>
.
*do $i = 1,<Num_ss>
    *open <SS_ldi[<$i>]> <SS_Lib_Name[<$i>]>
    *if < <$i> /eq 1 > /then
        *def/i iupdat = 0
    *else
        *def/i iupdat = 1
    *endif
    run NVST
        SET /lib  = <SS_ldi[<$i>]>      /olib = <MM_ldi> /update = <iupdat>
        SET /load = <SS_Load_Set[<$i>]> /con  = <SS_Con_Set[<$i>]>
        stop
*enddo
*close
*end
```

5.7.9. References

5.7-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

6. Master Model Analysis Procedures

6.1. Overview

This Chapter describes new COMET-AR command language procedures which control the generation and analysis of the Master Model. The Master Model Processor, **MSTR**, takes as input the substructure and interface element definitions (*i.e.*, node locations, connectivities, loads, boundary conditions) and then renumbers all of the input nodes (including pseudo-nodes and alpha-nodes) sequentially, renumbers the elements, rewrites the element connectivities, and copies all the data required for the solution into a single library file. The resulting master model therefore contains both finite elements (possibly several different types) and interface elements. The element stiffness matrices may then be assembled using an available assembly processor (*e.g.*, processor **ASM**) and the resulting global system of equations may be solved using an available solver (*e.g.*, processor **PVSNP**). A section is dedicated to each of the master model analysis procedures summarized in Table 6.1.

Table 6.1. Outline of Chapter 6: Master Model Analysis Procedures

Section	Procedure	Function
2	Merge_SS	Template for user-written procedure which generates the master model
3	Assemble_Master	Assembles the master model stiffness matrix and load vector
4	Solve_Master	Solves the global system of equations

THIS PAGE INTENTIONALLY BLANK

6.2. Master Model Generation - Procedure Merge_SS

6.2.1. General Description

The finite element substructures and the interface elements are merged into a single master model through the use of the **Merge_SS** procedure which calls the **MSTR** processor (see Section 8.2 for details on this processor) and which is called automatically by the **SS_control** procedure (see Section 3.2). The procedure **Merge_SS** may either be used as is (in which case all of the defined substructures are merged with all of the interface elements) or it may be used as a template (so that only selected substructures are merged). Table 6.2 is a listing of the **Merge_SS** procedure template.

Table 6.2. Template for User-Defined Procedure Merge_SS

```
*procedure Merge_SS
. Merge User-specified substructures into a single library
    Run MSTR
    . Define Substructures that will be merged
        DEFINE SUBSTRUCTURES
        *do  $j = 1, <Num_SS>
        . Finite Element Substructures
            SUBstructure <SS_List[<$j>]> /fe
            Library          = <SS_ldi[<$j>]>        . SS library numbers
            Mesh             = <SS_mesh[<$j>]>       . SS mesh numbers
            Load_set         = <SS_load_set[<$j>]>   . SS load set numbers
            Constraint_case  = <SS_con_set[<$j>]>    . SS constraint case numbers
            Load_step        = <SS_step[<$j>]>       . SS Load step numbers
        *enddo
        . Interface Element Substructure
            SUBstructure <<SS_List[<$j>]>+1> /ie
            Library          = <EI_ldi>              . Interface Element library
            Mesh             = <EI_mesh>             . Interface Element mesh
            Load_set         = <EI_load_set>         . Interface Element load set
            Constraint_case  = <EI_con_set>          . Interface Elem. constraint case
            Load_step        = <EI_step>             . Interface Element Load step
        END_DEFINE
    . Perform the Merge operation
        MERGE <SS_List[1:<Num_SS>]>,<<Num_SS>+1>
            File             = <MM_name>             . Master model library file name
            Library          = <MM_ldi>              . Master model library number
            Mesh             = <MM_mesh>             . Master model mesh
            Load_set         = <MM_load_set>         . Master model load set
            Constraint_case  = <MM_con_set>          . Master model constraint case
            Load_step        = <MM_step>             . Master model Load step
        END_MERGE
    STOP
*end
```

6.2.2. Argument Summary

It is recommended that required input parameters be defined using the macrosymbols defined in procedure **Initialize** (see Section 3.3) rather than through new procedure arguments. Users may choose to utilize new procedure arguments however, the procedure **SS_control** will then have to be customized by the user.

6.2.3. Argument Definitions

See previous Section.

6.2.4. Database Input/Output Summary

All database input and output requirements are imposed by the **MSTR** processor. These requirements are documented in detail in Chapter 8.

6.2.5. Subordinate Processors and Procedures

The **Merge_SS** procedure has only one subordinate processor, **MSTR**. While there are also no subordinate procedures in the template provided, the user may find it useful to define subordinate procedures, especially for complex models.

6.2.6. Current Limitations

Merge_SS is a user-written procedure. Limitations on the procedure usage are dictated by the limitations of the **MSTR** processor. These limitations are documented in detail in Chapter 8. Limitations on the use of interface elements in general are documented in Section 1.5.

6.2.7. Status and Error Messages

Merge_SS does not usually print any status or error messages directly (although the user may choose to include such messages). Most messages will be produced by the **MSTR** processor; these error messages are documented in Chapter 8.

6.2.8. Examples and Usage Guidelines

The **Merge_SS** procedure template shown in Table 6.2 has been compiled into the procedure library, **$AR_EIPRC/proclib.gal**. If all existing substructures are to be merged into a single master model, no user action is required beyond the definition of the **Merge_SS_Prc** macrosymbol (see Section 3.3).

6.2.8.1 Example 1: Merge all existing substructures into a single model.

The following procedure merges all existing substructures, including interface elements, into a single master model. All libraries associated with the substructures and the interface elements must be opened prior to calling this procedure. The user should refer to Chapter 8 for details of processor **MSTR** input.

```
*procedure Merge_SS
. Merge User-specified substructures into a single library
   Run MSTR
   . Define Substructures that will be merged
      DEFINE SUBSTRUCTURES
      *do $j = 1, <Num_SS>
      . Finite Element Substructures
         SUBstructure <SS_List[<$j>]> /fe
         Library         = <SS_ldi[<$j>]>      . SS library numbers
         Mesh            = <SS_mesh[<$j>]>     . SS mesh numbers
         Load_set        = <SS_load_set[<$j>]> . SS load set numbers
         Constraint_case = <SS_con_set[<$j>]>  . SS constraint case numbers
         Load_step       = <SS_step[<$j>]>     . SS load step numbers
      *enddo
      . Interface Element Substructure
         SUBstructure <<SS_List[<$j>]>+1> /ie
         Library         = <EI_ldi>            . Interface Element library
         Mesh            = <EI_mesh>           . Interface Element mesh
         Load_set        = <EI_load_set>       . Interface Element load set
         Constraint_case = <EI_con_set>        . Interface Element constraint case
         Load_step       = <EI_step>           . Interface Element load step
      END_DEFINE
   . Perform the Merge operation
      MERGE <SS_List[1:<Num_SS>]>,<<Num_SS>+1> . MERGE ALL SUBSTRUCTURES
         File            = <MM_name>           . Master model library file name
         Library         = <MM_ldi>            . Master model library number
         Mesh            = <MM_mesh>           . Master model mesh
         Load_set        = <MM_load_set>       . Master model load set
         Constraint_case = <MM_con_set>        . Master model constraint case
         Load_step       = <MM_step>           . Master model load step
      END_MERGE
   STOP
*end
```

6.2.8.2 Example 2: Merge only three selected substructures into a single model.

The following procedure merges substructures 1 and 3 and the existing interface elements into a single master model. All libraries associated with the substructures and the interface elements must be opened prior to calling this procedure. The user should refer to Chapter 8 for details on processor **MSTR** input.

```
*procedure Merge_SS
. Merge User-specified substructures into a single library
    Run MSTR
        . Define Substructures that will be merged
        DEFINE SUBSTRUCTURES
        . Finite Element Substructures
            SUBstructure 1 /fe
            Library          = 1            . SS 1 library number
            Mesh             = 0            . SS 1 mesh number
            Load_set         = 1            . SS 1 load set number
            Constraint_case = 1             . SS 1 constraint case number
            Load_step        = 0            . SS 1 load step number
            SUBstructure 3 /fe
            Library          = 4            . SS 3 library number
            Mesh             = 0            . SS 3 mesh number
            Load_set         = 2            . SS 3 load set number
            Constraint_case = 2             . SS 3 constraint case number
            Load_step        = 0            . SS 3 load step number
        . Interface Element Substructure
            SUBstructure 4 /ie
            Library          = 8            . Interface Element library
            Mesh             = 0            . Interface Element mesh
            Load_set         = 1            . Interface Element load set
            Constraint_case = 1             . Interface Element constraint case
            Load_step        = 0            . Interface Element load step
    . Perform the Merge operation
        MERGE 1,3,4
            File             = 'master.dbc' . Master model library file name
            Library          = 3            . Master model library number
            Mesh             = 0            . Master model mesh
            Load_set         = 1            . Master model load set
            Constraint_case = 1             . Master model constraint case
            Load_step        = 0            . Master model Load step
        END_MERGE
    STOP
*end
```

6.2.9. References

None.

6.3. Master Model Assembly - Procedure Assemble_Master

6.3.1. General Description

The assembly of the global stiffness matrix and load vector are normally carried out within the solution procedure (*e.g.*, **L_STATIC_1**). Due to the introduction of the interface elements, this solution procedure can no longer be used and the functions performed within it have been placed within different, individual procedures. Some of these new procedures have already been discussed (*e.g.*, **Defn_FE_Freedoms**). The new procedure for performing system matrix and vector assembly is discussed in this Section. **Assemble_Master** is called automatically by **SS_control** (see Section 3.2) using macrosymbols defined in the **Initialize** procedure (see Section 3.3).

6.3.2. Argument Summary

SS_control invokes procedure **Assemble_Master** with the COMET-AR *call directive, employing the arguments summarized in Table 6.3, which are described in detail subsequently.

Table 6.3. Procedure Assemble_Master Input Arguments

Argument	Default Value	Description
ASM_STIFFNESS	STRUCTURE.MATL_STIFFNESS	Assembled matrix data object
CONSTRAINT_SET	1	Constraint set number
ELT_STIFFNESS	E*.MATL_STIFFNESS	Element matrix data objects
LDI_C	1	Computational database
LDI_E	1	Element matrix database
LDI_S	1	System matrix database
LOAD_FACTOR	1.0	Load factor
LOAD_SET	1	Load set number
MESH	0	Mesh number
RHS	NODAL.EXT_FORCE	Applied force data object
SOLN	NODAL.DISPLACEMENT	Final solution data object
SPEC_DISP	*.SPEC_DISP	Specified displacement data object
STEP	0	Load step number

6.3.3. Argument Definitions

In this subsection, the procedure arguments summarized in Table 6.3 are defined in more detail. Note that arguments are listed in alphabetical order.

6.3.3.1 ASM_STIFFNESS Argument

Assembled Stiffness Matrix data object name. This argument specifies the first two words of the name of the output, assembled global stiffness matrix data object.

Argument syntax:

ASM_STIFFNESS = *global_stiffness_name*

where *global_stiffness_name* is a character string which must be a valid data object name. The **SS_control** procedure (see Section 3.2) allows this argument to default. (Default value: `STRUCTURE.MATL_STIFFNESS`)

6.3.3.2 CONSTRAINT_SET Argument

Constraint set number. This argument identifies the constraint set number for the merged, master model.

Argument syntax:

CONSTRAINT_SET = *constraint_set*

where the integer *constraint_set* must be a valid constraint set number. The **SS_control** procedure (see Section 3.2) calls **Assemble_Master** using the **MM_con_set** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: `1`)

6.3.3.3 ELT_STIFFNESS Argument

Element Stiffness Matrix data object name. This argument specifies the first two words of the name of the element stiffness matrices. If more than one element type is used during an analysis (with interface elements, there is always more than one element type in the analysis), it is recommended that the default value be used.

Argument syntax:

ELT_STIFFNESS = *elt_stiffness_name*

where *elt_stiffness_name* is a character string which must be a valid data object name. The **SS_control** procedure (see Section 3.2) allows this argument to default. (Default value: `E*.MATL_STIFFNESS`)

6.3.3.4 LDI_C Argument

Computational database logical device index. This argument is the logical device index, or library number, for the database containing the model data (*e.g.*, loads, nodal ordering, etc.) for the master model.

Argument syntax:

LDI_C= *ldi*

where the integer *ldi* must be set to the appropriate library number. The **SS_control** procedure (see Section 3.2) calls **Assemble_Master** using the **MM_ldi** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: `1`)

6.3.3.5 LDI_E Argument

Element database logical device index. This argument is the logical device index, or library number, for the database containing the element matrices for the master model.

Argument syntax:

LDI_E = *ldi*

where the integer *ldi* must be set to the appropriate library number. The **SS_control** procedure (see Section 3.2) calls **Assemble_Master** using the **MM_ldi** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: `1`)

6.3.3.6 LDI_S Argument

System database logical device index. This argument is the logical device index, or library number, for the database containing the system global stiffness matrix for the master model.

Argument syntax:

LDI_S = *ldi*

where the integer *ldi* must be set to the appropriate library number. The **SS_control** procedure (see Section 3.2) calls **Assemble_Master** using the **MM_ldi** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: `1`)

6.3.3.7 LOAD_FACTOR Argument

Load factor. This argument is the load factor for the master model analysis.

Argument syntax:

LOAD_FACTOR = *factor*

where *factor* is a floating point number. The **SS_control** procedure (see Section 3.2) allows this argument to default. (Default value: `1.0`)

6.3.3.8 LOAD_SET Argument

Load set number. This argument identifies the load set number for the master model.

Argument syntax:

LOAD_SET = *load_set*

where the integer *load_set* must be a valid load set number (*i.e.*, it must exist in the *ldi_c* data library). The **SS_control** procedure (see Section 3.2) calls **Assemble_Master** using the **MM_load_set** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: 1)

6.3.3.9 MESH Argument

Mesh identification number. This argument identifies the mesh to be processed for the master model.

Argument syntax:

MESH= *mesh*

where the integer *mesh* must be set to a valid mesh number. The **SS_control** procedure (see Section 3.2) calls **Assemble_Master** using the **MM_mesh** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: 0)

6.3.3.10 RHS Argument

Right-Hand Side vector data object name. This argument specifies the first two words of the name of the output, assembled global right-hand side vector data object.

Argument syntax:

RHS = *rhs_object_name*

where *rhs_object_name* is a character string and must be a valid data object name. The **SS_control** procedure (see Section 3.2) allows this argument to default. (Default value: NODAL.EXT_FORCE)

6.3.3.11 SOLN Argument

Solution vector data object name. This argument specifies the first two words of the name of the global solution vector data object.

Argument syntax:

SOLN = *soln_object_name*

where *soln_object_name* is a character string and must be a valid data object name. The **SS_control** procedure (see Section 3.2) allows this argument to default. (Default value: NODAL.DISPLACEMENT)

6.3.3.12 SPEC_DISP Argument

Specified Displacement data object name. This argument specifies the first two words of the name of the global specified displacement data object.

Argument syntax:

SPEC_DISP = *spec_disp_object_name*

where *spec_disp_object_name* is a character string and must be set to a valid data object name. The **SS_control** procedure (see Section 3.2) allows this argument to default. (Default value: `*.SPEC_DISP`)

6.3.3.13 STEP Argument

Load step identification number. This argument identifies the load step (for nonlinear analysis) to be processed for the master model.

Argument syntax:

STEP = *step*

where the integer *step* must be set to the appropriate load step number. The **SS_control** procedure (see Section 3.2) calls **Assemble_Master** using the **MM_step** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: `0`)

6.3.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the **ASM** processor. These dataset requirements are documented in detail in the COMET-AR User's Manual (Ref. 6.3-1).

6.3.5. Subordinate Processors and Procedures

Assemble_Master has only one subordinate processor, **ASM**, and no subordinate procedures. At present, **ASM** is the only assembler recognized.

6.3.6. Current Limitations

Limitations on the procedure usage are dictated by the limitations of the **ASM** processor. These limitations are documented in the COMET-AR User's Manual (Ref. 6.3-1). Limitations on the analysis in general are documented in Section 1.5.

6.3.7. Status and Error Messages

Assemble_Master will not print any status or error messages directly. All messages will be produced by the **ASM** processor. The user should refer to the COMET-AR User's Manual (Ref. 6.3-1) for specific error messages produced by this processor.

6.3.8. Examples and Usage Guidelines

The **Assemble_Master** procedure is called automatically by the **SS_control** procedure using the macrosymbols defined by the user through the **Initialize** procedure. A listing of **Assemble_Master** follows.

```
*procedure Assemble_Master (                                          --
                    ldi_c           = 1                             ; --
                    ldi_e           = 1                             ; --
                    ldi_s           = 1                             ; --
                    rhs             = NODAL.EXT_FORCE               ; --
                    soln            = NODAL.DISPLACEMENT            ; --
                    constraint_set  = 1                             ; --
                    load_set        = 1                             ; --
                    load_factor     = 1.0                           ; --
                    elt_stiffness   = E*.MATL_STIFFNESS             ; --
                    asm_stiffness   = STRUCTURE.MATL_STIFFNESS      ; --
                    spec_disp       = *.SPEC_DISP                   ; --
                    mesh            = 0                             ; --
                    step            = 0                             )
. Assemble Element Stiffness Matrix into System Matrix
    run ASM
        MODEL [ldi_c] CSM.SUMMARY...[mesh]
        INCLUDE [ldi_e],[elt_stiffness] DEFINITION = [ldi_c]
        INCLUDE [ldi_c] NODAL.DOF..[constraint_set].[mesh]
        OUTPUT [ldi_s],[asm_stiffness] FORMAT=Transpose
        SHOW/I,O
        ASSEMBLE
        STOP
. Check for Spec. Displacement and Right-hand Side Data Objects
    *find [ldi_c],[spec_disp].[load_set]..[mesh] /seq=ids_AMU
    *find [ldi_c],[rhs].[load_set]..[mesh] /seq=ids_RHS
. Assemble Right-hand Side Vector
    run ASM
        MODEL [ldi_c] CSM.SUMMARY...[mesh]
        INCLUDE [ldi_c] NODAL.DOF..[constraint_set].[mesh]
        *if < <ids_AMU> /gt 0 > /then
            INCLUDE [ldi_e],[elt_stiffness] DEFINITION = [ldi_c]
            INCLUDE [ldi_c],[spec_disp].[load_set]..[mesh] CONTENTS = DISP_N
        *endif
        *if < <ids_RHS> /gt 0 > /then
            INCLUDE [ldi_c],[rhs].[load_set]..[mesh] CONTENTS = FORC_N
        *endif
        OUTPUT [ldi_s] SYSTEM.VECTOR.[load_set]..[mesh] FORMAT = DOFVEC
        ASSEMBLE/VECTOR
        STOP
*end
```

6.3.9. References

6.3-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

6.4. Master Model Solution - Procedure Solve_Master

6.4.1. General Description

In COMET-AR, the solution of the global system of equations is normally carried out within the solution procedure (*e.g.*, **L_STATIC_1**). Due to the introduction of the interface elements, this usual solution procedure can no longer be used and the functions performed within it have been placed within different, individual procedures. Some of these additional procedures have already been discussed (*e.g.*, **Defn_FE_Freedoms**). The new procedure for performing the solution of the equation system is discussed in this Section.

Procedure **Solve_Master** obtains the solution to the global system of equations. Since constrained (either through multi-point or single-point constraints) degrees-of-freedom are not assembled, a call to **COP**, the constraint processor (Ref. 6.3-1), is required. This call to processor **COP** expands the solved system to include constraints thereby providing the user with results data objects which may be viewed and post-processed. **SS_control** (see Section 3.2) automatically calls **Solve_Master** using macrosymbols defined in procedure **Initialize** (see Section 3.3).

6.4.2. Argument Summary

SS_control invokes procedure **Solve_Master** with the COMET-AR *call directive, employing the arguments summarized in Table 6.3, which are described in detail subsequently.

Table 6.4. Procedure Solve_Master Input Arguments

Argument	Default Value	Description
CONSTRAINT_SET	1	Constraint set number
LDI_C	1	Computational database
LDI_S	1	System matrix database
LOAD_FACTOR	1.0	Load factor
LOAD_SET	1	Load set number
MESH	0	Mesh number
SOLN	NODAL.DISPLACEMENT	Final solution data object
SPEC_DISP	*.SPEC_DISP	Specified displacement data object
STEP	0	Load step number

6.4.3. Argument Definitions

In this subsection, the procedure arguments summarized in Table 6.3 are defined in more detail. Note that arguments are listed in alphabetical order.

6.4.3.1 CONSTRAINT_SET Argument

Constraint set number. This argument identifies the constraint set number for the global, merged, master model.

Argument syntax:

CONSTRAINT_SET = *constraint_set*

where the integer *constraint_set* must be a valid constraint set number. The **SS_control** procedure (see Section 3.2) calls **Solve_Master** using the **MM_con_set** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: 1)

6.4.3.2 LDI_C Argument

Computational database logical device index. This argument is the logical device index, or library number, for the database containing the model data (*e.g.*, loads, nodal ordering, etc.) for the master model.

Argument syntax:

LDI_C= *ldi*

where the integer *ldi* must be set to the appropriate library number. The **SS_control** procedure (see Section 3.2) calls **Solve_Master** using the **MM_ldi** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: 1)

6.4.3.3 LDI_S Argument

System database logical device index. This argument is the logical device index, or library number, for the database containing the system global stiffness matrix for the master model.

Argument syntax:

LDI_S = *ldi*

where the integer *ldi* must be set to the appropriate library number. The **SS_control** procedure (see Section 3.2) calls **Solve_Master** using the **MM_ldi** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: 1)

6.4.3.4 LOAD_FACTOR Argument

Load factor. This argument is the load factor (for nonlinear analysis) for the master model analysis.

Argument syntax:

LOAD_FACTOR = *factor*

where *factor* is a floating point number. The **SS_control** procedure (see Section 3.2) allows this argument to default. (Default value: `1.0`)

6.4.3.5 LOAD_SET Argument

Load set number. This argument identifies the load set number for the master model.

Argument syntax:

LOAD_SET = *load_set*

where the integer *load_set* must be a valid load set number (*i.e.*, it must exist in the *ldi_c* data library). The **SS_control** procedure (see Section 3.2) calls **Solve_Master** using the **MM_load_set** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: `1`)

6.4.3.6 MESH Argument

Mesh identification number. This argument identifies the mesh to be processed for the master model.

Argument syntax:

MESH = *mesh*

where the integer *mesh* must be set to a valid mesh number. The **SS_control** procedure (see Section 3.2) calls **Solve_Master** using the **MM_mesh** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: `0`)

6.4.3.7 SOLN Argument

Solution vector data object name. This argument specifies the first two words of the name of the global solution vector data object.

Argument syntax:

SOLN = *soln_object_name*

where *soln_object_name* is a character string and must be set to a valid data object name. The **SS_control** procedure (see Section 3.2) allows this argument to default. (Default value: `NODAL.DISPLACEMENT`)

6.4.3.8 SPEC_DISP Argument

Specified Displacement data object name. This argument specifies the first two words of the name of the global specified displacement data object.

Argument syntax:

SPEC_DISP = *spec_disp_object_name*

where *spec_disp_object_name* is a character string and must be set to a valid data object name. The **SS_control** procedure (see Section 3.2) allows this argument to default. (Default value: `*.SPEC_DISP`)

6.4.3.9 STEP Argument

Load step identification number. This argument identifies the load step (for nonlinear analysis) to be processed for the master model.

Argument syntax:

STEP = *step*

where the integer *step* must be set to the appropriate load step number. The **SS_control** procedure (see Section 3.2) calls **Solve_Master** using the **MM_step** macrosymbol defined in procedure **Initialize** (see Section 3.3). (Default value: 0)

6.4.4. Database Input/Output Summary

All database input and output requirements for this procedure are imposed by the **PVSNP** and **COP** processors. These dataset requirements are documented in detail in the COMET-AR User's Manual (Ref. 6.3-1).

6.4.5. Subordinate Processors and Procedures

Solve_Master has two subordinate processors, **PVSNP** and **COP**, and calls no procedures. The current interface element requires a solver capable of solving a non-positive-definite system of equations. The only solver so capable is, currently, **PVSNP**. Processor **COP** is executed to expand the solution system vector into a full nodal vector table so that the results may be post-processed.

6.4.6. Current Limitations

Limitations on the procedure usage are dictated by the limitations of the **PVSNP** and **COP** processors. These limitations are documented in the COMET-AR User's Manual (Ref. 6.3-1). Limitations on the analysis in general are documented in Section 1.5.

6.4.7. Status and Error Messages

Solve_Master does not print any status or error messages directly. All messages are produced by the **PVSNP** or **COP** processors. The user should refer to the COMET-AR User's Manual (Ref. 6.3-1) for specific error messages produced by these processors.

6.4.8. Examples and Usage Guidelines

The **Solve_Master** procedure is called automatically by the **SS_control** procedure using the appropriate macrosymbols as defined by the user. A listing of the procedure follows.

```
*procedure Solve_Master (                                          --
                ldi_c           = 1                             ; --
                ldi_s           = 1                             ; --
                spec_disp       = *.SPEC_DISP                   ; --
                soln            = NODAL.DISPLACEMENT            ; --
                constraint_set  = 1                             ; --
                load_set        = 1                             ; --
                load_factor     = 1                             ; --
                mesh            = 0                             ; --
                step            = 0                             )
. Check for Spec. Displacement and Right-hand Side Data Objects
   *find [ldi_s],[spec_disp].[load_set]..[mesh] /seq=ids_AMU
   *find [ldi_s], SYSTEM.VECTOR.[load_set].0.[mesh] /seq=ids_RHS
   *if <<ids_RHS> /le 0> /then
       *remark *** No right-hand side vector in library [ldi_s]
       *stop
   *endif
. Solve the system of equations with processor pvsnp
   run PVSNP
       SET LDiC  = [ldi_c]
       SET LdiS  = [ldi_s]
       SET MESH  = [mesh]
       SET STEP  = [step]
       SET IJUMP = 9
       FACTOR
       STOP
. Reinstate Deleted And Specified Freedoms
   run COP
       MODEL [ldi_c] CSM.SUMMARY...[mesh]
       *if < <ids_AMU> /gt 0 > /then
         *def/a am_ph='VALUES=[ldi_s],[spec_disp] [load_factor]'
       *else
         *def/a am_ph=' '
       *endif
       EXPAND/DOFVEC
       INPUT =[ldi_s],SYSTEM.VECTOR.[load_set]..[mesh] --
       OUTPUT=[ldi_s],[soln].[load_set].[constraint_set] <am_ph> --
       DOFDAT=[ldi_c] [constraint_set] [mesh] <am_phrase>
       STOP
*end
```

6.4.9. References

6.4-1 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

THIS PAGE INTENTIONALLY BLANK

Part IV.
PROCESSORS

THIS PAGE INTENTIONALLY BLANK

7. Interface Element Processors

7.1. Overview

In this Chapter, the Generic Interface Element Processor as well as a specific interface element processor are described. The Generic Interface Element Processor is much like the Generic Element Processor, or **GEP** (Ref. 7.2-1), in that it is a standard processor template (also called a "shell") from which many individual interface element processors may be developed. All of the individual processors share a common user interface and a common database interface through the Generic Interface Element Processor. This common shell is named the **EI** processor; individual element processors are named **EI*** processors (*e.g.*, **EI1**, **EI2**). Each **EI** processor performs all the functions associated with elements of a particular type (*e.g.*, defines elements, forms stiffness).

The Chapter is organized as listed in Table 7.1

Table 7.1. Outline of Chapter 7: Interface Element Processors

Section	Processor	Function
2	EI	Generic Interface Element Processor
3	EI1	Hybrid Variational Interface Element Processor

THIS PAGE INTENTIONALLY BLANK

7.2. Processor EI (Generic Interface Element Processor)

7.2.1. General Description

The generic interface element processor, or **EI** (for Element, Interface) provides a standard template which may be used to implement different interface elements, all of which may then coexist within COMET-AR as independent software modules. Processor **EI** provides a common user interface and a common database interface for each of these potentially independent modules. This processor was modeled after the Generic Element Processor for structural elements (Ref. 7.2-1), **ES**, which forms the foundation for all finite element implementation within COMET-AR. Indeed, the **EI** processor looks very much like the **ES** processor, using many of the same commands and similar underlying software. The **EI** processor is however, significantly different from the **ES** processor; many new data objects are required to define the interface element and the user input required for the interface element definition is radically different than that required for finite element definition.

This Section describes the interface element reference frames, the standard user interface, and the database interface employed by the **EI** processor and therefore, by all processors which use the **EI** processor template.

7.2.2. Interface Element Reference Frames

The **EI** processor shell creates the transformation matrices required to define two reference frames - the edge frame (attached to the substructure edges) and the interface frame (attached to pseudo-nodes) - as shown in Figure 7.1. The edge frame is the computational frame for the alpha-nodes and the interface frame is the computational frame for the pseudo-nodes. These frames need not be coincident with any of the finite element node computational frames therefore the interface element matrices must be transformed prior to assembly in the global system of equations.

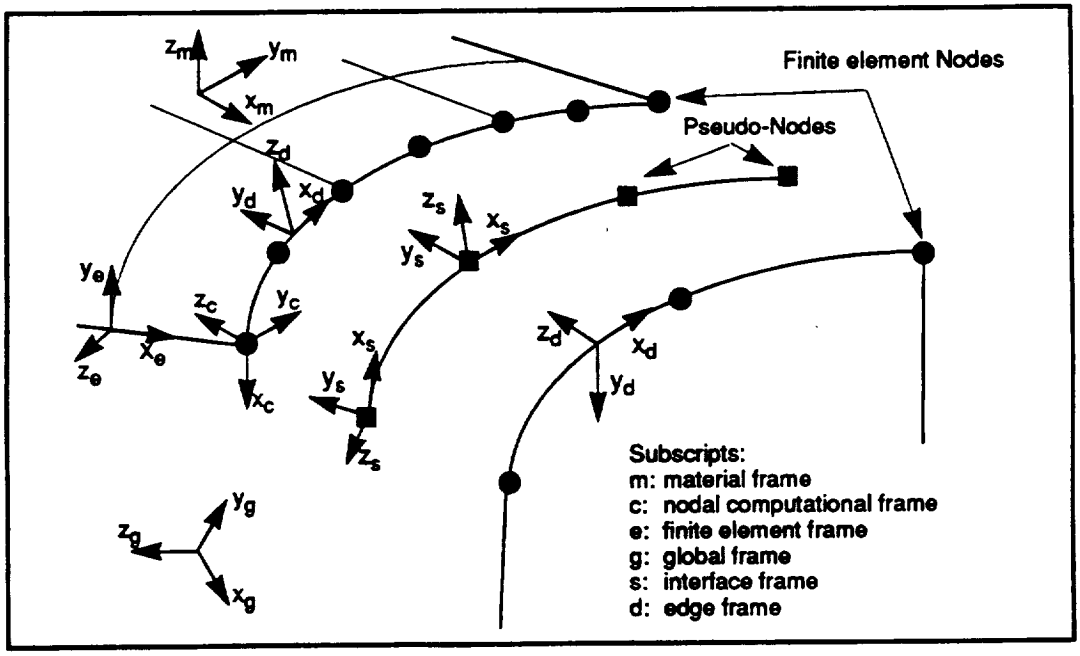


Figure 7.1. Interface Element Reference Frames

Creation of the transformation matrices for these reference frames is a two step process. First, the edge and interface frames are defined by creating tangent and normal vectors along the edge and interface respectively; these vectors are then saved in the database. This first step is completed during the interface element definition. Second, the vectors are read from the database and transformation matrices, which transform edge and interface to a global frame, are formed based on the vectors. This second step is accomplished during the formation of the interface element stiffness matrix.

The procedure employed by the shell for defining the edge and interface frames (denoted by the subscripts "d" and "s" respectively) is as follows:

1. Calculate the average nodal normals, $\mathbf{n}_d$, for the finite element nodes along the interface for all substructures. Note that in general, each finite element node may be connected to more than one finite element along the interface. A normal is defined at each node for each finite element in the global frame. The average is calculated based on these elemental normals. Only the normals from elements along an interface are considered at a given node.
2. Using a piecewise linear interpolation between the average nodal normals along the finest (discretized with the most nodes) substructure, calculate a normal, $\mathbf{n}_s$, for each pseudo-node.
3. Calculate tangents, $\mathbf{t}'_d$ (for finite element nodes) and $\mathbf{t}'_s$ (for pseudo-nodes), by differentiating along the interface element path. These are interim values which are discarded once the frames are created.
4. Calculate the transverse tangents $\mathbf{b}'_d = \mathbf{n}_d \times \mathbf{t}'_d$ and $\mathbf{b}'_s = \mathbf{n}_s \times \mathbf{t}'_s$. Normalize $\mathbf{b}'$ in both frames to recover $\mathbf{b}_d$ and $\mathbf{b}_s$.
5. Calculate $\mathbf{t}_d = \mathbf{b}_d \times \mathbf{n}_d$ and $\mathbf{t}_s = \mathbf{b}_s \times \mathbf{n}_s$.
6. Repeat steps 3-5 for each substructure, performing only edge frame calculations.
7. Locate the image of each pseudo-node along the edge of each substructure and linearly interpolate a normal and two tangents for each image in each edge frame (*i.e.*, form $\mathbf{t}_d^p$, $\mathbf{b}_d^p$, $\mathbf{n}_d^p$).

Once the various normals and tangents are created and saved, transformation matrices are formed as needed. The shell creates three sets transformation matrices: $\mathbf{T}_{sg}$, $\mathbf{T}_{dg}$, and $\mathbf{T}_{dg}^p$ which are defined by

$$\mathbf{T}_{sg} = \begin{bmatrix} \mathbf{t}_s^T \\ \mathbf{b}_s^T \\ \mathbf{n}_s^T \end{bmatrix}; \qquad \mathbf{T}_{dg} = \begin{bmatrix} \mathbf{t}_d^T \\ \mathbf{b}_d^T \\ \mathbf{n}_d^T \end{bmatrix}; \qquad \mathbf{T}_{dg}^p = \begin{bmatrix} \mathbf{t}_d^{pT} \\ \mathbf{b}_d^{pT} \\ \mathbf{n}_d^{pT} \end{bmatrix}. \tag{7.2-1}$$

These matrices are then passed down through the EI processor to the kernel along with the computational-to-global transformations, found in the NTT data object **NODAL.TRANSFORMATION...mesh**, for the finite element nodes of each substructure. The interface element developer is then responsible for applying the transformations as necessary to the matrix generated by the kernel so that the pseudo-node and alpha-node degrees-of-freedom (if any) are in the interface and edge frames respectively.

7.2.3. Automatic Drilling Degree-of-Freedom-Suppression

In specific applications, it may be necessary to constrain the so-called drilling degree-of-freedom. The suppression of this degree-of-freedom along the interface is only required when the drilling freedom is suppressed in all connected substructures and when the geometry of the structure is such that there is a degree-of-freedom in the connected structure for which there is no stiffness. For example, if a curved panel with a

central hole is modeled with an ES1/EX97 element (Ref. 7.2-3) as two substructures so that there is a local model in the neighborhood of a central hole and a global model away from the hole, the drilling degree-of-freedom for both pseudo-nodes and alpha-nodes along each interface element would need to be suppressed. If, on the other hand, one were using an interface element to attach a blade stiffener to a flat panel, no drilling freedom would need to be suppressed for the pseudo-nodes since along this intersection, all six degrees-of-freedom have some stiffness associated with them , with contributions coming either from the skin or from the stiffener. The alpha-nodes however, represent tractions along a substructure thus their drilling freedoms do need to be suppressed.

In this second example, the user would be required to manually constrain the alpha-node drilling freedoms. The first example however, is handled internally by the EI processor shell. That is, when two substructures connect at an interface element and neither substructure has drilling stiffness and both use the same edge and computational frames, the EI processor automatically suppresses the drilling freedom (always degree-of-freedom 6) for both pseudo- and alpha-nodes. This condition is detected by looking at an average normal for each pseudo-node (computed by taking the average of the normals of each incoming substructure at each pseudo-node) and comparing the substructure normals to this average. If the difference between the average and all incoming substructure normals is less than 1 degree for all pseudo-nodes along the interface element, then the drilling freedom is suppressed by the EI processor at all pseudo- and alpha-nodes. If this test is not passed, then the drilling freedom is not suppressed; if the user wishes to manually suppress the drilling freedom, the CONSTRAINT subcommands of the DEFINE ELEMENTS command will permit that suppression.

7.2.4. Command Classes

The generic interface element processor commands are partitioned into three classes. A summary of these classes is given in Table 7.4.

Table 7.2. Generic Interface Element (EI) Command Classes

Command Class	Function
RESET	Process element reset parameters. RESET is issued in conjunction with DEFINE and FORM.
DEFINE	Process element definition commands. Used during pre-processing, this class includes such information as the definition of element connectivity and the definition of active degrees of freedom.
FORM	Formation of element matrices.

Each of these classes is discussed in detail in subsequent Sections.

7.2.5. The EI Processor RESET Commands

The RESET commands are used to define certain parameters which are meaningful to each of the other commands (*i.e.*, to the DEFINE and FORM commands). Once a RESET command has been issued, it remains valid for each interface element defined or formed within the current execution. The command may be issued as many times as necessary within a given execution. There are several of these RESET commands; they are summarized in Table 7.3 and discussed in detail in subsequent sections.

Table 7.3. Summary of RESET Commands

Keyword	Default	Meaning
RESET LDI	1	Logical device index for output.
RESET MESH	0	Mesh number for output.
RESET STEP	0	Load step number for output.
RESET LOAD_SET	1	Load set number for output.
RESET CONSTRAINT_CASE	1	Constraint case number for output.
RESET ZERO	1.E-5	Zero value.

7.2.5.1 The RESET LDI Command

Logical Device Index for the interface elements.

Command Syntax:

RESET LDI = *ldi*

where the integer *ldi* signifies the output logical device index. When used with the DEFINE ELEMENTS command, *ldi* must be attached to a new, empty but open data library. When used with the FORM command class, this *ldi* must be open and contain the interface element definitions. (Default: 1)

7.2.5.2 The RESET MESH Command

Mesh Number. When used with the DEFINE ELEMENTS command, this command assigns the mesh number to all of the interface elements defined. When used with the FORM command class, the command is used to identify the interface elements to be processed.

Command Syntax:

RESET MESH = *mesh*

where the integer *mesh* identifies the mesh number. (Default: 0)

7.2.5.3 The RESET STEP Command

Load Step Number. When used with the DEFINE ELEMENTS command, this command assigns the step number to all of the interface elements defined. When used with the FORM command class, the command is used to identify the interface elements to be processed.

Command Syntax:

RESET STEP = *step*

where the integer *step* identifies the load step number. (Default: 0)

7.2.5.4 The RESET LOAD_SET Command

Load Set Number. When used with the DEFINE ELEMENTS command, this command assigns the load set number to all of the interface elements defined. When used with the FORM command class, the command is used to identify the interface elements to be processed.

Command Syntax:

RESET LOAD_SET = *load_set*

where the integer *load_set* identifies the load set number. (Default: 1)

7.2.5.5 The RESET CONSTRAINT_CASE Command

Constraint Case Number. When used with the DEFINE ELEMENTS command, this command assigns the constraint case number to all of the interface elements defined. When used with the FORM command class, the command is used to identify the interface elements to be processed.

Command Syntax:

RESET CONSTRAINT_CASE = *constraint_case*

where the integer *constraint_case* identifies the constraint case number. (Default: 1)

7.2.5.6 The RESET ZERO Command

Zero value. Zero value is the tolerance used in determining whether or not a node lies along the current interface element.

Command Syntax:

RESET ZERO = *zero*

where *zero* is a floating point number. (Default: 1.E-5)

7.2.6. The EI Processor DEFINE Commands

A summary of the DEFINE commands accessible via the generic interface element processor is given in Table 7.4. Complete descriptions of these commands are provided in subsequent Sections.

Table 7.4. Generic Interface Element (EI) Command Classes

Command Class	Function
DEFINE ELEMENTS	Define element connectivity; includes nodal connectivity for each substructure attached to a given interface element along with interpolation parameters, scale factors, and other element parameters.
DEFINE FREEDOMS	Define valid interface element nodal degrees of freedom for automatic freedom suppression.

7.2.6.1 The DEFINE ELEMENTS Command

The DEFINE ELEMENTS command has several subcommands. The command syntax is as follows:

```
DEFINE ELEMENTS
    ELEMENT i /CURVED /DSPLINE=(1,2,3} /P_NODES=np /SCALE=scalef
        CONSTRAINTS
            ZERO {d1,d2,d3,theta1,theta2,theta3}
            NONZERO
              d1 = value1
                 .
                 .
              theta3 = value6
            MPC ddof nind α
              idof β1
                 .
                 .
              idof βnind
        END_CONSTRAINTS
        SS k  /LDI=sldi {/FE /BE /RR /MESH=mesh /CONS=icon}
            NODES = n1, n2,... /GSPLINE=(1,2,3}
            CONSTRAINTS
                 .
                 .
            END_CONSTRAINTS
        SS m  /LDI=sldi {/FE /BE /RR /MESH=mesh /CONS=icon}
            COORDINATES = x1,x2, ... xp /GSPLINE=(1,2,3}
            CONSTRAINTS
                 .
                 .
            END_CONSTRAINTS
        SS p  /LDI=sldi {/FE /BE /RR /MESH=mesh /CONS=icon}
            END_NODES = i1,i2,... /GSPLINE=(1,2,3} --
              /FILTERx=x1,x2 /FILTERy=y1,y2 /FILTERz=z1,z2 /FILTERn=n1,n2
            CONSTRAINTS
                 .
                 .
            END_CONSTRAINTS
END_DEFINE
```

Each of the subcommands must be given in the order they are listed in the example. Each interface element is defined by specifying the finite element nodes along each substructure to which it is attached. In this syntax example, three options for defining these nodes are listed: NODES, COORDINATES, and END_NODES. The three options are mutually exclusive. That is, along a particular substructure for a given interface element, either NODES, COORDINATES, or END_NODES may be specified. Nodes along additional substructures need not necessarily be defined using the same option, but if options are mixed within an interface element extreme care should be taken. Each of these options is described in detail in subsequent sections.

Constraints may be applied at both the substructure and the interface element level. Constraints applied to the interface element (the first CONSTRAINT subcommand in the example) are applied to the pseudo-nodes. Constraints applied at the substructure level (the remaining CONSTRAINT subcommands in the example) are applied to the alpha-nodes. The general syntax for the constraints mimics the syntax of the COP constraint processor of COMET-AR (Ref. 7.2-2).

While there are default values for all of the *qualifiers* used in the example syntax (and they are therefore optional), the *subcommands* are required input to the DEFINE ELEMENTS command. Valid subcommands and their associated optional qualifiers are described in subsequent sections.

7.2.6.1.1 *The ELEMENT Subcommand*

The ELEMENT subcommand signifies that a new element definition is beginning. Elements should be numbered sequentially (to minimize database storage requirements) although there is no absolute requirement that they be so numbered.

Subcommand syntax:

ELEMENT *i* {/CURVED /DSPLINE={1,2,3} /P_NODES=*np* /SCALE=*scale*

where *i* is an integer identifying the interface element number and the optional qualifiers are described in the following subsections.

7.2.6.1.1.1 *The CURVED Qualifier*

This qualifier is required if the interface is to be represented geometrically as a curve. If the qualifier is not present, the geometry of the interface will be assumed to be piecewise linear. If the qualifier is present, the geometry of the interface will be assumed to be a curve and will be interpolated using the function defined by the GSPLINE qualifier (see Sections 7.2.6.1.4 through 7.2.6.1.6 for a description of this qualifier) for each attached substructure.

7.2.6.1.1.2 *The DSPLINE Qualifier*

This qualifier optionally sets the level of interpolation for the displacements along the interface. Permissible values are 1,2, or 3 denoting piecewise linear, and quadratic, or cubic spline functions (respectively) for the displacement interpolating functions. (Default: `/DSPLINE=1`)

7.2.6.1.1.3 *The P_NODES Qualifier*

This optional qualifier specifies the number of evenly-spaced pseudo-nodes which are to be placed along the interface element. If the number specified by the user is outside the permissible range for this element configuration, the number of pseudo-nodes will be automatically reset to an appropriate value. If the user

specifies no value, then the interface element will define automatically an appropriate number of pseudo-nodes.

7.2.6.1.1.4 *The SCALE Qualifier*

The SCALE qualifier sets a scale factor used to ensure that the assembled global stiffness matrix will not be too ill-conditioned. The value of *scale* should be set to within two orders of magnitude of $E_1 \times$ (element volume), where the element volume is from the largest finite element along the interface and E_1 is the corresponding longitudinal Young's modulus. (Default value: `/SCALE=1.E6`)

7.2.6.1.2 *The CONSTRAINT Subcommand*

The CONSTRAINT subcommand, when issued immediately after an ELEMENT subcommand, is used to define constraints on the pseudo-nodes. When issued after a SUBSTRUCTURE subcommand or between SUBSTRUCTURE subcommands, it is used to define constraints on the alpha-nodes. In both cases the syntax used for constraint definition is the same.

Subcommand syntax:

CONSTRAINTS
ZERO {d1, d2, d3, theta1, theta2, theta3}
NONZERO
d1 = d_1
d2 = d_2
d3 = d_3
theta1 = θ_1
theta2 = θ_2
theta3 = θ_3
MPC *ddof nind* α
idof$_1$ β_1
idof$_2$ β_2
⋮
idof$_{nind}$ β_{nind}
END_CONSTRAINTS

where the d_i are prescribed displacements; the θ_i are prescribed rotations; *ddof* is the dependent degree-of-freedom for the multipoint constraint (and may be: d1, d2, d3, theta1, theta2, or theta3); *nind* is the number of independent degrees-of-freedom that define the multipoint constraint for *ddof*; α is a floating point constant added to the multipoint constraint equation; the *idof*$_i$ are the independent freedoms upon which *ddof* depends (and may be: d1, d2, d3, theta1, theta2, or theta3); and the β_i are the coefficients of the *idof*$_i$ in the multipoint constraint equation.

The constraints defined using this subcommand are applied to all pseudo-nodes (if issued immediately following the ELEMENT subcommand) or all alpha-nodes (if issued after a SUBSTRUCTURE, or SS, subcommand) on the interface. Therefore no node numbers are specified. The syntax is very similar to, but not identical to, the syntax used in the COP processor of COMET-AR (Ref. 7.2-2).

The phrase "multipoint constraint" (or "MPC") is, in this context, not completely accurate as it is used to define relationships among the degrees-of-freedom associated with each pseudo-node or alpha-node rather than to define relationships among degrees-of-freedom associated with several nodes (or points). Put another way, the MPC defined in this subcommand will relate two or more degrees-of-freedom at a pseudo-node or alpha-node to one dependent degree-of-freedom at the same pseudo-node or alpha-node and it will establish the same relationship at each pseudo-node or alpha-node. Thus, the MPC connects one degree-of-freedom at a point to other degrees-of-freedom at the same point for each point along the interface. The specification of MPC's on the interface will be needed when constraints are defined on the substructures along the interface and the finite element nodal computational frames do not coincide with the edge or interface frames (the computational frames for the alpha-nodes and the pseudo-nodes respectively).

7.2.6.1.3 *The SS Subcommand*

The SS subcommand identifies the substructures connected to the current interface element. The command may also be issued as SUBS *k* or SUBSTRUCTURE *k*. The number *k* assigned here will be used throughout the analysis to identify the substructure.

Subcommand Syntax:

SS *k* /LDI=*sldi* {/FE /BE /RR /MESH=*mesh* /CONS=*icon*}
or
SUBS *k* /LDI=*sldi* {/FE /BE /RR /MESH=*mesh* /CONS=*icon*}
or
SUBSTRUCTURE *k* /LDI=*sldi* {/FE /BE /RR /MESH=*mesh* /CONS=*icon*}

7.2.6.1.3.1 *The LDI Qualifier*

The LDI qualifier identifies the logical device index of the library containing the model definition data for this substructure. The LDI specified here must exist and be open. More than one substructure may exist in a single library. (Default value:`/LDI=1`)

7.2.6.1.3.2 *The FE, BE, RR Qualifiers*

This set of qualifiers identifies the form of the idealization of the substructure. `/FE` identifies a finite element substructure, `/BE` identifies a boundary element substructure, and `/RR` identifies a Rayleigh-Ritz substructure. Currently, only the `/FE` qualifier can be used; COMET-AR has no current capability for boundary element or Rayleigh-Ritz substructures. This set of qualifiers is a mutually exclusive set (*i.e.*, a substructure can have no more than one of these qualifiers). (Default value: `/FE`)

7.2.6.1.3.3 *The MESH Qualifier*

The optional MESH qualifier identifies the mesh number of the substructure model data to be used in defining this interface element. (Default value: `/MESH=0`)

7.2.6.1.3.4 *The CONS Qualifier*

The optional CONS qualifier identifies the constraint set number of the substructure model data to be used in defining this interface element. (Default value: `/CONS=1`)

7.2.6.1.4 *The NODES Subcommand*

The NODES, COORDINATES, and END_NODES subcommands are mutually exclusive. That is, if NODES are specified for a given substructure, then COORDINATES and END_NODES may not be (and vice versa). If NODES are given, then the interface geometry will pass through the listed m nodes. If the command is issued multiple times (*i.e.*, once for each substructure), all the nodes listed will be used to define the geometry of the interface.

Subcommand Syntax:

NODES = n_1, n_2, ... n_m /GSPLINE={1,2,3}

where n_1, n_2, and n_m are integer finite element node numbers. (Default value: None)

7.2.6.1.4.1 *The GSPLINE Qualifier*

The GSPLINE qualifier sets the order of interpolation to be used for the representation of the geometry of the substructure along the interface. When GSPLINE is set to 1,2, or 3, then piecewise linear, or quadratic or cubic spline functions (respectively) will be used to represent the geometry of the interface. This qualifier is required only once per interface element; if it is specified more than once, then only the last value will be retained. If the /CURVED qualifier has not been set on the ELEMENT command line, then GSPLINE will be set to 1 regardless of the value specified using the GSPLINE qualifier. (Default: `/GSPLINE=1`)

7.2.6.1.5 *The COORDINATES Subcommand*

The COORDINATES subcommand defines the coordinates of p points, not necessarily nodes, to be used to define the geometry of the interface element. The COORDINATES, NODES, and END_NODES subcommands are mutually exclusive. That is, if COORDINATES are specified for a given substructure, then NODES and END_NODES may not be specified (and vice versa). If COORDINATES are given, then the interface geometry will pass through the listed p points. If the command is issued multiple times (*i.e.*, for each substructure), all the points listed will be used to define the geometry of the interface.

Subcommand Syntax:

COORDINATES = $\mathbf{x}_1$,$\mathbf{x}_2$, ... $\mathbf{x}_p$ /GSPLINE={1,2,3} --
/FILTERx=x_1,x_2 /FILTERy=y_1,y_2 /FILTERz=z_1,z_2 --
/FILTERn=n_1,n_2

where $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_p$ are the coordinates of points (which need not be nodes); x_i, y_i, z_i are coordinates; and n_i are node numbers. Currently, **p must be 2** and it is assumed that the two points specified by the user are the end points of a line.

7.2.6.1.5.1 *The GSPLINE Qualifier*

The GSPLINE qualifier sets the order of interpolation to be used for the representation of the geometry of the substructure along the interface. When GSPLINE is set to 1,2, or 3, then piecewise linear, or quadratic or cubic spline functions (respectively) will be used to represent the geometry of the interface. This qualifier is required only once per interface element; if it is specified more than once, then only the last value will be retained. If the /CURVED qualifier has not been set on the ELEMENT command line, then GSPLINE will be

set to 1. **The current implementation restricts the use of this qualifier in conjunction with the COORDINATES subcommand. It may only be used with a linear interface and therefore must be set to 1.** (Default: `/GSPLINE=1`)

7.2.6.1.5.2 *The FILTER* Qualifiers*

If the COORDINATES subcommand has been used to define the interface geometry, it may be useful to set filters on the coordinates and node numbers of nodes to be processed, especially if the model is very large. With no filters, the EI processor will search the entire domain for nodes along the interface. Four filters (/FILTERx, /FILTERy, /FILTERz, and /FILTERn) have been provided. The input to each is a pair of numbers representing the lower and upper bounds on the region to be searched (*e.g.*, `/FILTERX=1.0,10.0 /FILTERN=200,300`). The coordinate filters limit the geometric search region; the node number filter limits the topographic search region. Any combination of the four filters (or all of them) may be specified. (Default: None)

7.2.6.1.6 *The END_NODES Subcommand*

The END_NODES subcommand defines the node numbers at the end points of a line which defines the geometry of the interface element. The END_NODES, COORDINATES, and NODES subcommands are mutually exclusive. That is, if END_NODES are specified for a given substructure, then COORDINATES and NODES may not be (and vice versa). If END_NODES are given, then a straight line, passing through the listed nodes, will define the interface geometry.

Subcommand Syntax:

END_NODES = n_1,n_2 --
/FILTERx=x_1,x_2/FILTERy=y_1,y_2/FILTERz=z_1,z_2 --
/FILTERn=n_1,n_2

where n_1,n_2 are the integer node numbers of the interface element end nodes; x_i, y_i, z_i are coordinates; and n_i are node numbers.

7.2.6.1.6.1 *The FILTER* Qualifier*

If the END_NODES subcommand has been used to define the interface geometry, it may be useful to set filters on the coordinates and node numbers of other nodes to be processed, especially if the model is very large. With no filters, the EI processor will search the entire domain for nodes along the linear interface. Four filters (/FILTERx, /FILTERy, /FILTERz, and /FILTERn) have been provided. The input to each is a pair of numbers representing the lower and upper bounds on the region to be searched (*e.g.*, `/FILTERZ=10.325,105.920 /FILTERN=475,800`). The coordinate filters limit the geometric search region; the node number filter limits the topographic search region. Any combination of the four filters (or all of them) may be specified. (Default: None)

7.2.6.1.7 *The END_DEFINE Subcommand*

This subcommand signals the end of the interface element definitions. It should only be issued after all interface elements have been defined.

7.2.6.1.8 *Input Datasets Required by the DEFINE ELEMENTS Command*

Input datasets are those which define the individual substructures which are connected to the interface elements. The datasets listed in Table 7.5 must exist for each substructure used to define an interface element. A description of the contents of each data object may be found in Ref. 7.2-3.

Table 7.5. Input Datasets Required by the Define Elements Command

Dataset	Description	Type
CSM.SUMMARY...*mesh*	Model summary for input Substructure	CSM
NODAL.COORDINATE...*mesh*	Substructure nodal coordinates	NCT
NODAL.DOF..*concase.mesh*	Substructure constraints	NDT
NODAL.SPEC_DISP.*ldset.concase.mesh*	Substructure specified displacements	NVT
NODAL.TRANSFORMATION...*mesh*	Nodal global-to-local transformations	NTT
EltName.DEFINITION...*mesh*	Element definition for input Substructures	EDT
EltName.NORMALS...*mesh*	Element nodal normals for Substructures	EAT

Note that if displacements are specified for a given substructure, they must be specified prior to calling the EI processor to DEFINE ELEMENTS. If there are no specified displacements on any substructures then the NODAL.SPEC_DISP.*ldset.concase.mesh* dataset is not required.

7.2.6.1.9 *Output Datasets Created/Updated by the DEFINE ELEMENTS Command*

Datasets output to the interface element library are those which define the individual interface elements. Along with the usual datasets (*i.e.*, those created by ES element processors) several additional objects are used to define the interface elements. The datasets listed in Table 7.6 will exist in the interface element library. The datasets marked with the dagger (†) are new objects for which full descriptions appear in Chapter 10 of this report. A description of the contents of each of the other data objects (those not marked with a dagger) may be found in Ref. 7.2-3.

Table 7.6. Output Datasets Created/Updated by Define Elements Command

Dataset	Description	Type
CSM.SUMMARY...*mesh*	Model summary for interface element library	CSM
NODAL.COORDINATE...*mesh*	Nodal coordinates	NCT
NODAL.DOF..*concase.mesh*	Constraints	NDT
NODAL.TRANSFORMATION...*mesh*	Nodal global-to-local transformations	NTT
NODAL.TYPE...*mesh*†	Node types	NAT
EltName.DEFINITION...*mesh*	Element definition	EDT
EltName.ELTYPE...*mesh*†	List of finite element types along each interface element	EAT
EltName.NODSS...*mesh*†	List of substructures connected to each interface element	EAT
EltName.NORMALS...*mesh*†	Normal vectors for finite element nodes and pseudo-nodes	EAT
EltName.PARAMS...*mesh*†	Interface element parameters	EAT
EltName.SCALE...*mesh*†	Scale factor for each interface element	EAT
EltName.SCOORD...*mesh*†	Path coordinates for nodes on interface element	EAT

Table 7.6. Output Datasets Created/Updated by Define Elements Command (Continued)

EltName.SSID...*mesh*†	List of substructures connected to each interface element	EAT
EltName.TANGENT_S...*mesh*†	Element path tangent vectors for nodes and pseudo-nodes	EAT
EltName.TANGENT_T...*mesh*†	Element surface tangents for nodes and pseudo-nodes	EAT
EltName.TGC...*mesh*†	Computational-to-global transformation matrices for the finite element nodes in each interface element.	EAT

† New Object; see Chapter 10 for description.

7.2.6.2 The DEFINE FREEDOMS Command

The DEFINE FREEDOMS command triggers the suppression of inactive degrees-of-freedom. The EI processor will use the information supplied through the DEFINE ELEMENTS command to decide whether or not there are globally inactive degrees-of-freedom (*e.g.*, drilling freedoms). The command has no subcommands or qualifiers. Execution of the EI processor is all that is required. The command syntax is simply:

```
DEFINE FREEDOMS
```

The determination of the active degrees-of-freedom for the pseudo-nodes and the alpha-nodes is made by the interface element processor during the definition of the elements. In the present implementation, the computational frames for both the pseudo-nodes and the alpha-nodes are defined so that the drilling degree-of-freedom is always the sixth degree-of-freedom. During the element definition, two parameters are set automatically, **Drill_Dof** and **Drill_Sup**, and saved in the **EAT** data object named **EltName.PARAMS...mesh** (see Section 10.3 for a description of this data object). The parameter **Drill_Dof** is set to six. The parameter **Drill_Sup**, is a flag which indicates whether or not the **Drill_Dof** degree of freedom is to be suppressed.

The decision to suppress the drilling degree-of-freedom is made based on two criteria. First, the suppression need occur only if the interface element connects two substructures, as more than two substructures cannot be coplanar. Second, if the difference between either substructure normal and the average normal is greater than one degree at any pseudo-node, the drilling degree of freedom is not flagged for suppression (*i.e.*, **Drill_Sup** is set to `<false>`). If the difference between both substructure normals and the average normal are within one degree for all pseudo-nodes, the drilling degree-of-freedom is flagged for suppression (*i.e.*, **Drill_Sup** is set to `<true>`).

When the DEFINE FREEDOMS command is issued, the processor reads in the values of **Drill_Dof** and **Drill_Sup** set for each interface element when the DEFINE ELEMENTS command was issued. If **Drill_Sup** has been set to `<true>`, then the degree-of-freedom specified by **Drill_Dof** is suppressed for each pseudo- and alpha-node in the interface element. If **Drill_Sup** has been set to `<false>`, then no degrees-of-freedom are suppressed for the interface element pseudo- and alpha-nodes. Once the inactive freedoms have been suppressed, the remaining active degrees-of-freedom are assigned equation numbers.

7.2.6.2.1 *Input Datasets Required by the DEFINE FREEDOMS Command*

Input datasets for the DEFINE FREEDOMS command are those which define the interface elements. The datasets listed in Table 7.7 are used by the EI processor during processing of this command. A description of the EAT object may be found in Chapter 10; all other objects are described in Ref. 7.2-3.

Table 7.7. Input Datasets Required by the DEFINE FREEDOMS Command

Dataset	Description	Type
CSM.SUMMARY...*mesh*	Model summary for interface elements	CSM
NODAL.DOF..*concase.mesh*	Pseudo-node and alpha-node constraints	NDT
NODAL.TRANSFORMATION...*mesh*	Nodal global-to-local transformations	NTT
EltName.DEFINITION...*mesh*	Element definition for interface elements	EDT
EltName.PARAMS...*mesh*	Interface element parameters	EAT

7.2.6.2.2 *Output Datasets Created/Updated by the Define Freedoms Command*

Output datasets listed in Table 7.8 are those which define the active nodal degrees of freedom and the nodal reference frames. A description of these objects may be found in Ref. 7.2-3.

Table 7.8. Output Datasets Created/Updated by the DEFINE FREEDOMS Command

Dataset	Description	Type
CSM.SUMMARY...*mesh*	Model summary for interface elements	CSM
NODAL.DOF..*concase.mesh*	Pseudo-node and alpha-node constraints	NDT
NODAL.TRANSFORMATION...*mesh*	Nodal global-to-local transformations	NTT

7.2.7. The EI Processor FORM Command

There is presently only one FORM command implemented within the EI processor, the FORM STIFFNESS/MATL command. This command triggers the formation of the element stiffness matrices for all of the interface elements identified by the processor RESET commands. The command syntax is:

```
FORM STIFFNESS/MATL
```

7.2.7.1 Input/Output Datasets

Input datasets are largely those created during the interface element definition. The command output is, for each interface element, the element stiffness matrix which may be assembled along with other finite element matrices.

7.2.7.1.1 *Input Datasets Required by the FORM STIFFNESS Command*

Input datasets are those which define the interface elements. The datasets listed in Table 7.7 are used by the EI processor during the processing of this command. A description of the EAT and NAT objects may be found in Chapter 10; all other objects are described in Ref. 7.2-3.

Table 7.9. Input Datasets Required by FORM STIFFNESS Command

Dataset	Description	Type
CSM.SUMMARY...*mesh*	Model summary for input interface elements	CSM
NODAL.DOF..*concase.mesh*	Pseudo-node and alpha-node constraints	NDT
NODAL.TRANSFORMATION...*mesh*	Nodal global-to-local transformations	NTT
NODAL.TYPE...*mesh*	Node types	NAT
EltName.DEFINITION...*mesh*	Element definition for interface elements	EDT
EltName.ELTYPE...*mesh*	Finite element types along each interface element	EAT
EltName.NodSS...*mesh*	SS connected to each node of each interface element	EAT
EltName.NORMALS...*mesh*	Normal vectors for finite element nodes and pseudo-nodes	EAT
EltName.PARAMS...*mesh*	Interface element parameters	EAT
EltName.SCALE...*mesh*	Scale factor for each interface element	EAT
EltName.SCOORD...*mesh*	Path coordinates for nodes on each interface element	EAT
EltName.SSID...*mesh*	List of substructures connected to each interface element	EAT
EltName.TANGENT_S...*mesh*	Element path tangent vectors for nodes and pseudo-nodes	EAT
EltName.TANGENT_T...*mesh*	Element surface tangents for nodes and pseudo-nodes	EAT
EltName.TGC...*mesh*	Computational-to-global transformation matrices for the finite element nodes in each interface element.	EAT

7.2.7.1.2 *Output Datasets Created/Updated by the Form Stiffness Command*

Only one dataset is output by the FORM STIFFNESS command, **EltName.STIFFNESS...mesh**, an EMT object which contains the element stiffness matrix for each interface element.

7.2.8. EI Processor Limitations

Along with the limitations listed in Section 1.5, there are currently limits on problem parameters which may be changed by adjusting internal parameter statements. If adjustments to these limits are required, the COMET-AR maintenance team should be consulted. The current limits are listed in Table 7.10.

Table 7.10. Current Limits on the Interface Element Implementation

Parameter	FORTRAN Parameter	Value
Maximum number of degrees of freedom per node	MaxDoF	6
Maximum number of input geometry points	MaxXYZ	15
Maximum number of pseudo-nodes which may be generated or specified per interface element	MaxPpE	40
Maximum number of alpha-nodes which may be generated per interface element	MaxAlT	60
Maximum number of substructures connected to any one interface element	MaxSpE	4
Maximum number of nodes along the interface per substructure	MaxNpS	50
Maximum number of finite elements along the interface per substructure	MaxEpS	25
Maximum number of interface elements	MaxNIE	30
Maximum total number of nodes per substructure	MaxTnS	20000
Maximum number of finite element types in each substructure	MaxTyp	10
Maximum order of finite elements attached to an interface element	MaxFEo	3

7.2.9. EI Processor Error Messages

The EI processor shell performs error checking each time a data object is manipulated. The processor also checks certain maximum values to ensure that they are within the limits set out in Table 7.10. Additionally, error messages are printed if user input is incorrect; in this case, the user will typically be prompted for the correct input and given the opportunity to re-enter the data.

7.2.10. Examples and Usage Guidelines

7.2.10.1 Example 1: An Example of DEFINE ELEMENTS.

The following procedure defines two interface elements. The first element connects finite element substructures 1 and 2; the second interface element connects finite element substructures 1 and 3. Substructures 1, 2, and 3 reside in libraries 1, 2, and 3 respectively. A nonzero displacement of 0.01 in the 3-direction (in the interface frame) has been applied to the pseudo-nodes of interface element 1. The same element has a zero constraint on the 3-direction rotation on the alpha-nodes in substructure 1 (in the edge frame for substructure 1) and a zero constraint on the 2-direction rotation on the alpha-nodes in substructure 2 (in the edge frame for substructure 2). No additional constraints have been applied to interface element 2. The interface elements will be written to a library with a logical device index of 4 and will be assigned a mesh identification of 0, a load set number of 1, and a constraint set of 1.

```
*procedure EI_Define
. Define Interface Elements
    run  EI1
    . Processor Resets
        reset      ldi        = 4
        reset      mesh       = 0
        reset      step       = 0
        reset      load_set   = 1
        reset      cons_set   = 1
    . Element Definitions
        DEFINE ELEMENTS
        ELEMENT 1 /DSPLINE=3
           CONSTRAINTS
               NONZERO
                   d3 = 0.01
           END_CONSTRAINTS
           SS  1 /LDI=1 /FE /MESH=0 /CONS=1
               NODES = 1,7,2 /GSPLINE=3
               CONSTRAINTS
                   ZERO theta3
               END_CONSTRAINTS
           SS  2 /LDI=2 /FE /MESH=0 /CONS=1
               NODES = 25,50,5 /GSPLINE=3
               CONSTRAINTS
                   ZERO theta2
               END_CONSTRAINTS
        ELEMENT 2 /DSPLINE=3 /SCALE=10000.
           SS  1 /LDI=1 /FE /MESH=0 /CONS=1
               NODES = 35,45,2 /GSPLINE=3
           SS  3 /LDI=3 /FE /MESH=4 /CONS=2
               NODES = 110,200,10 /GSPLINE=3
        END_DEFINE
*end
```

7.2.10.2 Example 2: An Example of DEFINE FREEDOMS

The following example runstream sets the active degrees of freedom for interface elements located in the library with logical device index of 1. The active freedoms are defined for those interface elements associated with mesh 0, load set and constraint set 1.

```
*procedure Defn_EI_Freedoms
. Suppress inactive degrees of freedom
    run EI1
    . Processor Resets
         reset   ldi        = 1
         reset   mesh       = 0
         reset   step       = 0
         reset   load_set   = 1
         reset   cons_set   = 1
    . Issue command to set active freedoms
         DEFINE FREEDOMS
         STOP
*end
```

7.2.10.3 Example 3: An Example of FORM STIFFNESS

The following example runstream forms the element stiffness matrices for interface elements located in the library with logical device index of 1. The stiffness matrices are defined for those interface elements associated with mesh 0, load set and constraint set 1.

```
*procedure Form_EI_Stiffness
. Form Element Stiffness matrices
    run EI1
    . Processor Resets
         reset   ldi        = 1
         reset   mesh       = 0
         reset   step       = 0
         reset   load_set   = 1
         reset   cons_set   = 1
    . Issue command to form stiffness
         FORM STIFFNESS/MATL
         STOP
*end
```

7.2.11. References

7.2-1 Stanley, G. M. and Nour-Omid, S., *The Computational Structural Mechanics Testbed Generic Structural-Element Processor Manual*, NASA Contractor Report 181728, March 1990.

7.2-2 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

7.2-3 Stanley, G. M. and Swenson, L., *HDB Object-Oriented Database Utilities for COMET-AR*, NASA CSM Contract Report, August, 1992.

THIS PAGE INTENTIONALLY BLANK

7.3. Processor El1 - Hybrid Variational (HybV) Interface Element

7.3.1. Element Description

Processor El1 contains a hybrid variational interface element (hereafter referred to as the HybV interface element) which may be used to connect substructures that have been modeled with independently discretized, nodally incompatible finite element models. The element's active degrees of freedom include potentially three displacements and three active rotations per node as well as traction degrees of freedom along the connecting finite element substructures. All incoming substructures must have the same number of active degrees of freedom at each node (*i.e.*, a substructure with five active freedoms per node cannot be connected to a substructure with six active freedoms per node through the HybV interface element). The element formulation has been discussed in detail in References 7.3-1 through 7.3-3, however, key elements of the formulation are reproduced in the following sections. Additional discussion of the implementation of computational reference frames is also included.

7.3.1.1 Theoretical Description

In the following discussion, it is assumed that there is only one interface element in the system. This assumption is made solely to simplify the discussion; the actual implementation is general and accommodates more than one interface element. (Note: current FORTRAN parameter definitions limit the number of interface elements to 30 per analysis; see Section 7.2.8)

Consider, for example, the domain shown in Figure 7.2 and modeled as three independently discretized substructures, Ω_1, Ω_2, and Ω_3. The depiction of three substructures is for discussion purposes only; the element formulation is generally applicable to an arbitrary number of independently discretized substructures. The generally curved interface element path, S_I, is discretized with a mesh of evenly spaced pseudo-nodes (open circles in the figure) which need not be coincident with the interface nodes (filled circles in the figure) of any of the substructures. That is, the discretization of the interface path is independent of the discretization of the connected substructures.

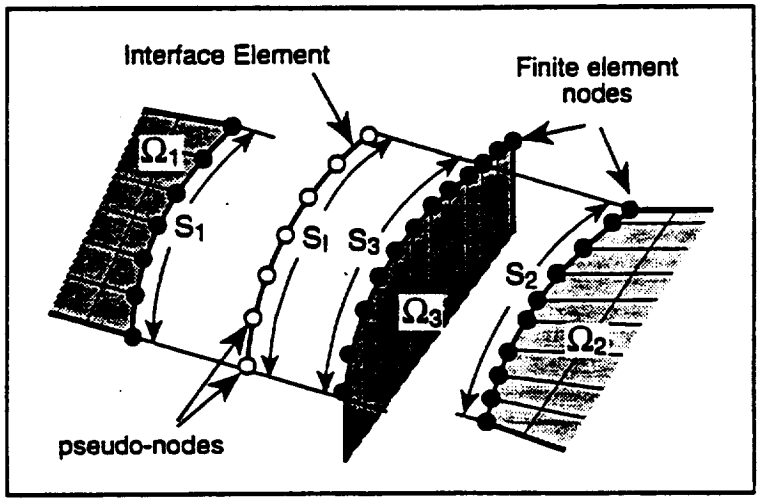


Figure 7.2. HybV Interface Element

The hybrid variational formulation uses an integral form for the compatibility between the interface element and the substructures. The total potential energy is modified by adding a constraint integral for each interface. Namely,

$$\bar{\Pi} = \sum_{k=1}^{N_{ss}} \Pi_k + \sum_{j=1}^{n_{ss}} \left(\int_S \lambda_j^T (\mathbf{v} - \mathbf{u}_j)\, ds_j \right) = \sum_{k=1}^{N_{ss}} \Pi_k + \Pi_c \tag{7.3-1}$$

where $\bar{\Pi}$ is the modified total potential energy of the system, the subscript k denotes the substructure, N_{ss} is the total number of substructures (three for Figure 7.2), n_{ss} is the number of substructures connected to the specific interface element (three for Figure 7.2), $\mathbf{v}$ is the interface element displacement vector (transformed to the edge frame for substructure j), λ_j is a vector of Lagrange multipliers (in the edge frame) for substructure j, $\mathbf{u}_j$ is the displacement vector (in the edge frame) for substructure j, Π_c is the constraint integral, and S is the path of integration along the interface. The potential energy of each substructure, Π_k, is defined as usual by

$$\Pi_k = \tfrac{1}{2}\mathbf{q}_k^T \mathbf{K}_k \mathbf{q}_k - \mathbf{q}_k^T \mathbf{f}_k \tag{7.3-2}$$

where $\mathbf{K}_k$ is the stiffness matrix, $\mathbf{q}_k$ is the generalized displacement vector, and $\mathbf{f}_k$ is the external force vector corresponding to substructure k, all in the finite element nodal computational frame.

The form of the displacement $\mathbf{u}_j$ is assumed as is usual in the finite element method except that it is assumed in the edge frame (the computational frame for the alpha-nodes), namely, $\mathbf{u}_j = \mathbf{u}_{d_j} = \mathbf{N}_{d_j}\mathbf{q}_{d_j}$, where $\mathbf{q}_{d_j}$ is a vector of generalized displacements for the finite element nodes of substructure j along the interface. The vectors $\mathbf{q}_{d_j}$ are transformed subsets of the generalized global displacement vector $\mathbf{q}_k$. The unknown Lagrange multipliers, λ_j, are assumed to be of the form $\lambda_j = \lambda_{d_j} = \mathbf{R}_{d_j}\alpha_{d_j}$.

Along each interface element it is assumed that the displacement, $\mathbf{v}$, is defined in the edge frame by

$$\mathbf{v}_j = \mathbf{v}_{d_j} = \Phi \mathbf{q}_{d_j}^p \tag{7.3-3}$$

where Φ is a matrix of interpolating functions and $\mathbf{q}_d$ is a vector of generalized displacements associated with the images of the pseudo-nodes on substructure j. The interpolation matrix Φ is formed by passing a cubic spline through the evenly spaced pseudo-nodes.

Making the appropriate substitutions, Equation (7.3-1) yields the following expression for the total potential energy:

$$\bar{\Pi} = \sum_{k=1}^{N_{ss}} \left(\tfrac{1}{2}\mathbf{q}_k^T \mathbf{K}_k \mathbf{q}_k - \mathbf{q}_k^T \mathbf{f}_k \right) + \sum_{j=1}^{n_{ss}} \left(\alpha_{d_j}^T \mathbf{G}_j^T \mathbf{q}_s + \alpha_{d_j}^T \mathbf{M}_j^T \mathbf{q}_j \right) \tag{7.3-4}$$

where the vector $\mathbf{q}_s$ is a vector of pseudo-node displacements ($\mathbf{q}_{d_j}^p$ transformed from the edge to the interface frame) and the vector $\mathbf{q}_j$ is a subset of the global nodal computational vector, $\mathbf{q}_k$. The matrices $\mathbf{M}_j$ and $\mathbf{G}_j$ are integrals on the interface which contain transformations and are defined as

$$\mathbf{M}_j = -\int_S \mathbf{T}_{cd_j} \mathbf{N}_{d_j}^T \mathbf{R}_{d_j}\, ds_j \quad \text{and} \quad \mathbf{G}_j = \int_S \mathbf{T}_{sd_j}^p \Phi^T \mathbf{R}_{d_j}\, ds_j \tag{7.3-5}$$

where $\mathbf{N}$, $\mathbf{R}$, and Φ are as previously defined and the transformations, $\mathbf{T}_{cd_j}$ and $\mathbf{T}_{sd_j}^p$ are defined in the following manner.

The matrix T_{cd_j} transforms the q_{d_j} into q_j (*i.e.*, transforms from the edge to nodal computational frame) and is defined by the relationship:

$$q_{d_j} = T_{dc_j} q_j = T_{dg_j} T_{gc_j} q_j \tag{7.3-6}$$

where T_{dg_j} permits the coupling of the alpha-nodes and the finite element nodes and is the global-to-edge frame transformation which may be constructed at each finite element node once the edge frame has been defined and which is constructed by the EI shell and passed to the EI1 kernel. The matrix T_{gc_j} is the nodal-computational-to-global transformation matrix which resides in the NTT data object, exists for each node in each finite element substructure, and is passed down to the EI1 kernel by the EI shell.

The matrix $T^p_{sd_j}$ permits the coupling of the alpha-nodes and the pseudo-nodes and transforms the edge frame pseudo-node displacements for each substructure into the interface frame pseudo-node displacements, the q_s. Interface frame pseudo-node displacements are defined for each substructure in terms of the edge frame pseudo-node displacements as:

$$q_s = T^p_{sd_j} q^p_{d_j} = T_{sg_j} T^p_{gd_j} q^p_{d_j} \tag{7.3-7}$$

The matrix $T^p_{gd_j}$ is the edge-to-global frame transformation for the pseudo-nodes and is constructed at the image of each pseudo-node for each substructure by the EI processor shell. The matrix T_{sg_j} is the global-to-interface transformation which is constructed at each pseudo-node by the EI processor shell.

The hybrid variational interface element "stiffness" matrix thus contains coupling terms which augment the stiffness matrices of the substructures to which each interface element is attached. For the interface element shown in Figure 7.2, the interface element "stiffness" matrix, K_e, and vector of unknowns are given by

$$K_e = \begin{bmatrix} 0 & 0 & M \\ 0 & 0 & G \\ M^T & G^T & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & M_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & M_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & M_3 \\ 0 & 0 & 0 & 0 & G_1 & G_2 & G_3 \\ M_1^T & 0 & 0 & G_1^T & 0 & 0 & 0 \\ 0 & M_2^T & 0 & G_2^T & 0 & 0 & 0 \\ 0 & 0 & M_3^T & G_3^T & 0 & 0 & 0 \end{bmatrix}; \quad \begin{bmatrix} q_c \\ q_s \\ \alpha_d \end{bmatrix} = \begin{bmatrix} q_{c1} \\ q_{c2} \\ q_{c3} \\ q_s \\ \alpha_{d1} \\ \alpha_{d2} \\ \alpha_{d3} \end{bmatrix}. \tag{7.3-8}$$

Users should fully understand the differences among the various computational reference frames. Constraints along the interface, should they be required, must be applied in the appropriate computational frames, namely, the edge frames for the alpha-nodes, the interface frame for the pseudo-nodes, and the nodal computational frame for the finite element nodes. In general, these may all be different frames and a zero value in one frame may well result in multipoint constraints in the other two frames. Additional care must be taken in the application of drilling freedom suppression which is automatic for the pseudo-nodes but may need to be applied to the alpha-nodes. For both pseudo-nodes and alpha-nodes however, the drilling degree-of-freedom is always the sixth degree-of-freedom.

7.3.2. Displacement and Traction Representation

For each interface element, the form of the substructure nodal displacement along the interface, u_d, is assumed in the edge frame using the usual Lagrange shape functions along each substructure edge (*i.e.*, linear functions for 4-node finite element edges; quadratic functions for 9-node finite element edges, etc.). The interface displacement, v_s, is assumed in the interface frame and depends on the function chosen by the user (piecewise linear, or quadratic or cubic spline functions). The unknown Lagrange multipliers, λ_d, are also assumed in the edge frame for each substructure and are taken to be constants when linear finite elements are used along the interface for a given substructure, and linear functions when quadratic finite elements are used along the interface for a given substructure.

7.3.3. Element Geometry and Node Numbering

The HybV interface element is illustrated in Figure 7.3. Finite element nodes are shown as filled circles, pseudo-nodes are shown as open circles. The tractions are attached to the finite elements along the interface of each substructure through alpha-nodes. These alpha-nodes have no actual physical location; they are called nodes only to facilitate the implementation of the interface element. Alpha-nodes are defined according to the finite element type along the substructure edge. Since the tractions are assumed to be linear when 9-node finite elements are used, two alpha-nodes are created for each 9-node finite element along the interface. Likewise, since tractions are assumed to be constant when 4-node finite elements are used, one alpha-node is created for each 4-node element along the interface.

Element node numbering includes the node numbers of the nodes along the interface from each connecting substructure as well as the interface element pseudo-nodes and alpha-nodes. Both the pseudo- and alpha-nodes are generated internally, assigned numbers internally, and are in general, completely transparent to the user. Their node numbers begin at 1 and run sequentially until all are numbered. New pseudo- and alpha-nodes are generated by each interface element. An example of interface element connectivity is shown in Figure 7.3.

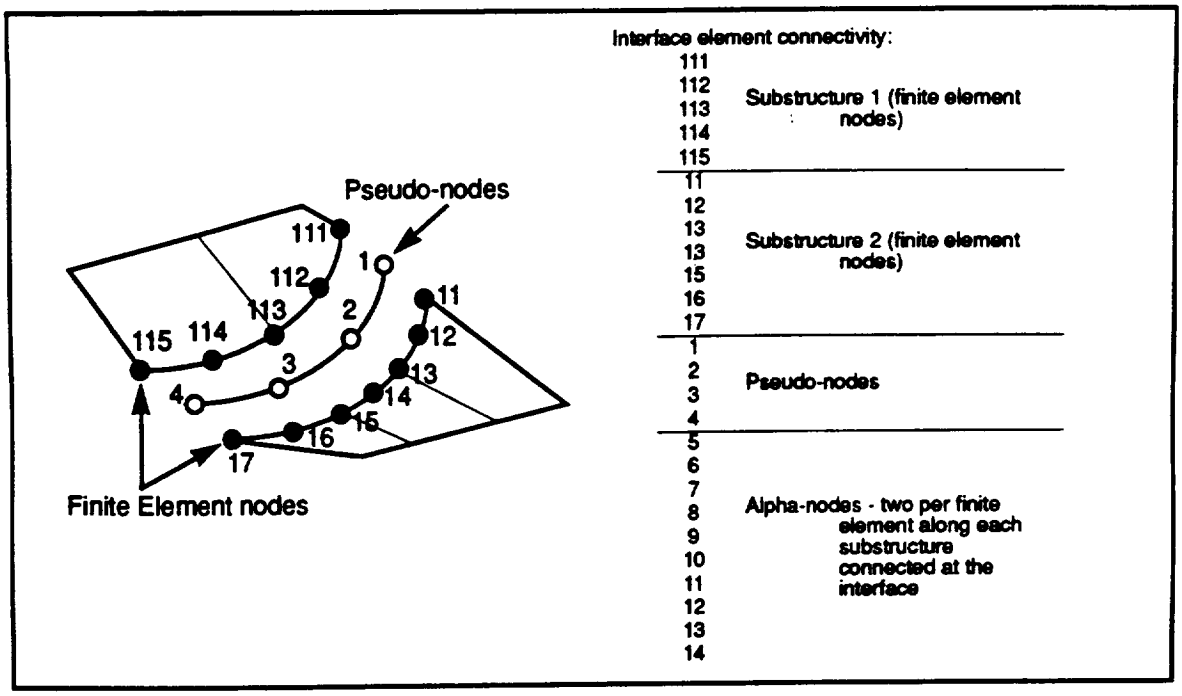


Figure 7.3. Node Numbering for the HybV Interface Element

HybV interface elements can only intersect each other at interface element ends. When two interface elements do intersect, a duplicate pseudo-node is placed at the intersection (*i.e.*, end) point.

7.3.4. Element Implementation Status and Limitations

The HybV interface element is a one dimensional element so it will only join substructures along a line or general curve in space. It may only be used for linear, static, elastic analysis at present although in the future it is expected that a general nonlinearity capability will be developed and implemented. Attached substructures must be modeled using either 4-, 8-, or 9-node quadrilateral or 3- or 6-node triangular finite elements. There is currently no 2-dimensional version of the HybV element (to connect solid models). The limitations on the number of incoming substructures, degrees-of-freedom, and other problem parameters are dictated by the limitations enumerated in Section 7.2.8.

7.3.5. References

7.3-1 Aminpour, M. A., Ransom, J. B., and McCleary, S. L., "Coupled Analysis of Independently Modeled Finite Element Subdomains," AIAA Paper Number 92-2235, 1992.

7.3-2 Aminpour, M.A., McCleary, S.L., and Ransom, J.B., "A Global/Local Analysis Method for Treating Details in Structural Design," *Proceedings of the Third NASA Advanced Composites Technology Conference*, compiled by J.G. Davis, Jr. and H.L. Bohon, NASA CP-3178, Vol. 1, Part 2, 1992, pp. 967-986.

7.3-3 Ransom, J. B., McCleary, S. L., and Aminpour, M. A., "A New Interface Element for Connecting Independently Modeled Substructures," AIAA Paper Number 93-1503, 1993.

THIS PAGE INTENTIONALLY BLANK

8. Master Model Generation

8.1. Overview

In this Chapter, generation of the master model is described. Current assemblers, renumbering strategies, and solvers, all require that element matrices exist in a single library file and that a single global system matrix exist for a given model. The interface element allows the user to keep different models in different library files; therefore, these files must be combined into a single library, or Master Model. The Master Model generator, processor **MSTR**, takes as input any number of substructures and writes out a single database containing one, consolidated, structural model. When interface elements are used, a Master Model must be built using this utility processor regardless of whether the input substructures reside in one or more than one library. The interface elements are always written out to a separate database library and thus will always have to be combined with the substructures to which they are connected.

The Chapter is organized as listed in Table 8.1

Table 8.1. Outline of Chapter 8: Master Model Generation

Section	Processor	Function
2	MSTR	Generate a single master model

THIS PAGE INTENTIONALLY BLANK

8.2. Processor MSTR - Master Model Generator

8.2.1. General Description

The **MSTR** processor takes as input any number of substructures (currently they may be either finite element or interface element substructures) and creates a single, consolidated structural model. It performs this merging of substructures by stacking all of the nodes in a list (*i.e.*, nodes from substructure 1, nodes from substructure 2, etc.) and relabeling them sequentially. Nodes from the interface element substructure are added at the end of the node list, with pseudo-nodes listed first and alpha-nodes following. The same stacking and relabeling is also performed on the element definitions. Once the nodes have been relabeled, the element connectivities are changed to reflect the new node labels. All the data needed to solve the system of equations are saved based on the new node and element labels. It should be noted that at no time in this process is the original data changed; the **MSTR** processor creates an entirely new model, in a library separate from the original substructures' data libraries. The **MSTR** processor also has a post-processing function which permits the user to split the **NODAL.DISPLACEMENT.*** results data object of the Master Model into substructure objects for further post-processing.

8.2.2. Command Classes

The **MSTR** processor recognizes three command classes as listed in Table 8.2. Each of these command classes has different keywords and additional subcommands which are described in the following sections.

Table 8.2. Master Model Generator (Processor MSTR) Command Classes

Command Class	Function
DEFINE	Defines substructures to be merged.
MERGE	Merges specified substructures into a single master model.
POST_PROCESS	Splits the master model results data back into substructure results data for those substructures specified.

8.2.3. The MSTR Processor DEFINE Command

The DEFINE command class currently has only one form, DEFINE SUBSTRUCTURES. This command has several associated subcommands and additional qualifiers. The DEFINE SUBSTRUCTURES command must be issued prior to both the MERGE and the POST_PROCESS commands or the processor will not know which substructures need to be merged or post-processed. A template for execution of the DEFINE SUBSTRUCTURES command is provided as follows:

```
DEFINE SUBSTRUCTURES
    SUBSTRUCTURE i /FE
        LIBRARY          = ildi
        MESH             = imesh
        LOAD_SET         = iiset
        CONSTRAINT_CASE  = incon
        LOAD_STEP        = iistep
    SUBSTRUCTURE j /IE
        LIBRARY          = jldi
        MESH             = jmesh
        LOAD_SET         = jiset
        CONSTRAINT_CASE  = jncon
        LOAD_STEP        = jistep
END_DEFINE
```

Each of the subcommands and qualifiers are discussed in detail in subsequent sections.

8.2.3.1 The SUBSTRUCTURE Subcommand

The SUBSTRUCTURE subcommand signifies that a new substructure definition is beginning. These substructures are usually the same as those identified during the DEFINE ELEMENTS process using the EI processor.

Subcommand syntax:

```
SUBSTRUCTURE i {/FE /BE /RR /IE}
```

where *i* is an integer identifying the substructure number and the optional qualifiers are described in the following subsection.

8.2.3.1.1 *The /FE, /BE, /RR, /IE Qualifiers*

This set of qualifiers identifies the form of the idealization of the substructure. `/FE` identifies a finite element substructure, `/BE` identifies a boundary element substructure, `/RR` identifies a Rayleigh/Ritz substructure, and `/IE` identifies a substructure which contains interface elements. Currently, ***only the /FE and /IE qualifiers can be used***; COMET-AR has no current capability for boundary element or Rayleigh/Ritz substructures. This set of qualifiers is a mutually exclusive set (*i.e.*, a substructure can have no more than one of these qualifiers). (Default value: None)

8.2.3.2 The LIBRARY Subcommand

The LIBRARY subcommand identifies the logical device index of the library containing this substructure.

Subcommand syntax:

LIBRARY = *ldi*

where *ldi* is an integer identifying the logical device index number of the library. (Default value:1)

8.2.3.3 The MESH Subcommand

The MESH subcommand identifies the mesh number of the current substructure.

Subcommand syntax:

MESH = *mesh*

where *mesh* is an integer identifying the mesh number of the current substructure. (Default value: 0)

8.2.3.4 The LOAD_SET Subcommand

The LOAD_SET subcommand identifies the load set number for the current substructure.

Subcommand syntax:

LOAD_SET = *load_set*

where *load_set* is an integer identifying the load set number of the current substructure. (Default value: 1)

8.2.3.5 The CONSTRAINT_CASE Subcommand

The CONSTRAINT_CASE subcommand identifies the constraint case number for the substructure.

Subcommand syntax:

CONSTRAINT_CASE = *constraint_case*

where *constraint_case* is an integer identifying the constraint case for this substructure. (Default value:1)

8.2.3.6 The LOAD_STEP Subcommand

The LOAD_STEP subcommand identifies the load step number for this substructure.

Subcommand syntax:

LOAD_STEP = *load_step*

where *load_step* is an integer identifying the load step number for this substructure. For linear analyses, *load_step* should be zero. (Default value: 0)

8.2.3.7 The END_DEFINE Subcommand

The END_DEFINE subcommand signals the end of substructure definitions.

Subcommand syntax:

END_DEFINE

8.2.4. The MERGE (or MERGE_SUBSTRUCTURES) Command

This command is used to trigger the merging of the identified substructures. The input to the MERGE command is a list of substructures to be merged into the master model (for example, `MERGE 1,3,4` implies that substructures 1, 3, and 4 are to be merged into the master model). The command requires that the substructures be listed using the substructure identifier of the DEFINE SUBSTRUCTURES command (*e.g.*, the substructure identified as substructure 2 when defined will be merged as substructure 2). A template for execution of the MERGE command follows:

MERGE *i,j,k* OR MERGE_SUBSTRUCTURES *i,j,k*		
LIBRARY	=	*ldi*
FILE	=	*file_name*
MESH	=	*mesh*
LOAD_SET	=	*iset*
CONSTRAINT_CASE	=	*ncon*
LOAD_STEP	=	*istep*
EAT	=	*data_object_name*
NAT	=	*data_object_name*
SVT	=	*data_object_name*
STOP		

Each of the subcommands are discussed in detail in subsequent sections.

8.2.4.1 The LIBRARY Subcommand

The LIBRARY subcommand identifies the logical device index of the library which will contain the merged master model.

Subcommand syntax:

LIBRARY = *ldi*

where *ldi* is an integer identifying the logical device index number of the library. If the LIBRARY subcommand is not issued, MSTR will use the next available logical device index. (Default value: None)

8.2.4.2 The FILE Subcommand

The FILE subcommand identifies the name of the library which will contain the merged master model.

Subcommand syntax:

FILE = *file_name*

where *file_name* is a character string identifying the name of the master model library. If the LIBRARY subcommand is not issued, **MSTR** will assign the next available logical device index to *file_name*. (Default value: None)

8.2.4.3 The MESH Subcommand

The MESH subcommand identifies the mesh number assigned to the merged master model.

Subcommand syntax:

MESH = *mesh*

where *mesh* is an integer identifying the mesh number of the merged model. (Default value: 0)

8.2.4.4 The LOAD_SET Subcommand

The LOAD_SET subcommand identifies the load set number assigned to the merged master model.

Subcommand syntax:

LOAD_SET = *load_set*

where *load_set* is an integer identifying the load set number of the merged model. (Default value: 1)

8.2.4.5 The CONSTRAINT_CASE Subcommand

The CONSTRAINT_CASE subcommand identifies the constraint case number assigned to the merged master model.

Subcommand syntax:

CONSTRAINT_CASE = *constraint_case*

where *constraint_case* is an integer identifying the constraint case for the merged model. (Default value: 1)

8.2.4.6 The LOAD_STEP Subcommand

The LOAD_STEP subcommand identifies the load step number assigned to the merged master model.

Subcommand syntax:

LOAD_STEP = *load_step*

where *load_step* is an integer identifying the load step number for the merged model. For linear analyses, *load_step* should be set to zero. (Default value: 0)

8.2.4.7 The EAT Subcommand

The EAT subcommand identifies additional element attribute tables which are to be merged for the listed substructures.

Subcommand syntax:

EAT = *data_object_name*

where *data_object_name* is a character string identifying the **EAT** to be merged. The **EAT** must exist in all of the merged substructure libraries. (Default value: None) ***NOT OPERATIONAL***.

8.2.4.8 The NAT Subcommand

The NAT subcommand identifies additional nodal attribute tables which are to be merged for the listed substructures.

Subcommand syntax:

NAT = *data_object_name*

where *data_object_name* is a character string identifying the **NAT** to be merged. The **NAT** must exist in all of the merged substructure libraries. (Default value: None) ***NOT OPERATIONAL***.

8.2.4.9 The SVT Subcommand

The SVT subcommand identifies additional system vector tables which are to be merged for the listed substructures.

Subcommand syntax:

SVT = *data_object_name*

where *data_object_name* is a character string identifying the **SVT** to be merged. The **SVT** must exist in all of the merged substructure libraries. (Default value: None) ***NOT OPERATIONAL***.

8.2.5. The POST_PROCESS Command

This command is used to post-process the master model. It splits the master model into its component substructures once the solution has been obtained. This splitting process is typically done to facilitate the recovery of stresses for the individual substructures. The input to the POST_PROCESS command is a list of substructures to be split from the master model (for example, `POST_PROCESS 1,3,4` implies that the displacement results from substructures 1, 3, and 4 are to be split from the master model and placed in the substructure libraries). The POST_PROCESS command requires that the substructures be identified using the same identifiers used during the DEFINE SUBSTRUCTURES command (*e.g.*, the substructure identified as substructure 2 when defined will be post-processed as substructure 2). In fact, it is currently required that the DEFINE SUBSTRUCTURES command be reissued. A template for execution of the POST_PROCESS command follows:

```
POST_PROCESS i,j,k
        LIBRARY          =  ldi
        MESH             =  mesh
        LOAD_SET         =  iset
        CONSTRAINT_CASE  =  ncon
        LOAD_STEP        =  istep
END_POST
STOP
```

Each of the subcommands are discussed in detail in subsequent Sections.

8.2.5.1 The LIBRARY Subcommand

The LIBRARY subcommand identifies the logical device index of the library containing the Master Model.

Subcommand syntax:

```
LIBRARY = ldi
```

where *ldi* is an integer identifying the logical device index number of the library. (Default value:1)

8.2.5.2 The MESH Subcommand

The MESH subcommand identifies the mesh number of the merged model.

Subcommand syntax:

```
MESH = mesh
```

where *mesh* is an integer identifying the mesh number of the merged model. (Default value: 0)

8.2.5.3 The LOAD_SET Subcommand

The LOAD_SET subcommand identifies the load set number for the merged model.

Subcommand syntax:

```
LOAD_SET = load_set
```

where *load_set* is an integer identifying the load set number of the merged model. (Default value:1)

8.2.5.4 The CONSTRAINT_CASE Subcommand

The CONSTRAINT_CASE subcommand identifies the constraint case number for the merged model.

Subcommand syntax:

CONSTRAINT_CASE = *constraint_case*

where *constraint_case* is an integer identifying the constraint case for the merged model. (Default value:1)

8.2.5.5 The LOAD_STEP Subcommand

The LOAD_STEP subcommand identifies the output load step number for the merged model.

Subcommand syntax:

LOAD_STEP = *load_step*

where *load_step* is an integer identifying the load step number for the merged model. For linear analyses, *load_step* should be set to zero. (Default value: 0)

8.2.5.6 The END_POST Subcommand

The END_POST subcommand signals the end of processing for the POST_PROCESS command. The command is required input.

Subcommand syntax:

END_POST

8.2.6. Database Input/Output

8.2.6.1 Input Datasets

Several datasets are required by **MSTR** during the merge process; others are optional and will be merged if they are present in one or more of the substructures (*e.g.*, nodal forces). Table 8.3 lists those datasets required for both the merge and the post-processing options. Datasets which are optional (*i.e.*, not required) are indicated with an asterisk on the Type (*e.g.*, **EAT*** is an optional **EAT** object). All datasets listed appear in both the substructure and the interface element data libraries. The following definitions apply: *mesh* is the mesh number; *concase* is the constraint case number; *ldset* is the load set number and *EltName* is the finite or interface element name.

Table 8.3. Input Datasets Required by MSTR Processor

Function	Dataset	Description	Type
MERGE SS	CSM.SUMMARY...*mesh*	Model summary	CSM
	NODAL.COORDINATE...*mesh*	Nodal coordinates	NCT
	NODAL.DOF..*concase.mesh*	Constraints	NDT
	NODAL.EXT_FORCE.*ldset..mesh*	Applied nodal forces	NVT*
	NODAL.SPEC_DISP.*ldset..mesh*	Specified displacements	NVT*
	NODAL.TRANSFORMATION...*mesh*	Nodal global-to-local transformations	NTT
	NODAL.TYPE...*mesh*	Node types	NAT
	EltName.DEFINITION...*mesh*	Element definitions	EDT
	EltName.PARAMS...*mesh*	Interface element parameters	EAT
	EltName.MATRIX...*mesh*	Element stiffness matrices	EMT
POST-PROCESS	CSM.SUMMARY...*mesh*	Model summary	CSM
	NODAL.NIDS...*mesh*	Original nodal identifiers	NAT
	NODAL.DISPLACEMENT.*ldset.concase.mesh*	Solution vector	NVT

8.2.6.2 Output Datasets

Several datasets are created by **MSTR** during the merge process; some of these are considered optional and will be created only if they are present in one or more of the substructures (*e.g.*, nodal forces). Table 8.4 lists those datasets which are created either always or optionally for both the merge and the post-processing options. The datasets listed as active for the merge process will be written to the master model library; the post-processing datasets will be written to the individual substructure objects. Datasets which are optional (*i.e.*, not required) are indicated with an asterisk on the Type (*e.g.*, **EAT*** is an optional **EAT** object). The following definitions apply: *mesh* is the mesh number; *concase* is the constraint case number; *ldset* is the load set number and *EltName* is the finite or interface element name.

Table 8.4. Output Datasets Created/Modified by the MSTR Processor

Function	Dataset	Description	Type
MERGE SS	CSM.SUMMARY...*mesh*	Model summary	CSM
	NODAL.COORDINATE...*mesh*	Nodal coordinates	NCT
	NODAL.DOF..*concase.mesh*	Constraints	NDT
	NODAL.EXT_FORCE.*ldset..mesh*	Applied nodal forces	NVT*
	NODAL.NIDS.*ldset..mesh*	Original node labels (numbers)	NAT
	NODAL.SPEC_DISP.*ldset.concase.mesh*	Specified displacements	NVT*
	NODAL.TRANSFORMATION...*mesh*	Nodal global-to-local transformations	NTT
	NODAL.TYPE...*mesh*	Node types	NAT
	EltName.DEFINITION...*mesh*	Element definitions	EDT
	EltName.MATRIX...*mesh*	Element stiffness matrices	EMT
POST-PROCESS	NODAL.DISPLACEMENT.*ldset.concase.mesh*	Solution vector	NVT

8.2.7. Processor Limitations

Along with the limites listed in Section 1.5, there are currently limits on many of the problem parameters which may be changed by adjusting internal parameter statements. If adjustments on these limits are required, the COMET-AR maintenance team should be consulted. The current limits are listed in Table 8.5.

Table 8.5. Current Limits on the Master Model Generation

Parameter	Value
Maximum number of degrees of freedom per node (maxDoF)	6
Maximum total number of pseudo-nodes model-wide (maxnPn)	1000
Maximum total number of alpha-nodes model-wide (maxnAn)	2000
Maximum total number of nodes per substructure	20000
Maximum number of finite element types	10
Maximum number of element types (including interface elements)	100
Maximum number of substructures	5
Maximum total number of nodes (in the master model)	50000

8.2.8. Processor Error Messages

The **MSTR** processor performs error checking each time a data object is manipulated. The processor also checks certain maximum values to ensure that they are within the limits set out in Table 8.5. Additionally, error messages are printed if user input is incorrect; in this case, the user will typically be prompted for the correct input and given the opportunity to re-enter the data.

8.2.9. Examples and Usage Guidelines

8.2.9.1 Example 1: An Example of Merging two Finite Element Substructures with an Interface Element Substructure

The following example runstream combines the finite element models labeled as substructures 1 and 2 and the interface element substructure labeled as substructure 3and creates a new model which is written to library 4 with the file name ***master.model***. This new master model carries the mesh identifier of 0, a load set number of 1, and a constraint case of 1.

```
Run MSTR
    DEFINE SUBSTRUCTURES
        SUBSTRUCTURE 1 /FE
              LIBRARY           = 1
              MESH              = 0
              LOAD_SET          = 1
              CONSTRAINT_CASE   = 1
              LOAD_STEP         = 0
        SUBSTRUCTURE 2 /FE
              LIBRARY           = 2
              MESH              = 0
              LOAD_SET          = 2
              CONSTRAINT_CASE   = 1
              LOAD_STEP         = 0
        SUBSTRUCTURE 3 /IE
              LIBRARY           = 4
              MESH              = 0
              LOAD_SET          = 1
              CONSTRAINT_CASE   = 1
              LOAD_STEP         = 0
    END_DEFINE
    MERGE SUBSTRUCTURES 1,2,4
        LIBRARY                 = 3
        FILE                    = 'master.model'
        MESH                    = 0
        LOAD_SET                = 1
        CONSTRAINT_CASE         = 1
    STOP
```

8.2.9.2 Example 2: An Example of Post-processing the Master Model into two Finite Element and one Interface Element Substructures

The following example runstream takes the master model which resides in library 3 and splits off results for finite element substructures labeled as substructures 1 and 2 and for the interface element substructure, labeled as substructure 3.

```
Run MSTR
    DEFINE SUBSTRUCTURES
        SUBSTRUCTURE 1 /FE
              LIBRARY            = 1
              MESH               = 0
              LOAD_SET           = 1
              CONSTRAINT_CASE    = 1
              LOAD_STEP          = 0
        SUBSTRUCTURE 2 /FE
              LIBRARY            = 2
              MESH               = 0
              LOAD_SET           = 2
              CONSTRAINT_CASE    = 1
              LOAD_STEP          = 0
        SUBSTRUCTURE 3 /IE
              LIBRARY            = 4
              MESH               = 0
              LOAD_SET           = 1
              CONSTRAINT_CASE    = 1
              LOAD_STEP          = 0
    END_DEFINE
    POST_PROCESS 1,2,4
        LIBRARY                  = 3
        MESH                     = 0
        LOAD_SET                 = 1
        CONSTRAINT_CASE          = 1
    END_POST                     = 1
    STOP
```

8.2.10. References

None.

Part V.
DEVELOPER INTERFACE

THIS PAGE INTENTIONALLY BLANK

9. Developer Interface

9.1. Overview

The interface element processor, EI, is composed of three parts: the generic EI processor shell, the user-written EI processor cover, and the user-written EI processor kernel. The generic shell manages all interaction between the user and the individual EI processor (using CLIP routines described in Ref. 9.3-1) as well as all interaction between the database and the individual EI processor (using HDB routines described in Ref. 9.3-2). The developer of new interface elements will need to access and modify only two files: the ***ei*_cover.ams*** file and the ***ei*_kernel.ams*** file. This Chapter contains a description of the generic shell and the requirements for the cover routines. The Chapter is organized as follows:

Table 9.1. Outline of Chapter 9: Developer Interface

Section	Subject	Function
2	New qSymbols	Definitions of new qSymbols.
3	EI_shell	Description of the uniform user and database interface for all interface element processors.
4	EI_cover	Description of the software that translates between the shell and the kernel routines.
5	makefile	Example makefile.

THIS PAGE INTENTIONALLY BLANK

9.2. New qSymbols

9.2.1. General Description

A qSymbol (Ref. 9.2-1) is simply a FORTRAN integer parameter which is usually used in place of a character string. There are currently several hundred of these parameters used in COMET-AR. During the implementation of the interface element, it was found that new qSymbols were needed. Ten new parameters were added to the ***qsymbol.inc*** file using the method outlined in Ref. 9.2-1. These new parameters are listed in Table 9.2.

Table 9.2. New qSymbol Parameters

Parameter	Value	Usage
qAlpha	281	Denotes alpha-nodes (this is not a new qsymbol; this is an additional, new, meaning assigned to the existing parameter).
qBE	380	Identifies boundary element substructures
qD	385	Denotes pseudo-nodes
qFE	379	Identifies finite element substructures
qFInd	376	Signals the need to locate (find) finite element nodes along a specified path
qForm	378	Signals the need to form a path through the pseudo-nodes
qGet	377	Signals the need to form a path through specified finite element nodes
qIE	382	Identifies an interface element substructure
qPost	386	Identifies the post-processing function of processor **MSTR**
qRR	381	Identifies Rayleigh/Ritz substructures

9.2.2. References

9.2-1 Stanley, G. M. and Swenson, L., *HDB Object-Oriented Database Utilities for COMET-AR*, NASA CSM Contract Report, August, 1992.

THIS PAGE INTENTIONALLY BLANK

9.3. The Generic Interface Element Processor Shell

9.3.1. General Description

The developer of a new interface element must create his or her own kernel subroutines and must generate a corresponding ***el*_cover.ams*** file. This cover file translates between the user kernel routines and the EI shell which performs all of the database manipulation. The following sections provide a summary of each subroutine in the EI shell file, ***el*_shell.ams*** , including its function, argument list (if any), include files used, and special requirements.

Element names are assigned by the element developer. The shell however assumes that each interface element defined by the user is a different element type (since each element can, and usually does, have a unique number of nodes). The convention adopted by the **El1** processor is that the element name is composed of the element processor name, the element type name, and the element number all separated by underscores (*e.g.*, **El1_HybV_1** is the name of element 1, **El1_HybV_5** is the name of element 5). This element name is used to name element table datasets. It is strongly recommended that developers of new elements adopt the same naming convention; this will provide uniformity and minimize confusion for users.

9.3.2. Shell Include Files

The ***el*_shell.ams*** file uses a number of include files which are also available for use by the element developer. These include files generally contain common blocks, parameter definitions, and type declarations for variables used throughout the processor. One include file is used by virtually all subroutines, ***qsymbol.inc***. This file is described in detail in the HDB Manual (Ref. 9.3-2) and the reader is referred to that document for specifics on the use of qsymbols (integer parameters which all begin with the letter "q" and which generally replace character data). Several new qSymbols have been added and are described in Section 9.2. Each of the remaining include files is summarized in the Table 9.3 and listed in subsequent Sections. Variables and parameters are also defined.

Table 9.3. Summary of Include Files

File	Description	Section
el0lim.inc	Sets limits on problem parameters (e.g., maximum number of nodes, number of substructures)	**9.3.2.1**
el0cmn.inc	Common block data	**9.3.2.2**
el0com.inc	Common block data	**9.3.2.3**
el0ptr.inc	Pointer array parameter and common blocks	**9.3.2.4**

9.3.2.1 The *ei0lim.inc* Include File

The file ***ei0lim.inc*** contains parameter definitions which are used throughout the processor. These parameters set the maximum permissible values for various items such as number of nodes, number of degrees of freedom per node, and number of interface elements. These maximums are used to dimension arrays in other include files as well as in subroutines. A listing of the include file is provided as Table 9.4.

Table 9.4. Listing of the *ei0lim.inc* Include File

```
c_beg EI0LIM.INC
      integer MAXDOF ! Maximum number of degrees of freedom per node
      parameter ( MAXDOF  =   6 )
      integer MAXNIP ! Maximum number of integration points per element
      parameter ( MAXNIP = 144 )
      integer MaxXYZ ! Maximum number of input geometry points
      parameter ( MaxXYZ =  15 )
      integer MaxPpE ! Maximum number of pseudo-nodes per interface element
      parameter ( MaxPpE =  40 )
      integer MaxAlT ! Maximum number of alpha nodes per interface element
      parameter ( MaxAlT =  60 )
      integer MaxSpE ! Maximum number of substructures per element
      parameter ( MaxSpE =   4 )
      integer MaxNpS ! Maximum number of nodes per substructure (along the interface)
      parameter ( MaxNpS =  50 )
      integer MaxEpS ! Maximum number of elements per substructure (along interface)
      parameter ( MaxEpS = MaxNpS/2)
      integer MaxNEN ! Maximum number of nodes per interface element
      parameter ( MaxNEN = MaxSpE*MaxNpS+MaxAlT+MaxPpE )
      integer MaxTnS ! Maximum total number of nodes per substructure
      parameter ( MaxTnS = 20000 )
      integer MaxTyp ! Maximum number of finite element types
      parameter ( MaxTyp =  10 )
      integer MaxFEo ! Maximum order of finite elements
      parameter ( MaxFEo =   3 )
      integer MaxNG  ! Maximum number of interface geometry nodes
      parameter ( MaxNG  = MaxNpS*MaxSpE )
      integer MaxPAR ! Maximum number of miscellaneous element parameters
      parameter ( MaxPAR = 3*MaxSpE+10 )
      integer MaxNEE ! Maximum number of element equations
      parameter ( MaxNEE = MaxDOF*MaxNEN )
      integer MaxNUT ! Maximum number of items in the upper triangle
      parameter ( MaxNUT = MaxNEE*(MaxNEE+1)/2 )
      integer MaxEdg ! Maximum number of element edges per finite element
      parameter ( MaxEdg = 16 )
      integer MaxNIE ! Maximum number of interface elements
      parameter ( MaxNIE = 30 )
      integer MaxInd ! Maximum number of independent dofs for mpcs
      parameter ( MaxInd = 20 )
      integer MaxMPC ! Maximum number of mpcs along interface
      parameter ( MaxMPC =  6 )
c_end EI0LIM.inc
```

9.3.2.2 The *ei0cmn.inc* Include File

The file ***ei0cmn.inc*** contains several common blocks and type declarations. A summary of the common blocks and a general description of their contents follows in Table 9.5. A complete listing of the include file is provided as Table 9.6.

Table 9.5. Summary of Common Blocks in *ei0cmn.inc*

Common Block	Data Type	Contents
EI0CIB	Integer	Contains integer data for the substructures
EI0CFB	Single or Double	Contains floating point data for both the substructures and the interface elements
EI0CCB	Character	Contains character data for both the substructures and the interface elements
EI0IED	Integer	Contains integer data for the interface elements
EI0CON	Integer	Contains integer constraint data
EI0EID	Single or Double	Contains floating point constraint data for both pseudo- and alpha-nodes
EI0EIC	Character	Contains character representation of constraints
EI0EII	Integer	Contains integer constraint data

Table 9.6. Listing of the *ei0cmn.inc* Include File

```
c_beg ei0cmn.inc
      integer Gcurv,   Dspline, NumElty, nss,  Gspline
      integer ssid,    ssldi,   ssmesh,  ssnn, ssnode, ssns,    ssdofs,
     1        sscons,  sstelt,  ssnelt,  ssNet,ssDofn, ssNdofn, ssend,
     2        ssCSM,   ssNen,   ssMdofn, ssFE, ssBE,   ssIE,
     3        ssRR,    ssNnode, ssINelt, sstn, nFilter,
     4        Filterx, Filtery, Filterz, ssnact, ssActv, ssfid
c
      common /EI0CIB/  Gcurv,   Dspline, NumElty, nss, Gspline,
     1                 ssid(MaxSpE),   ssldi(MaxSpE), ssmesh(MaxSpE),
     2                 ssnn(MaxSpE),   ssnode(MaxNpS,MaxSpE),
     3                 ssns(MaxXYZ),   ssdofs(MaxDOF,MaxNpS,MaxSpE),
     4                 sscons(MaxSpE), sstelt(MaxTyp,MaxNpS,MaxSpE),
     5                 ssnelt(MaxNpS,MaxSpE), ssNet(MaxSpE),
     6                 ssDofn(MaxDOF,MaxSpE), ssNdofn(MaxSpE),
     7                 ssend(2,MaxSpE),
     8                 ssCSM(MaxSpE),  ssNen(MaxTyp,MaxSpE),
     9                 ssMdofn(MaxSpE),ssFE(MaxSpE),  ssBE(MaxSpE),
     $                 ssIE(MaxSpE),   ssRR(MaxSpE),  ssNnode(MaxSpE),
     1                 ssINelt(MaxTyp,MaxSpE),
     2                 sstn(MaxSpE),
     3                 nFilter(2,MaxSpE), Filterx, Filtery, Filterz,
     4                 ssActv(MaxTnS,MaxSpE), ssnAct(MaxSpE),
     5                 ssfid
c
c    Definitions:
c      ssid:    substructure id's
c      ssldi:   substructure libraries
c      ssmesh:  substructure mesh number
c      ssnn:    number of interface nodes per substructure
c      ssnode:  list of interface nodes for each substructure
```

Table 9.6. Listing of the *ei0cmn.inc* Include File(Continued)

```
c      ssns:    number of interface geometry nodes for each substructure
c      ssdofs:  list of active dofs for each node for each substructure
c      sscons:  substructure constraint set number
c      sstelt:  list of element types connected to each interface node
c      ssnelt:  number of element types connected to each interface node
c      ssNet:   number of element types in each substructure
c      ssDofn:  List of active nodal DOF types substructure wide
c      ssNdofn: Number of active nodal DOF types substructure wide
c      ssCSM:   id's for the substructure CSM objects
c      ssNen:   Number of nodes per finite element type for each substructure
c      ssFE:    Flags (when = qFE) denoting finite element substructure
c      ssBE:    Flags (when = qBE) denoting boundary element substructure
c      ssIE:    Flags (when = qIE) denoting other existing interface elements
c      ssRR:    Flags (when = qRR) denoting Rayleigh-Ritz substructure
c      ssNnode: Number of nodes in the entire substructure
c      ssINelt: Number of element of each element type in each substructure
c      sstn:    Total number of nodes per substructure.
c      Filter*: Flags to indicate that the filters are on
c      ssActv:  List of active nodes for the "Find" operation of path routine
c      ssnAct:  Number of active nodes for the "Find" operation
c      ssfid:   ID of the "fine" substructure (with most nodes along interface)
C=IF DOUBLE
      double precision
C=ELSE
      real
C=ENDIF
     1        xFilter, yFilter, zFilter, xyzss, coordss, coordei,
     2        ssforc,  Gxyz,    pathss,  pathei,ssxyz,   scale,
     3        tangei,  tangss,  zero,  normsse, normssn, normei,
     4        tranei,  transs,  ssTdg, ssTgc,   ssTcg,   eiTdg,   eiTgs
c
      common /EI0CFB/  xFilter(2), yFilter(2),
     1                 zFilter(2), xyzss(3,MaxXYZ),
     2                 coordss(3,MaxNpS,MaxSpE),
     3                 coordei(3,MaxPpE),ssforc(MaxDOF,MaxNpS,MaxSpE),
     4                 Gxyz(3,MaxNg),    pathss(MaxNpS,MaxSpE),
     5                 pathei(MaxPpE),
     6                 ssxyz(3,MaxTns,MaxSpE), scale,
     7                 tangei(3,MaxPpE), tangss(3,MaxNpS,MaxSpE),
     9                 zero,
     $                 normsse(3,MaxTyp,MaxNpS,MaxSpE),
     1                 normssn(3,MaxNpS,MaxSpE),normei(3,MaxPpE),
     2                 tranei(3, MaxPpE), transs(3,MaxNpS,MaxSpE),
     3                 ssTdg(3,3,MaxNpS,MaxSpE),
     4                 ssTgc(3,3,MaxNpS,MaxSpE),
     5                 ssTcg(3,3,MaxNpS,MaxSpE),
     6                 eiTdg(3,3,MaxPpE,MaxSpE),
     7                 eiTgs(3,3,MaxPpE)
c
c     xFilter: bounds on x-coordinates when interface nodes must be found
c     yFilter: bounds on y-coordinates when interface nodes must be found
c     zFilter: bounds on z-coordinates when interface nodes must be found
c     xyzss:   coordinates of points (not nodes) used to define path
c     coordss: coordinates along the interface for the interface nodes
c     coordei: coordinates along the interface for the pseudo-nodes.
c     Gxyz:    concatinated geometry coordinates along interface
c     pathss:  coordinates along the path for the interface nodes
c     pathei:  coordinates along the path for the pseudo-nodes.
```

Table 9.6. Listing of the *ei0cmn.inc* Include File(Continued)

```
c     ssxyz:   xyz coordinates for all nodes in each substructure.
c     scale:   scale factor for interface element.
c     tangei:  tangents (along the path s) for the pseudo-nodes.
c     tangss:  tangents (along the path s) for finite element nodes.
c     ssTcg:   nodal global-to-computational transformations
c     zero:    zero value
c     normsse: element nodal normals for each finite element along interface
c     normssn: average nodal normal based on only f.e.s along the interface
c     normei:  pseudo-node normal
c     tranei:  transverse surface tangent for pseudo-nodes
c     transs:  transverse surface tangent for finite element nodes
c     ssTdg:   nodal global-to-edge frame transformation
c     eiTdg:   pseudo-nodal global-to-edge frame transformations
c     eiTgs:   pseudo-nodal interface element path-to-global transformation
c
      character*40 EltNam, EltPro, EltTyp
      character*40 sscElt
c
      common /EI0CCB/  EltNam, EltPro, EltTyp, sscElt(MaxTyp,MaxSpE)
c
c     EltNam:  element name for interface element; EltPro_EltTyp.
c     EltPro:  element processor name for interface element
c     EltTyp:  element type
c     sscElt:  element names of substructure elements.
c
      integer  Npn,     pnid,   nAlpha,  nAlphaT, alid
      integer  ieNen,   ieDofn, ieNdofn, ieConn,  ieNodSS, anodes
      integer  DrilDof, DrilSup
c
      common /EI0IED/ aNodes(MaxSpE), Npn,
     2                pnid(MaxPpE),   nAlpha(MaxSpE),
     3                nAlphaT,        alid(MaxAlT),
     4                ieNen, ieNdofn, ieDofn(MaxDOF),
     5                ieConn(MaxNEN), ieNodSS(MaxNEN),
     6                nApE(MaxSpE),   Net,
     7                DrilDof,        DrilSup
c
c     aNodes:  number of alpha-type nodes per substructure
c     Npn:     Total number of interface element pseudo-nodes
c     pnid:    pseudo-node "node" numbers
c     nAlpha:  number of alpha dofs for each SS of each interface element
c     nAlphaT: Total number of alphas
c     alid:    "node" number for the alphas (placed Ndofn per "node")
c     ieNen:   Total number of interface element "nodes" - Npn + nAlphaT +
c                number of nodes along each substructure
c     ieDofn:  Active dofs for the interface element
c     ieNdofn: Number of active dofs for the interface element
c     ieConn:  Interface element connectivity
c     ieNodSS: Substructure id's corresponding to nodes in element connectivity
c     nApE:    number of alphas per finite element
c     Net:     Number of element types
c     DrilDof: Drilling freedom for interface element
c     DrilSup: Drilling freedom suppression flag
c
c /-/-/-/-/-/-/-/- CONSTRAINT COMMON BLOCKS SS and EI -/-/-/-/-/-/-/-/
c
      integer ssState, nssmpc, issmpc
c
```

Table 9.6. Listing of the *ei0cmn.inc* Include File(Continued)

```
      common /EI0CON/  ssState(MaxDof,MaxSpE),
     1                 nssmpc(MaxSpE), issmpc(MaxDof,MaxMPC,MaxSpE)
c     ssState: State attributes for substructure nodes
c     nssmpc:  number of mpcs for alpha-nodes
c     issmpc:  dependent dofs for alpha-nodes
c
C=IF DOUBLE
      double precision
C=ELSE
      real
C=ENDIF
     4        eivals,ssvals,feimpc,fssmpc
c
      common /EI0EID/ eivals(MaxDoF),    ssvals(MaxDoF,MaxSpE),
     1                feimpc(MaxInd,MaxMPC),
     2                fssmpc(MaxInd,MaxMPC,MaxSpE)
c
c     eivals:  values (zero and nonzero) for pseudo-node constraints
c     ssvals:  values (zero and nonzero) for alpha-node constraints
c     feimpc:  values of coefficients for pseudo-node mpcs
c     fssmpc:  values of coefficients for alpha-node mpcs
c
      character*6  ceimpc, cssmpc
c
      common /EI0EIC/ ceimpc(MaxInd,MaxMPC),
     1                cssmpc(MaxInd,MaxMPC,MaxSpE)
c
c     ceimpc:  names of independent dofs for pseudo-node mpcs
c     cssmpc:  names of independent dofs for alpha-node mpcs
c
      integer eiState, neimpc, ieimpc
c
      common /EI0EII/ eiState(MaxDoF), neimpc, ieimpc(MaxDoF,MaxMPC)
c
c     eiState:  Constraint flags for pseudo-nodes
c     neimpc:   Number of MPC's defined for pseudo-nodes.
c     ieimpc:   Dependent freedoms for pseudo-node mpcs
c_end ei0cmn.inc
```

9.3.2.3 The *ei0com.inc* Include File

The file ***ei0com.inc*** contains a common block of pointers for the data objects used by the EI processor (EI0CBC), a common block which stores the processor reset values and data used by the low level database routines (EI0CBI), and a common block which stores the data object names (EI0CBA). A complete listing of the include file is provided as Table 9.7. This include file is based on a file used by the ES processor, ***es0com.inc***.

Table 9.7. Listing of the *ei0com.inc* Include File

```
c_beg ei0com.inc
c     ------------------------------------------------------------------
      integer         ctls       , defs       , dofs       , nodes
      common /EI0CBC/ ctls(Mctls), defs(Mdefs), dofs(MaxDof,MaxNen),
     $                nodes(MaxNen)
c ctls:  array which contains flags which control the processor functions
c defs:  array containing element definition (e.g., defs(pdNEN) = the
c        number of nodes for the current element
c dofs:  array containing an active dof table for the current element
c nodes: array containing node numbers for the current element
c     ------------------------------------------------------------------
      integer         NumDO
      parameter     ( NumDO = 25 )
c
      integer         pCSM   , pDIS   , pEDT   , pEFT   , pEGT   , pEIT
      integer         pELT   , pFRC   , pMAS   , pNCT   , pNTT   , pROT
      integer         pSTF   , pSTE   , pSTS   , pSTN   , pGCP   , pTEM
      integer         pAUX   , pATT   , pNVT
      integer         pERT   , pNDT   , pEAT   , pNAT
c
      parameter     ( pCSM = 1, pDIS = 2, pEDT = 3, pEFT = 4, pEGT = 5 )
      parameter     ( pEIT = 6, pELT = 7, pFRC = 8, pMAS = 9, pNCT =10 )
      parameter     ( pNTT =11, pROT =12, pSTF =13, pSTE =14, pSTS =15 )
      parameter     ( pSTN =16, pGCP =17, pTEM =18, pAUX =19, pATT =20 )
      parameter     ( pERT =21, pNDT =22, pNVT =23, pEAT =24, pNAT =25 )
c NumDO: Number of different data object types
c pCSM:  Pointer to CSM data object
c pDIS:  Pointer to nodal displacement (NVT) data object
c pEDT:  Pointer to EDT (element definition) data object
c pEFT:  Pointer to EFT (element fabrication) data object
c pEGT:  Pointer to EGT (element geometry) data object
c pEIT:  Pointer to EIT (element interpolation) data object
c pELT:  Pointer to ELT (element line) data object
c pFRC:  Pointer to FRC nodal force (NVT) data object
c pMAS:  Pointer to nodal lumped mass (NVT) data object
c pNCT:  Pointer to NCT (nodal coordinate) data object
c pNTT:  Pointer to NTT (nodal transformation) data object
c pROT:  Pointer to nodal rotation (NAT) data object
c pSTF:  Pointer to ELT (element line) data object
c pSTE:  Pointer to FRC nodal force (NVT) data object
c pSTS:  Pointer to nodal lumped mass (NVT) data object
c pSTN:  Pointer to NCT (nodal coordinate) data object
c pGCP:  Pointer to NTT (nodal transformation) data object
c pAUX:  Pointer to auxiliary storage (EAT) data object
c pATT:  Pointer to additional attribute (EAT) data object
c pNVT:  Pointer to NVT (nodal vector) data object
c pERT:  Pointer to ERT (element refinement) data object
c pNDT:  Pointer to NDT (nodal definition) data object
```

Table 9.7. Listing of the *ei0com.inc* Include File

```
c pEAT:  Pointer to EAT (element attribute) data object
c pNAT:  Pointer to NAT (nodal attribute) data object
c     ------------------------------------------------------------------
      integer         DBlen , reserv, ldi0  , step0 , mesh0, ldset
      integer         id    , ldi   , step  , mesh  , buf  , coset
      integer         AUXtyp, AUXloc, StrAtt
c
      common /EI0CBI/ DBlen, reserv, ldi0, step0, mesh0, ldset, coset,
     $                id(NumDO), ldi(NumDO), step(NumDO), mesh(NumDO),
     $                buf(NumDO), AUXtyp, AUXloc, AUXcyc, StrAtt
c DBlen:   Pointer to EAT (element attribute) data object
c reserv:  Pointer to NAT (nodal attribute) data object
c ldi0:    user specified logical device index for output
c step0:   user specified step id for output
c mesh0:   user specified mesh id for output
c ldset:   user specified load set number
c coset:   user specified constraint set number
c id:      DB pointer identifier for each active, open data object
c ldi:     ldi identifier for each active, open data object
c step:    step identifier for each active, open data object
c mesh:    mesh identifier for each active, open data object
c buf:     buffer length for each active, open data object
c AUXtyp:  data type for element auxiliary storage object
c AUXloc:  location (e.g., qCent) for element auxiliary storage object
c StrAtt:  Type of data (stress, strain, strain energy) saved in EST
c     ------------------------------------------------------------------
      character*72    Name, NamDef
c
      common /EI0CBA/ Name(NumDO), NamDef(NumDO)
c Name:   Data object names
c NamDef: Default values for data object names
c     ------------------------------------------------------------------
c_end ei0com.inc
```

9.3.2.4 The *el0ptr.inc* Include File

The ***el0ptr.inc*** file contains pointers into two element definition arrays: **ctls** and **defs** (which are found in the ***el0cmn.inc*** file). The **ctls** array contains flags which define the individual element implementation scope (*e.g.*, **ctls(pcNLG)** indicates whether or not the element formulation is capable of handling geometric nonlinearity). The **defs** array contains integer data which define the individual element (*e.g.*, **defs(pdNen)** is set to the number of nodes per element). These arrays are passed from the shell to the element cover; the element developer must fill them (as in the example cover routines of Section 9.4). A complete listing of the include file is provided as Table 9.8. Note that this include file is based on the include file used by the **ES** processor, ***es0ptr.inc***. Some items have been deleted as unneeded in ***el0ptr.inc*** but the definitions of the items remaining are the same as the ES version of this file.

Table 9.8. Listing of the *el0ptr.inc* Include File

```
c_beg EI0PTR.INC
c
c Pointers for CTLS Array:
c     ------------------------------------------------------------c
      integer     pcCORO     , pcNLG      , pcNLM      , pcNLL
      parameter ( pcCORO =  1, pcNLG  = 2 , pcNLM  =  3, pcNLL  = 4 )
      integer     pcTEMP
      parameter ( pcTEMP = 22 )
      integer     pcLLliv    , pcSLliv    , pcPLliv    , pcBLliv
      parameter ( pcLLliv= 23, pcSLliv= 24, pcPLliv= 25, pcBLliv=26 )
      integer     pcPSTN     , pcPSTS
      parameter ( pcPSTN = 27, pcPSTS = 28 )
      integer     pcSTNo     , pcSTSo     , pcSTEo
      parameter ( pcSTNo = 29, pcSTSo = 30, pcSTEo = 31 )
      integer     pcLDS      , pcNORO
      parameter ( pcLDS  = 32, pcNORO = 33)
      integer     mCTLS
      parameter ( mCTLS  = 33 )
c     ------------------------------------------------------------
c Pointers for DEFS Array:
c     ------------------------------------------------------------
      integer     pdOPT      , pdNEN      , pdCLAS     , pdNIP
      parameter ( pdOPT  = 1, pdNEN  = 2, pdCLAS = 3 , pdNIP  = 4 )
      integer     pdNDOF     , pdC        , pdNSTR     , pdSTOR
      parameter ( pdNDOF = 5, pdC    = 6, pdNSTR = 7 , pdSTOR = 8 )
      integer     pdDIM      , pdCNS      , pdNEE      , pdSHAP
      parameter ( pdDIM  = 9, pdCNS  =10, pdNEE  =11 , pdSHAP =12 )
      integer     pdTWIS     , pdPARS     , pdCENT     , pdNORO
      parameter ( pdTWIS =13, pdPARS =14, pdCENT =15 , pdNORO =16 )
      integer     pdESPD     , pdTGE      , pdPROJ
      parameter ( pdESPD =17, pdTGE  =18, pdPROJ =19 )
      integer     pdNLE      , pdNSE      , pdNNLT     , pdNNST
      parameter ( pdNLE  =20, pdNSE  =21, pdNNLT =22 , pdNNST =23 )
      integer     pdP        , pdCLASS    , pdSHAPE
      parameter ( pdP    =32, pdCLASS=33, pdSHAPE=34 )
      integer     mDEFS
      parameter ( mDEFS  =34 )
c     ------------------------------------------------------------
c Legitimate Values of DEFS(pdCLAS):
c     ------------------------------------------------------------
      integer     idBEAM     , idSHEL     , idSOLI
      parameter ( idBEAM = 1, idSHEL = 2, idSOLI = 3 )
c     ------------------------------------------------------------
c Legitimate Values of DEFS(pdSHAP):
c     ------------------------------------------------------------
      integer     idQUAD     , idTRIA
      parameter ( idQUAD = 1, idTRIA = 2 )
c     ------------------------------------------------------------
c_end EI0PTR.INC
```

9.3.3. Shell Subroutines

The ***el0_shell.ams*** file is composed of a number of subroutines which each manage a different function (*e.g.*, subroutine ***El0res*** handles the processor resets). Table 9.9 provides a summary of the functions performed by each subroutine. The command class (see Section 7.2.4 for a description of the options) for which the subroutine is active is also listed.

Table 9.9. Summary of El_shell Subroutines

Subroutine Name	File name	Function	Command Class
El0	***el0.msc***	Main driver routine	All
El0beg	***el0beg.msc***	Initialize the processor	All
El0chks	***el0chks.msc***	Check available processor space	All
El0cmd	***el0cmd.msc***	Process user input commands	All
El0crf	***el0crf.msc***	Forms normal and tangent vectors in edge and interface reference frames	DEFINE
El0csm	***el0csm.msc***	Save the necessary data in the CSM data object.	DEFINE
El0def	***el0def.msc***	Process the DEFINE command class	DEFINE
El0defe	***el0defe.msc***	Process the DEFINE ELEMENTS command	DEFINE
El0deff	***el0deff.msc***	Process the DEFINE FREEDOMS command	DEFINE
El0defs	***el0defs.msc***	Defines the path coordinates for the interface element; constructs a path based on user input data	DEFINE
El0edef	***el0edef.msc***	Determines the finite element types of the incoming substructures	DEFINE
El0elt	***el0elt.msc***	Saves element data in the appropriate element data objects.	DEFINE
El0end	***el0end.msc***	Close the data objects/database before exiting	All
El0find	***el0find.msc***	Find the finite elements connected to the nodes of each substructure	DEFINE
El0frm	***el0frm.msc***	Process the FORM command class	FORM
El0log	***el0log.msc***	Set logical flags for execution control	All
El0mtx	***el0mtx.msc***	Process the element matrix generation	FORM
El0nod	***el0nod.msc***	Saves nodal data in the appropriate nodal data objects.	DEFINE
El0res	***el0res.msc***	Process RESET command class	All
El0set	***el0set.msc***	Process individual RESET commands	All
El0tran	***el0tran.msc***	Form transformation matrices based on normal and tangents	FORM

Each command class has its own execution flow through the processor. The three currently available command classes and their individual execution flows are shown in Figures 9.1-9.3. Note that in these Figures, subroutines listed as "Auxiliary Subroutines" are used to process user input, place and retrieve data to and from the database, set up arrays, define path variables, and perform other such auxiliary functions. Subroutines labeled as CSM*, NCT*, NTT*, etc. are HDB utility routines. The reader is referred to Ref. 9.3-2 for more information about these subroutines.

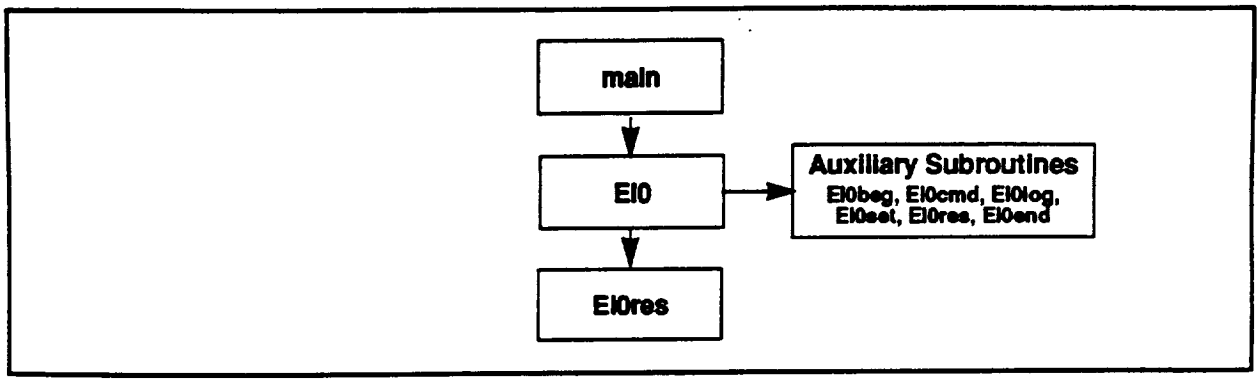


Figure 9.1. Execution Flow for the RESET Command Class

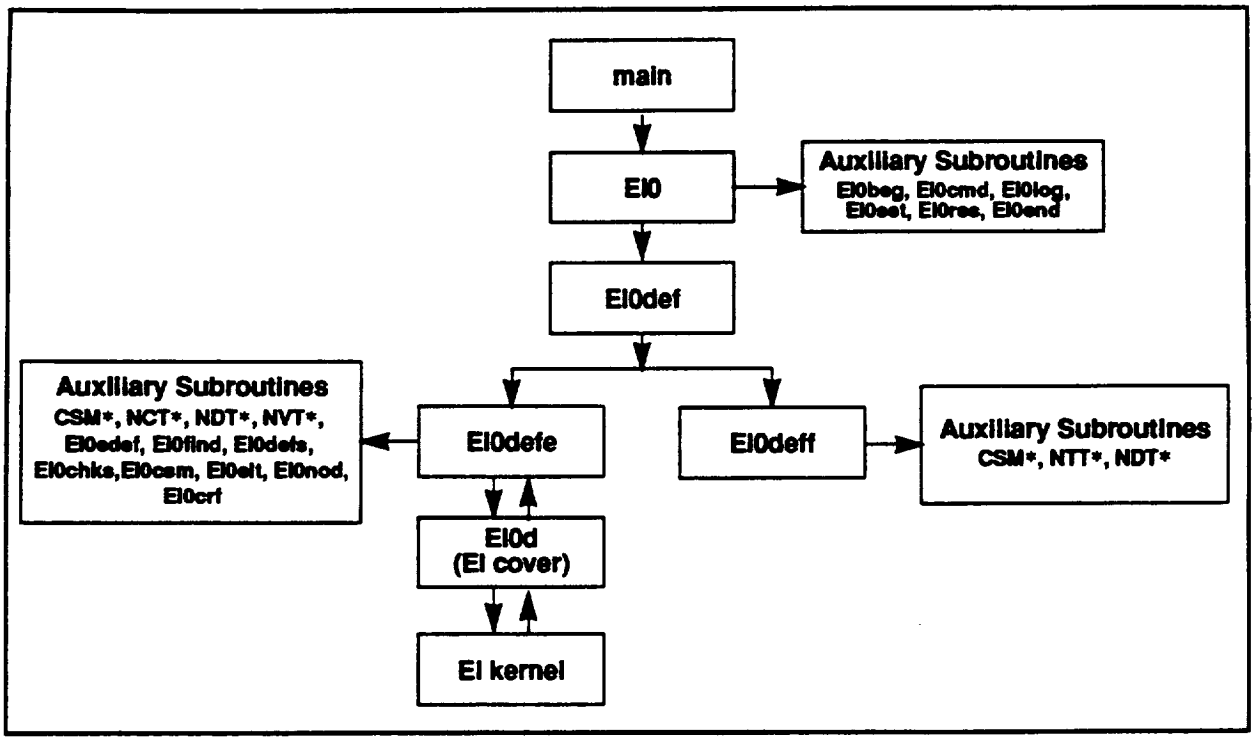


Figure 9.2. Execution Flow for the DEFINE Command Class

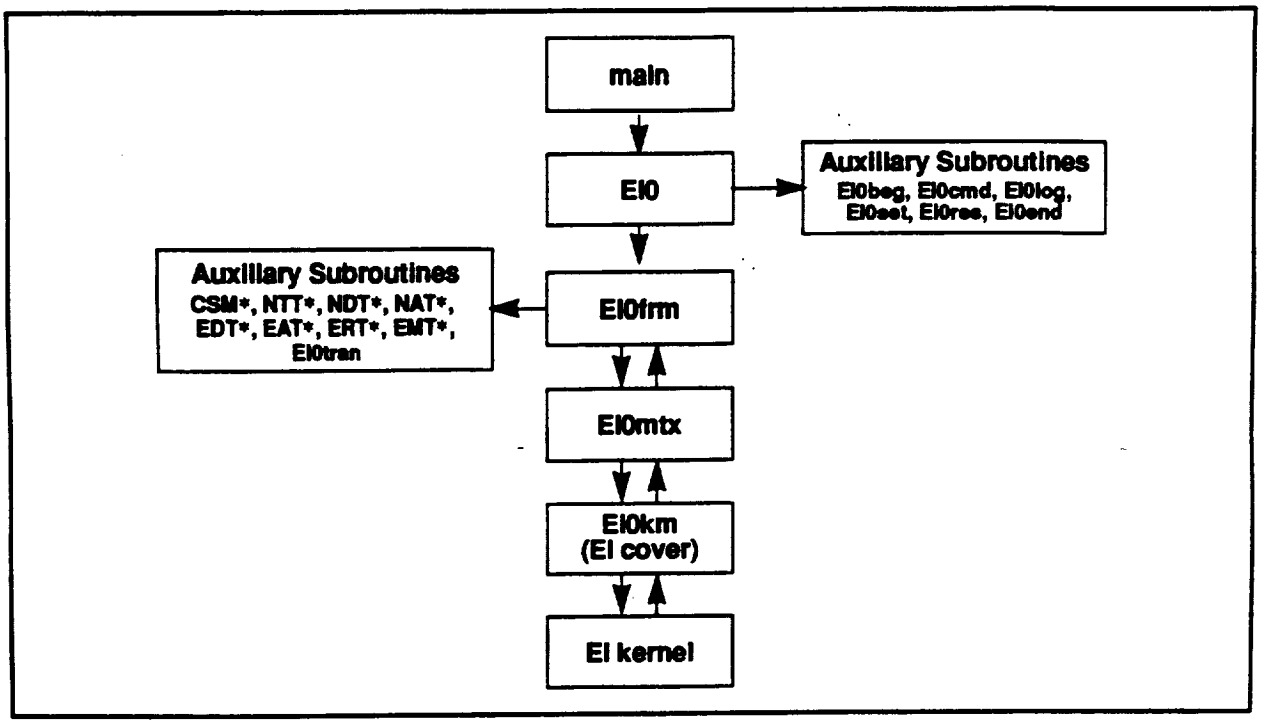


Figure 9.3. Execution Flow for the FORM Command Class

In the following subsections, additional information about each of the subroutines listed in Table 9.9 is provided. The subroutines are described in alphabetical order. The reader is referred to the previous Figures to visualize the actual order of execution. For descriptions of the subroutines listed as utilities, the reader is referred to Refs. 9.3-1 through 9.3-4. Additional subroutines which may be thought of as utilities are located in the file ***el0util.ams***. These utility subroutines are generally called by the ***El0**** subroutines listed in Figures 9.1-9.3 as Auxiliary Subroutines.

9.3.3.1 Subroutine *El0*

Subroutine ***El0*** is the primary driver routine for the processor. It processes the user input and directs the processor execution. Subroutine ***El0*** uses the include files and calls the subroutines listed in Table 9.10.

Table 9.10. Include Files and Subroutines Used by Subroutine *El0*

Include Files	Subordinate Subroutines	Utility Subroutines
el0com.inc *el0fig.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*El0beg, El0cmd, El0log, El0set, El0res, El0def, El0frm, el0end*	*CmdPro, Err, CMATCH*

9.3.3.2 Subroutine *El0beg*

Subroutine ***El0beg*** initializes the EI processor and the processor resets (*i.e.*, the variables *ldi*, *step*, *mesh*, *zero*, *ldset*, *ldfac*). It also sets default dataset names.

Table 9.11. Include Files and Subroutines Used by Subroutine *El0beg*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0fig.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*None*	*BegPro, GSCLRI, UpCase*

9.3.3.3 Subroutine *El0chks*

Subroutine ***El0chks*** verifies that the size limits for the EI processor are not violated. These size limits include limits on the number of nodes per element (currently set to 300) and the number of degrees of freedom per node (currently set to 6). The include files and subroutines used by ***El0chks*** are listed in Table 9.12.

Table 9.12. Include Files and Subroutines Used by Subroutine *El0chks*

Include Files	Subordinate Subroutines	Utility Subroutines
el0lim.inc *el0ptr.inc* *qsymbol.inc*	*None*	*ERR*

9.3.3.4 Subroutine *El0cmd*

Subroutine ***El0cmd*** parses the command input line for the EI processor. The include files and subroutines used by ***El0cmd*** are listed in Table 9.13

Table 9.13. Include Files and Subroutines Used by Subroutine *El0cmd*

Include Files	Subordinate Subroutines	Utility Subroutines
None	*None*	*CCLVAL, CLOADQ*

9.3.3.5 Subroutine *El0crf*

Subroutine ***El0crf*** forms computational reference frames (in the form of normals, path and surface tangents) along the interface.The include files and subroutines used by ***El0crf*** are listed in Table 9.14.

Table 9.14. Include Files and Subroutines Used by Subroutine *El0crf*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0lim.inc* *qsymbol.inc*	*None*	*GSCROS, GSNORM, GSDOT, ERR*

9.3.3.6 Subroutine *El0csm*

Subroutine ***El0csm*** saves the interface element general summary and element summary attributes in the CSM data object. The include files and subroutines used by ***El0csm*** are listed in Table 9.15.

Table 9.15. Include Files and Subroutines Used by Subroutine *El0csm*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*None*	*CSM*, ERR*

9.3.3.7 Subroutine *El0def*

Subroutine ***El0def*** processes the DEFINE command class. This class is currently limited to the DEFINE ELEMENTS and DEFINE FREEDOMS commands. The include files and subroutines used by ***El0def*** are listed in Table 9.16.

Table 9.16. Include Files and Subroutines Used by Subroutine El0def

Include Files	Subordinate Subroutines	Utility Subroutines
el0com.inc *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*El0defe, El0deff*	*ERR*

9.3.3.8 Subroutine *El0defe*

Subroutine ***El0defe*** processes all of the subcommands and qualifiers associated with the DEFINE ELEMENTS command. It also performs or directs all database interaction required for this command. The include files and subroutines used by ***El0defe*** are listed in Table 9.17.

Table 9.17. Include Files and Subroutines Used by Subroutine El0defe

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*El0edef, El0D, El0defs, El0chks, El0csm, El0elt, El0nod*	*CLread, Iclear, Rclear, CLVALI, CLVALf, GMCODn, ERR, CSM*, NCT*, NDT*, NVT*, CLput, CI2CL*

9.3.3.9 Subroutine *El0deff*

Subroutine ***El0deff*** processes the DEFINE ELEMENTS command. It also performs or directs all the database interaction required for this command. The include files and subroutines used by ***El0deff*** are listed in Table 9.18.

Table 9.18. Include Files and Subroutines Used by Subroutine *El0deff*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*None*	*GMCODn, CSM*, NTT*, NDT*, ERR*

9.3.3.10 Subroutine *El0defs*

Subroutine ***El0defs*** defines the interface element path variables and coordinates. The include files and subroutines used by ***El0defs*** are listed in Table 9.19.

Table 9.19. Include Files and Subroutines Used by Subroutine *El0defs*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0lim.inc* *qsymbol.inc*	*EI utilities*	*None*

9.3.3.11 Subroutine *El0edef*

Subroutine ***El0edef*** is a utility subroutine which determines the finite element types along the interface. ***El0edef*** creates a table of adjacency information which lists the element types connected to each interface node. The include files and subroutines used by ***El0edef*** are listed in Table 9.20.

Table 9.20. Include Files and Subroutines Used by Subroutine *El0edef*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0lim.inc* *qsymbol.inc*	*El utilities*	*None*

9.3.3.12 Subroutine *El0elt*

Subroutine ***El0elt*** saves the element data in element data objects. This subroutine is only called during the DEFINE ELEMENTS command execution. The include files and subroutines used by ***El0elt*** are listed in Table 9.21.

Table 9.21. Include Files and Subroutines Used by Subroutine *El0elt*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*None*	*GMCODn, CSM*, EDT*, EAT**

9.3.3.13 Subroutine *El0end*

Subroutine ***El0end*** ends processing. The include files and subroutines used by ***El0end*** are listed in Table 9.22.

Table 9.22. Include Files and Subroutines Used by Subroutine *El0end*

Include Files	Subordinate Subroutines	Utility Subroutines
None	*None*	*EndPro*

9.3.3.14 Subroutine *El0find*

Subroutine ***El0find*** is a utility subroutine which creates a list of nodes and their coordinates for each substructure. This list is used in the definition of the interface element path. ***El0find*** processes any filters on the coordinates and/or node numbers which may have been specified by the user. Only the nodes which pass through the various coordinate and nodal filters are listed. The include files and subroutines used by ***El0find*** are listed in Table 9.23.

Table 9.23. Include Files and Subroutines Used by Subroutine *El0find*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*None*	*GMCODn, NDT*, NCT*, ERR*

9.3.3.15 Subroutine *El0frm*

Subroutine ***El0frm*** is the driver routine for the element matrix formation. ***El0frm*** both reads and writes the required data from and to the database. The include files and subroutines used by ***El0frm*** are listed in Table 9.24.

Table 9.24. Include Files and Subroutines Used by Subroutine *El0frm*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*El0mtx*	*GMCODn, CSM*, EDT*, ERT*, EMT*, EAT*, NDT*, NTT*, NAT*, ERR*

9.3.3.16 Subroutine *El0log*

Subroutine ***El0log*** is a utility which constructs the logical command flags from the ***El*** command input line. The include files and subroutines used by ***El0log*** are listed in Table 9.25.

Table 9.25. Include Files and Subroutines Used by Subroutine *El0log*

Include Files	Subordinate Subroutines	Utility Subroutines
el0fig.inc *el0ptr.inc* *qsymbol.inc*	*None*	*ERR, GSCRI*

9.3.3.17 Subroutine *El0mtx*

Subroutine ***El0mtx*** forms the appropriate element matrix. Currently the implementation is limited to formation of the material stiffness matrix for linear elastic materials. The include files and subroutines used by ***El0mtx*** are listed in Table 9.26.

Table 9.26. Include Files and Subroutines Used by Subroutine *El0mtx*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*El0KM (El cover routine)*	*GSCLRv, MID2*

9.3.3.18 Subroutine *El0nod*

Subroutine ***El0nod*** saves the nodal data in the database. ***El0nod*** is called only during the execution of the DEFINE ELEMENTS command. The include files and subroutines used by ***El0nod*** are listed in Table 9.27.

Table 9.27. Include Files and Subroutines Used by Subroutine *El0nod*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*None*	*GMCODn, CSM*, NDT*, NCT*, NAT*, ERR*

9.3.3.19 Subroutine *El0res*

Subroutine ***El0res*** processes the RESET commands. The include files and subroutines used by ***El0res*** are listed in Table 9.28.

Table 9.28. Include Files and Subroutines Used by Subroutine *El0res*

Include Files	Subordinate Subroutines	Utility Subroutines
el0com.inc *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*None*	*CMATCH, fCLVAL, ICLVAL, cCLVAL*

9.3.3.20 Subroutine *El0set*

Subroutine ***El0set*** initializes the logical device indices and the data object names if RESET is *NOT* used to perform these tasks. The include files and subroutines used by ***El0set*** are listed in Table 9.29.

Table 9.29. Include Files and Subroutines Used by Subroutine *El0set*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0com.inc* *el0lim.inc* *el0ptr.inc* *qsymbol.inc*	*None*	*GMBUDn, GMCODn*

9.3.3.21 Subroutine *El0tran*

Subroutine ***El0tran*** forms the transformation matrices from the edge and interface to global reference frames using the normals and tangents created and saved during the DEFINE ELEMENTS operation. The include files and subroutines used by ***El0tran*** are listed in Table 9.30.

Table 9.30. Include Files and Subroutines Used by Subroutine *El0tran*

Include Files	Subordinate Subroutines	Utility Subroutines
el0cmn.inc *el0lim.inc* *qsymbol.inc*	*None*	*None*

9.3.4. References

9.3-1 Felippa, C. A., *The Computational Structural Mechanics Testbed Architecture: Volume III - The Interface*, NASA CR 178386, December 1988.

9.3-2 Stanley, G. M. and Swenson, L., *HDB Object-Oriented Database Utilities for COMET-AR*, NASA CSM Contract Report, August, 1992.

9.3-3 Felippa, C. A., *The Computational Structural Mechanics Testbed Architecture: Volume IV - The Global-Database Manager GAL-DBM*, NASA CR 178387, January 1989.

9.3-4 Stanley, G.M., Hurlbut, B., Levit, I., Stehlin, B., Loden, W., and Swenson, L., *COMET-AR User's Manual*, LMSC Report #P032583, 1993.

THIS PAGE INTENTIONALLY BLANK

9.4. The Generic Interface Element Processor Cover

9.4.1. General Description

The interface element cover is a single file, called ***ei*_cover.ams***, which contains several subroutines each of which the developer of new interface elements must customize. Each interface element developer must create a new file, replacing the * in the file name with a processor number (*e.g.*, ***ei1_cover.ams***, ***ei20_cover.ams***). The subroutines in the cover act as translators between the generic shell part of the processor and the developer-written kernels. The calls from the shell to the cover routines are standard. The developer must fill in calls to the appropriate kernel routines using the data passed through to the cover subroutines from the shell. If sufficient data has not been passed down to a specific cover routine, the developer should first look to the include files listed in the previous section. If incorporating the use of one or more include files still does not provide all of the required information, a revision to the basic assumptions for the shell may be required and the developer should contact COMET-AR maintenance personnel.

In the following section, a summary of the currently required cover subroutines is listed. The final section contains an example of each of these cover routines.

9.4.2. Required Subroutines

The interface element implementation is currently limited to linear, static, elastic analysis. Therefore, the currently active cover subroutines are those which supply the functionality which falls within these limitations. Table 9.31 provides a summary of the active subroutines. As new capabilities are added, new cover routines will be added.

Table 9.31. Summary of *ei_cover.ams* Subroutines

Subroutine Name	Function
EI0D	Called during the DEFINE ELEMENTS command. This subroutine must set up the **defs** and **ctls** arrays and define the element and processor names.
EI0KM	Called during the FORM STIFFNESS/MATL command. This subroutine must pass the element stiffness matrix (in the proper computational frames) for the current interface element back to the shell.

9.4.3. An *ei*_cover.ams* Example

The example given in this section provides cover routines ***EI0D*** and ***EI0KM*** for the current version of the EI1 processor. Each subroutine is listed and annotated. A developer of new interface elements need only copy the ***ei_cover.ams*** template file (which contains both subroutines) into a file named ***ei*_cover.ams*** (*e.g.*, ***ei3_cover.ams***) and make changes as needed for the specific element implementation.

9.4.3.1 The *EI0D* Subroutine

This subroutine initializes element definition variables (including the element and processor names) and calls the element kernel to DEFINE ELEMENTS. The current version of EI1 permits either user or automatic definition of the number of pseudo-nodes along the interface element. It also requires the definition of alpha-nodes, which are defined by the element kernel (no user selection of alpha-nodes is permitted). Table 9.32 is an annotated listing of the *EI0D* subroutine currently used in the EI1 processor. Note that the calling sequence for subroutine *EI0D*, the argument type declarations, and usually the include files will be the same (namely, the one listed in the Table) for every EI processor.

Table 9.32. Listing of the EI1 Processor *EI0D* Subroutine for the EI1 Processor

Arguments listed here are defined as per the include files in previous sections.

Include selected common blocks and parameters

Type the input and output arguments

Type the internal variables

```
C=DECK EI0D
C=PURPOSE    Element Definition Cover Routine for Interface Elements.
C=BLOCK FORTRAN
      subroutine EI0D ( EltNam, EltPro, EltTyp, EltNum,  defs,
     1                  ssNdofn,ssDofn, nSS,    ssnode,
     2                  ssnn,   ssdofs,
     3                  sstelt, ssnelt, nAlpha,  nAlphaT,
     4                  aNodes, ieDofn, ieNdofn,ieNen,   nPn,
     5                  nape,   ssfid,  status )
c ------------------------------------------------------------------
c                       I n c l u d e   F i l e s
c ------------------------------------------------------------------
      include 'ei0lim.inc'
      include 'ei0ptr.inc'
      include 'qsymbol.inc'
c ------------------------------------------------------------------
c              A r g u m e n t   D e c l a r a t i o n s
c ------------------------------------------------------------------
      character*(*) EltNam                  ! Element Name
      character*(*) EltTyp                  ! Element Type
      character*(*) EltPro                  ! Element Processor
      integer       EltNum                  ! Element Number
      integer       defs(*)
      integer       nSS                     ! Number of SS
      integer       ssNdofn(MaxSpE)         ! Number of dofs/node
      integer       ssDofn(maxDoF,MaxSpE)   ! Active dofs/SS
      integer       ssnode(MaxNpS,MaxSpE)   ! Interface nodes
      integer       ssnn(MaxSpE)            ! Number of i-nodes
      integer       ssdofs(MaxDOF,MaxNpS,MaxSpE) ! dofs at each i-node
      integer       sstelt(MaxTyp,MaxNpS,MaxSpE) ! Element types
      integer       ssnelt(MaxNpS,MaxSpE)   ! Number of f.e. types
      integer       nAlpha(MaxSpE)          ! Number of alpha dofs
      integer       nAlphaT                 ! Total number of alphas
      integer       aNodes(MaxSpE)
      integer       nPn                     ! Number of pseudo-nodes
      integer       nApE                    ! number of alfas/f.e.
      integer       ssfid                   ! fine substructure id
      integer       status                  ! return status
c ------------------------------------------------------------------
c            I n t e r n a l   D e c l a r a t i o n s
c ------------------------------------------------------------------
      character*4 CEltNum         ! Character element number
      integer     ieNen           ! number of interface element nodes
      integer     ieNdofn         ! number of dofs/node for the i.e.
      integer     ieDofn(MaxDOF)  ! int. elt. active dofs
c ------------------------------------------------------------------
c                       L O G I C
c ------------------------------------------------------------------
```

Table 9.32. Listing of the El1 Processor *EI0D* Subroutine for the El1 Processor(Continued)

Check status

Define the element name for this EI processor

Call Developer written kernel routine with necessary arguments

Set the element definition parameters used by this element type

Return to the shell

```
      if (status .ne. qOK) return
c     -----------------------
c     Define the Element Name:
c     -----------------------
      call CI2CL (EltNum,CEltNum,3,len)
      EltPro  =  'EI1'
      EltTyp  =  'HYBV'
      EltNam  =   EltPro(1:3)//'_'//EltTyp(1:4)//'_'//CEltNum(1:len)
c     ------------------------------------------
c     Define the number of alpha's and pseudo-nodes:
c     ------------------------------------------
      call HYBDEF ( ssNdofn,ssDofn, nSS,    ssnode,
     2              ssnn,   ssdofs,
     3              sstelt, ssnelt, nAlpha, nAlphaT,
     4              aNodes, ieNen,  ieNdofn,ieDofn, nPn, nApE, ssfid,
     5              status )
c     ---------------------------------------
c     Set element parameters in the DEFS array:
c     ---------------------------------------
      do 100 i = 1, Mdefs
         defs(i) = 0
  100 continue
c
      defs(pdNEN)  = ieNen
      defs(pdCLAS) = qBeam
      defs(pdNIP)  = 1
      defs(pdNDOF) = ieNdofn
      defs(pdP)    = 1
      defs(pdDIM)  = 1
      defs(pdNEE)  = ieNen*ieNdofn
      defs(pdSHAP) = qLine
      defs(pdNORO) = qTrue
      defs(pdNLE)  = 1
      defs(pdNSE)  = 1
      defs(pdNNLT) = ieNen
      defs(pdNNST) = 0
      defs(pdCLASS)= qBeam
      defs(pdSHAPE)= qLine
c
      return
      end
C=END FORTRAN
```

9.4.3.2 The *EI0KM* Subroutine

This subroutine drives the formation of the element material stiffness matrix for each interface element. It is invoked by the FORM STIFFNESS/MATL command. Unlike ***EI0D*** which both calls the kernel routine for element definition and sets array values, ***EI0KM*** acts solely as a translator between the data passed in by the shell and the data required by the kernel routine. A new element developer therefore must only replace the call to the kernel routine ***HYBFRM*** with a call to the appropriate new kernel routine; all else about ***EI0KM*** will remain the same for all interface element types. Table 9.33 is an annotated listing of the ***EI0KM*** subroutine currently used in the EI1 processor. Note that the calling sequence for subroutine ***EI0KM***, the argument type declarations, and usually the include files will be the same (namely, the one listed in the Table) for every EI processor.

Table 9.33. Listing of the EI1 Processor *EI0KM* Subroutine

Arguments listed here defined previously.

Include common blocks and parameters

Type the input and output arguments

Type the internal variables

Check status

Call Developer written kernel routine with necessary arguments

Return to the shell

```
C=DECK EI0KM
C=PURPOSE  Generic Material Stiffness Routine for Interface Elements
C=BLOCK FORTRAN
      subroutine EI0KM (defs,   ieDofN,  nSS,    ieEtyp,  pathss,
     1                  pathei, dSpline, nAlpha, nAlphaT, aNodes,
     2                  nPn,    Matrix,  scale,  nApE,    ssnn,
     3                  ssTdg,  ssTgc,   eiTdg,  eiTgs,   status)
c ------------------------------------------------------------------
c                       I n c l u d e   F i l e s
c ------------------------------------------------------------------
      include 'ei0lim.inc'
      include 'ei0ptr.inc'
      include 'qsymbol.inc'
c ------------------------------------------------------------------
c              A r g u m e n t   D e c l a r a t i o n s
c ------------------------------------------------------------------
      integer  defs(*)
      integer  ieDofn(MaxDOF)
      integer  nSS                      ! Number of substructures
      integer  ieEtyp(MaxTYp,MaxNEN)
      integer  nAlpha(nss)              ! Number of alpha dofs
      integer  nAlphaT                  ! Total number of alphas
      integer  aNodes(MaxSpE)
      integer  nPn                      ! Total number of pseudo-nodes
      integer  nApE(MaxSpE)
      integer  ssnn(MaxSpE)
      integer  status                   ! return status  <I/O>
C=BLOCK DOUBLE
      double precision
C=ELSE
      real
C=END DOUBLE
     2         scale,                  ! element scale factor
     3         pathss(MaxNps,MaxSpE),
     4         pathei(MaxPpE),
     5         ssTdg(3,3,MaxNpS,MaxSpE),
     6         ssTgc(3,3,MaxNpS,MaxSpE),
     7         eiTdg(3,3,MaxPpE,MaxSpE),
     8         eiTgs(3,3,MaxPpE),
     9         Matrix(MaxNEE,MaxNEE)  ! Element matrix
c ------------------------------------------------------------------
c            I n t e r n a l   D e c l a r a t i o n s
c ------------------------------------------------------------------
      character*4 CEltNum     ! Character  element number
      integer  ieNen          ! number of i.e. nodes (total)
      integer  ieNdofn        ! number of dofs per node for the i.e.
c ------------------------------------------------------------------
c                        L  O  G  I  C
c ------------------------------------------------------------------
      if (status .ne. qOK) return
      ieNdofn = defs(pdNDOF)
      ieNEN   = defs(pdNEN)
      call HYBFRM ( nSS,    ieEtyp, nAlpha, nAlphaT,aNodes,
     2              pathss, pathei, ieDofn, ssnn,   ieNen,  ieNdofn,
     3              nPn,    dSpline,Matrix, scale,  nApE,
     4              ssTdg,  ssTgc,  eiTdg,  eiTgs,  status )
      return
      end
C=END FORTRAN
```

9.4.4. References

None.

9.5. *makefile* Example

The creation of an executable interface element processor is a two step process. The first step must be taken by COMET-AR maintenance personnel and is the creation of object files of the shell subroutines. Should problems arise when linking with the shell subroutines or when using shell parameters, the interface element developer should seek assistance from COMET-AR maintenance personnel. The second step in the creation of an executable interface element processor is the creation of the actual EI processor executable. Table 9.34 contains a listing of the makefile used to create the EI1 processor exectuable (which can be found in ***$AR_EIPRC/makefile.eip***).

Table 9.34. EI1 Processor *makefile* Example

	`.IGNORE:`
	`.SUFFIXES:`
	`.SUFFIXES: .o .ams`
	`# Set default name for processor here`
Processor Name	`ei=ei1`
	`FC = fc`
Set Fortran Flags and Max keys	`FFLAGS = -c -O2 -72 -p8`
	`MAXKEYS = NICE SINGLE CONVEX MALLOC EXTP`
	`.ams.o:`
	`rm -f $*.tmp`
	`rm -f $*.f`
Compilation rules	`include -i $*.ams -o $*.tmp -d $(EI0INC)`
	`max /wc/for/sic/ti/mach=unix -i $*.tmp -o $*.f $(MAXKEYS) $(EI0KEY)`
	`- rm $*.tmp`
	`$(FC) -c $(FFLAGS) $*.f >$*.lis 2>&1`
	`EI0 = /usr/u5/gimmp/ar/mods/ei`
	`EI0INC = /usr/u5/gimmp/ar/mods/inc`
	`CSM = /csm`
	`AR = /usr/u2/newlock/ar`
	`AR_LIB = $(AR)/lib`
	`AR_INC = $(EI0)`
	`UTL = $(CSM)/sam/utl`
	`PRO_LIB = $(AR_LIB)/prolib.a`
	`HDB_LIB = $(AR_LIB)/hdblib.a`
	`DB_LIB = $(AR_LIB)/dblib.a`
Define Library Objects to be used in the Link	`UTLS_LIB = $(AR_LIB)/sutl.a`
	`ARUTL_LIB = $(AR_LIB)/arutl.a`
	`GEN_LIB = $(AR_LIB)/genlib.a`
	`LIBOBJS = $(AR_LIB)/gsutil.a $(AR_LIB)/crutil.a`
	`NICELIBS = $(AR_LIB)/clp86lb.a \`
	`$(AR_LIB)/gal86lb.a \`
	`$(AR_LIB)/dmg86lb.a \`
	`$(AR_LIB)/utl86lb.a \`
	`$(AR_LIB)/bio86lb.a`
	`LIBS = $(PRO_LIB) $(UTLS_LIB) $(ARUTL_LIB) \`
	`$(HDB_LIB) $(DB_LIB) $(GEN_LIB) $(LIBOBJS) $(NICELIBS)`
	`EI_OBJS = $(EI0)/ei_shell.a $(ei)_cover.o $(ei)_kernel.o`
	`#`
Create an executable file	`$(ei): $(EI_OBJS) $(LIBS)`
	`$(FC) $(LFLAGS) -o $(ei) $(EI0)/main.o $(EI_OBJS) $(LIBS)`
	`cp *.f ..`
Dependencies	`$(ei)_cover.o : $(ei)_cover.ams hyblim.inc`
	`$(ei)_kernel.o : $(ei)_kernel.ams hyblim.inc`

THIS PAGE INTENTIONALLY BLANK

Part VI.
DATA OBJECTS

30, 31, 32

THIS PAGE INTENTIONALLY BLANK

10. New Data Objects

10.1. Overview

The implementation of the interface element required the creation of several new nodal and element attribute tables. This Chapter describes these new objects and is outlined as follows:

Table 10.1. Outline of Chapter 10: New Data Objects

Section	Subject	Function
2	New Nodal Objects	Provide details on the new nodal data objects (NATs)
3	New Element Objects	Provide details on the new element data objects (EATs)

THIS PAGE INTENTIONALLY BLANK

10.2. New Nodal Data Objects

10.2.1. General Description

The interface element implementation required several new nodal attribute tables. These new nodal data objects are summarized in Table 10.2 and described in subsequent sections.

Table 10.2. Summary of New Nodal Attribute Tables (NATs)

Object Name	Purpose	Creator
NODAL.IEID...*mesh*	Identifies the interface elements to which the pseudo-nodes and alpha-nodes are attached.	EI
NODAL.NIDS...*mesh*	Identifies the original node number of the master model nodes.	MSTR
NODAL.TANGENTS...*mesh*	Stores path tangents for the pseudo-nodes.	EI
NODAL.TYPE...*mesh*	Identifies the node type (pseudo-node, alpha-node, or finite element node).	EI (MSTR modifies)

In the following discussion, the term "I-node" denotes, collectively, the pseudo-nodes and alpha-nodes.

10.2.2. Nodal Attribute Table (NAT): NODAL.IEID...*mesh*.

The **NODAL.IEID...*mesh*** table contains a single integer for each of the I-nodes. The object is created in the **EI** processor and is composed of the element identification number of the interface element to which each I-node is attached. The object format is:

NAT: NODAL.IEID..mesh			
Attribute	I-node 1	. . .	I-node *n*
NodAtt	$\mathbf{IntElt}_1$	. . .	$\mathbf{IntElt}_n$

where **IntElt** is the element identifier of the interface element to which I-node i is attached. Note that each I-node is, upon creation, attached to only one interface element; this interface element number is listed as **IntElt**. This data object exists solely in the library containing all and only interface elements.

10.2.3. Nodal Attribute Table (NAT): NODAL.NIDS...*mesh*.

The **NODAL.NIDS...*mesh*** table contains a single number for each of the finite element nodes and I-nodes. This object is created by the **MSTR** processor during the node renumbering process and contains the original node number of each node in the master model. The object format is:

NAT: NODAL.NIDS...mesh			
Attribute	fe_Node 1...*nfe*	p_Node 1...*npn*	a_Node 1...*nan*
NodAtt	Nid_1 ... Nid_{nfe}	Nid_1 ... Nid_{npn}	Nid_1 ... Nid_{nan}

where Nid is the original node number, *nfe* is the number of finite element nodes, *npn* is the number of pseudo-nodes, and *nan* is the number alpha-nodes.

10.2.4. Nodal Attribute Table (NAT): NODAL.TANGENTS...*mesh*.

The **NODAL.TANGENTS...*mesh*** table contains a vector for each of the I-nodes. This object is created by the EI processor during the element definition and contains the path tangent for each pseudo-node and zeroes for each alpha-node. The object format is:

NAT: NODAL.TANGENTS..mesh			
Attribute	I-node 1	. . .	I-node *n*
NodAtt(3)	$tangent_1$	. . .	$tangent_n$

where **tangent** is the tangent vector along the interface element path for the pseudo-nodes and zeroes for the alpha-nodes. This data object exists solely in the library containing all and only interface elements.

10.2.5. Nodal Attribute Table (NAT): NODAL.TYPE...*mesh*

The **NODAL.TYPE...*mesh*** table is created in the EI processor and recreated by the **MSTR** processor. The version of the object used by the EI processor contains flags for each of the I-nodes. All of the interface elements are stored in a single library, so there will be one **NODAL.TYPE...*mesh*** object containing all of these new I-nodes. Pseudo-nodes are listed first for each element, alpha-nodes are listed second for each element. The object format is:

NAT: NODAL.TYPE...*mesh*			
Attribute	I-node 1	. . .	I-node *n*
NodAtt	$Type_1$	. . .	$Type_n$

where **Type** will be set to a value of **qD** or **qAlpha** corresponding to displacement and traction type nodes (pseudo- and alpha-nodes) respectively. The user may define the number of displacement type nodes although it is recommended that automatic definition be incorporated by the developer whenever possible.

The modified object created by the **MSTR** processor, contains these flags for all of the nodes in the master model (*i.e.*, for the finite element nodes, the pseudo-nodes, and the alpha-nodes). The form of the object is the same as the original form except that the finite element nodes are all listed first, all of the pseudo-nodes follow, and all of the alpha-nodes are listed at the end of the table.

10.3. Element Data Objects

10.3.1. General Description

For finite elements, each element attribute table contains potentially one record per finite element. Because of the variable nature of the interface element (that is, each interface element may have a different number of nodes, pseudo-nodes, and alpha-nodes), each interface element is treated as a different element type. Therefore, each element table contains data for only one interface element. Several new element attribute tables have been created. These new objects (all created by the EI processor) are summarized in Table 10.2 and discussed in detail in subsequent sections.

Table 10.3. Summary of New Element Attribute Tables (EATs)

Object Name	Purpose
EltName.ELTYPE...*mesh*	List of finite element types to which each finite element interface node is attached.
EltName.NODSS...*mesh*	Substructure identifier for each node of the element.
EltName.NORMALS...*mesh*	Normal vectors for finite element nodes and pseudo-nodes.
EltName.PARAMS...*mesh*	Element integer parameters.
EltName.SCALE...*mesh*	Scale Factor.
EltName.SCOORD...*mesh*	Path coordinates for all of the element nodes.
EltName.SSID...*mesh*	List of substructures to which the element is attached.
EltName.SSLDI...*mesh*	Logical device index (ldi) of each substructure library.
EltName.TANGENT_S...*mesh*	Element path tangent vectors for finite element nodes and pseudo-nodes.
EltName.TANGENT_T...*mesh*	Element surface tangent (perpendicular to the path tangent) for finite element nodes and pseudo-nodes.
EltName.TGC...*mesh*	Computational to global transformation matrices for the finite element nodes in each element.

In the following discussion, the phrase "element nodes" denotes all of the finite element nodes, the pseudo-nodes and the alpha-nodes associated with an element. In all interface element element data objects (not just those listed in this section), the finite element nodes are listed first, the pseudo-nodes follow, and the alpha-nodes are at the end of the list. The total number of element nodes is therefore

$$\mathbf{Nen} = \sum_{i=1}^{ns} \langle \text{number of nodes along interface for substructure } i \rangle + \begin{matrix} \text{number of evenly-spaced pseudo-nodes} + \\ \text{number of alpha-nodes} \end{matrix}$$

where *ns* is the number of substructures attached to the element.

10.3.2. Element Attribute Table (EAT): EltName.ELTYPE...*mesh*.

The **EltName.ELTYPE...*mesh*** table contains a list of the finite element types attached to each node of the interface element.

EAT: EltName.ELTYPE...*mesh*	
Attribute	Element 1
EltAtt(*)	ElType(i,j,k)

where **Eltype(i,j,k)** lists the element types, **j**, connected to interface node **i** of substructure **k**. The finite element types are described by the number of nodes along the edge of the finite element. For example, if the ith node of substructure 1 is connected to a four and a nine node element along the interface the array values will be **Eltype(i,1,1)**=2, **Eltype(i,2,1)**=3. The nodes are in substructure order and range over the maximum number of finite element element types. Zeroes are stored for the pseudo-nodes and the alpha–nodes.

10.3.3. Element Attribute Table (EAT): EltName.NODSS...*mesh*.

The **EltName.NODSS...*mesh*** table contains a list of the substructures attached to each node of the interface element.

EAT: EltName.NODSS...*mesh*	
Attribute	Element 1
EltAtt(Nen)	$NodSS_1, NodSS_2, ... NodSS_{Nen}$

where **Nen** is the total number of nodes (as defined previously) and $NodSS_i$ is the substructure to which the ith node is connected. A value of 0 (zero) indicates that the node is a pseudo-node or an alpha-node. This object provides a cross reference which allows all merge functions to be performed in the **MSTR** processor and not in the **EI** processor.

10.3.4. Element Attribute Table (EAT): EltName.NORMALS...*mesh*.

The **EltName.NORMALS...*mesh*** table contains a normal vector for each pseudo-node (in the interface frame) and each finite element node (in the edge frame) of the interface element. A zero vector is saved for the alpha-nodes as they have no physical location.

EAT: EltName.NORMALS...*mesh*	
Attribute	Element 1
EltAtt(3,Nen)	$Vector_1, Vector_2, ... Vector_{Nen}$

where **Nen** is the total number of nodes (as defined previously) and $Vector_i$ is the unit normal vector for the ith element node.

10.3.5. Element Attribute Table (EAT): EltName.PARAMS...*mesh*.

The **EltName.PARAMS...*mesh*** table contains a list of the interface element integer parameters.

EAT: EltName.PARAMS...*mesh*	
Attribute	Element 1
EltAtt(*)	Params(1),Params(2)...

Currently, this object is used to store the following integer data:

Attribute	Meaning
Params(1)	Number of substructures connected through this element.
Params(2)	Order of interpolation used for the deformation along this interface element (1,2,3 corresponds to a piecewise linear, quadratic spline, and cubic spline functions respectively).
Params(3)	Order of interpolation used for the geometry along this interface element (1,2,3 corresponds to a piecewise linear, quadratic spline, and cubic spline functions respectively).
Params(4)	Number of pseudo-nodes along this interface element.
Params(5)	Number of alpha-nodes along this interface element.
Params(6:n6)	Number of alpha-nodes along each substructure. n6 = 5+(number of substructures attached to this element).
Params(n6+1:n7)	Number of alpha-nodes per element for each substructure n7 =n6+(number of substructures attached to this element).
Params(n7+1:n8)	Number of f.e. nodes per substructure along this interface element. n8 =n7+(number of substructures attached to this element).
Params(n8+1)	Drilling freedom for this interface element.
Params(n8+2)	Drilling freedom suppression flag for this interface element.

10.3.6. Element Attribute Table (EAT): EltName.SCALE...*mesh*.

The **EltName.SCALE...*mesh*** table contains a scale factor for the interface element.

EAT: EltName.SCALE...*mesh*	
Attribute	Element 1
EltAtt	Scale

where **Scale** is an optional real scale factor used to ensure that the stiffness matrix does not become ill-conditioned.

10.3.7. Element Attribute Table (EAT): EltName.SCOORD...*mesh*.

The **EltName.SCOORD...*mesh*** table contains a list of the path coordinates for all interface element nodes.

EAT: EltName.SCOORD...*mesh*	
Attribute	Element 1
EltAtt(Nen)	$s_1, s_2, s_3, \ldots s_{Nen}$

where s_i is the interface path coordinate for ith element node. While the current interface element implementation will only accommodate a one-dimensional interface, this object may, in general, accommodate a two dimensional interface.

10.3.8. Element Attribute Table (EAT): EltName.SSID...*mesh*.

The **EltName.SSID...*mesh*** table contains a list of the substructures attached to the element.

EAT: EltName.SSID...*mesh*	
Attribute	Element 1
EltAtt(MaxSpE)	$SSID_1, SSID_2, \ldots SSID_{MaxSpE}$

where MaxSpE is a parameter which defines the maximum number of substructures per interface element and $SSID_i$ is the substructure to which this element is connected.

10.3.9. Element Attribute Table (EAT): EltName.SSLDI...*mesh*.

The **EltName.SSLDI...*mesh*** table contains a list of the substructures attached to the element.

EAT: EltName.SSID...*mesh*	
Attribute	Element 1
EltAtt(MaxSpE)	$SSLDI_1, SSLDI_2, \ldots SSLDI_{MaxSpE}$

where MaxSpE is a parameter which defines the maximum number of substructures per interface element; $SSLDI_i$ is the logical device index of the ith substructure.

10.3.10. Element Attribute Table (EAT): EltName.SSTGC...*mesh*.

The **EltName.SSTGC...*mesh*** table contains the computational-to-global transformation vector for each finite element node of the interface element. A zero vector is saved for the I-nodes.

EAT: EltName.SSTGC...*mesh*	
Attribute	Element 1
EltAtt(3,Nen)	$Vector_1, Vector_2, ... Vector_{Nen}$

where **Nen** is the total number of nodes (as defined previously) and $Vector_i$ is the finite element nodal computational-to-global vector for the ith element node.

10.3.11. Element Attribute Table (EAT): EltName.TANGENT_S...*mesh*.

The **EltName.TANGENT_S...*mesh*** table contains a tangent vector along the interface for each pseudo-node and each finite element node of the interface element. A zero vector is saved for the alpha-nodes as they have no physical location.

EAT: EltName.TANGENT_S...*mesh*	
Attribute	Element 1
EltAtt(3,Nen)	$Vector_1, Vector_2, ... Vector_{Nen}$

where **Nen** is the total number of nodes (as defined previously) and $Vector_i$ is the unit tangent vector along the interface for the ith element node.

10.3.12. Element Attribute Table (EAT): EltName.TANGENT_T...*mesh*.

The **EltName.TANGENT_T...*mesh*** table contains a transverse surface tangent vector for each pseudo-node and each finite element node of the interface element. A zero vector is saved for the alpha-nodes as they have no physical location.

EAT: EltName.TANGENT_T...*mesh*	
Attribute	Element 1
EltAtt(3,Nen)	$Vector_1, Vector_2, ... Vector_{Nen}$

where **Nen** is the total number of nodes (as defined previously) and $Vector_i$ is the unit surface tangent vector for the ith element node.

10.3.13. References

10.3-1 Stanley, G. M. and Swenson, L., *HDB Object-Oriented Database Utilities for COMET-AR*, NASA CSM Contract Report, August, 1992.

THIS PAGE INTENTIONALLY BLANK

Appendix A.
GLOSSARY

THIS PAGE INTENTIONALLY BLANK

Appendix A: Glossary

alpha-nodes	Nodes generated by the interface element processor which are assigned the traction degrees of freedom (if any exist) along the interface. Alpha-nodes have no meaningful physical location at this time.
analysis procedure	A sequence of commands, written in the COMET-AR command language CLAMP. An analysis procedure may call upon other procedures or processors.
analysis processor	A software processor which performs one or more specific analysis tasks.
application procedure	An analysis procedure used to solve a specific application problem. Will typically call analysis procedures and execute analysis processors.
application	The structural analysis problem to be solved.
AR (Adaptive Refinement)	A type of analysis which involves the automatic adaptation of the finite element model to ensure a user-specified accuracy in the solution.
COMET-AR	Acronym for the COmputational MEchanics Testbed - with Adaptive Refinement. A general-purpose, modular, structural analysis software system.
command language	An interpretable language consisting of a stream of commands that controls the execution of a software system.
computational database	The database used to store the data associated with the finite element model, the solution, and, possibly, post-processed data.
computational frame	Reference frame in which the solution is obtained at each node.
cover procedure	A command language procedure used to mask the execution of one or more processors.
database	A collection of stored data.
data library	A term used to refer to a named file within a COMET-AR database.
data object	A tabular data structure that contains both data attributes and utilities that perform operations on the data.
data set	The data attributes part of a data object.

developer	A person that develops new processors and/or procedures which implement new methods (*e.g.*, a new type of finite element, a new type of solution strategy, a new interface element). Those who use the COMET-AR system are typically divided into two groups: users and developers.
directive	A special command record that is processed directly by CLIP and not transmitted to the running processor.
edge frame	Reference frame attached to the finite element edges along a substructure edge. Defines the computational frame for the alpha-nodes.
element frame	Reference frame attached to each finite element.
GEP (Generic Element Processor)	A software template for all COMET-AR structural element processors; provides a common generic user and developer interface to such processors. Also referred to as ES. Individual processor names begin with ES (*e.g.*, ES1, ES10).
global frame	Fixed reference frame in which nodal coordinates are defined.
interface element	A special type of finite element which connects independently created finite element models.
interface frame	Reference frame attached to each interface element. Defines the computational frame for the pseudo-nodes.
library file	A term used to refer to a named file within a COMET-AR database.
macrosymbol	A character string that character string that represents another character string or a number. Like a variable name in FORTRAN.
master model	A single model created by combining two or more finite element models.
nodal compatibility	A one-to-one nodal correspondence across substructure boundaries.
procedure	A command language program written in CLAMP and delimited by a procedure header (`*procedure`) and terminator (`*end`) which may be parameterized by arguments specified in a calling sequence.
procedure argument	A parameter specified in the header of a command procedure that may be used to replace text within the procedure.
processor	A semi-independent software program which exchanges information with the database. Processors typically read data from and write new data to the database.

pseudo-nodes Nodes generated by the interface element processor which are assigned the displacement degrees of freedom along the interface. Pseudo-nodes are evenly spaced along the interface and are assigned coordinates accordingly.

qSymbol A FORTRAN integer parameter used in place of an explicit character string. Several hundred pre-defined qSymbols are used in the COMET-AR system.

script file A set of UNIX commands that perform a specific function (*e.g.*, run a particular analysis) and are placed within an executable file.

substructure A semi-independent part of a global finite element model. Substructures are typically used to extract local models from global models and to model different components which are part of a larger structure.

template file A file which provides the user with an example of the required input for a given procedure or processor. Template files have been provided for the interface element processors and control procedure.

user Any individual that uses the COMET-AR system for performing an analysis. Those who use the COMET-AR system are typically divided into two groups: users and developers.

THIS PAGE INTENTIONALLY BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1995	Contractor Report

4. TITLE AND SUBTITLE

A Generic Interface Element for COMET-AR

5. FUNDING NUMBERS

C NAS1-19000
C NAS1-19700

WU 505-63-53-01

6. AUTHOR(S)

Susan L. McCleary and Mohammad A. Aminpour

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lockheed Engineering and Sciences Company
Hampton, Virginia 23681-0001 and
Analytical Services and Materials, Inc.
Hampton, VA 23681-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

NASA CR-195075

11. SUPPLEMENTARY NOTES

Langley Technical Monitor: Jerrold M. Housner

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 39

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This report documents the implementation of an interface element capability within the COMET-AR software system. The report is intended for use by both users of currently implemented interface elements and developers of new interface element forumlations. For guidance on the use of COMET-AR the reader should refer to Ref. 1-1. A glossary is provided as an Appendix to this report for readers unfamiliar with the jargon of COMET-AR. A summary of the currently implemented interface element formulation is presented in Section 7.3 of this report. For detailed information on the formulation of this interface element, the reader is referred to Refs. 1-8 through 1-10.

14. SUBJECT TERMS

Finite Element, Interface Element, COMET-AR, Shells, Substructures, Global/Local

15. NUMBER OF PAGES

196

16. PRICE CODE

A09

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED		

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102