CA MICS®Resource Management

System Modification Guide

Release 12.9

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2014 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction 9

1.1 Modification Philosophy 12
1.1.1 Vendor Product Maintenance and Support 14
1.1.2 Tailoring CA MICS to the User's Special Needs 14
1.2 Document Organization 19

Chapter 2: System Overview 21

2.1 Installed CA MICS System 21
2.2 CA MICS Naming Conventions 24
2.2.1 General Naming Conventions 24
2.2.2 SOURCE Modules 25
2.2.3 MACAUTOS Modules 26
2.2.4 PROTOLIB Modules 28
2.2.5 Generated Jobs 29
2.2.6 INCLLIB Modules 30
2.2.7 Documentation Members 31

Chapter 3: Maintenance Policies and Procedures 35

3.1 Product Maintenance Policies 35
3.2 User Maintenance Procedures 37
3.2.1 Applying Supplied Maintenance 37
3.2.2 User Change Control Procedure 39
3.2.3 Change Control Test Database 47

Chapter 4: CA MICS Facilities 51

4.1 Overview 51
4.2 CA MICS Component Generator (MCG) 54
4.2.1 Generation Dependencies 55
4.2.2 Generated Outputs 56
4.2.3 MCG Operation 67
4.2.4 Generator Definition Statements 71
4.2.5 Guidelines for Coding MCG Statements 148
4.2.6 MCG Illustrations 149
4.3 User Exit Facilities 151
4.3.1 Definition and Methodology 153

4.3.2 Types of CA MICS Exits 169
4.4 SAS Functions and Call Routines 181
4.4.1 SAS Call Routine GBLFCN 182
4.4.2 SAS Function and Call Routine MALABEL 183
4.5 User-defined Informats and Formats 184
4.5.1 Creating Complex-level Catalog Entries 188
4.5.2 Creating Unit-level Catalog Entries 192
4.5.3 Creating Catalog Entries During Operational Jobs 197
4.5.4 Input/Output Formats 199
4.6 Documentation Programs 199
4.6.1 Document Utility Control Commands 202
4.6.2 DOCPICK Program Documentation 210
4.6.3 Document Program Documentation 218
4.7 Calendar Derivations 219
4.7.1 Calendar Support Macro Descriptions 221
4.7.2 Non-Standard Calendar Modifications 225
4.8 Account Code Determination Routine Coding 258

Chapter 5: CA MICS/SAS Coding Conventions 263

5.1 SAS Coding 263
5.2 CA MICS Coding 268

Chapter 6: Database Tailoring and Retrofitting Techniques 277

6.1 Tailoring Guidelines 278
6.2 Tailoring the Database 281
6.2.1 Tailoring Data Elements 283
6.2.2 Tailoring Database Files 316
6.2.3 Tailoring Archive Files 327
6.3 Retrofitting Guidelines 349
6.3.1 Correcting a File After Creation 350
6.3.2 Retrofitting Database Files 354
6.3.3 Retrofitting Archive Files 361

Chapter 7: Writing Field-developed Applications 373

7.1 Development Steps 373
7.2 Deciding What the Application Must Do 374
7.2.1 Input Source 375
7.2.2 Reports Needed 381
7.2.3 Files and Elements Needed 383
7.3 Writing the Code 388

7.3.1 Define the Application (fdaGENIN) 389
7.3.2 Define the Input Format Routine (DYfdaFMT) 399
7.3.3 Define Reports (DYfdaEXC, fdaEVA, and DYfdaMBO) 418
7.4 Testing, Documenting and Deploying the FDA 424
7.5 MSI Definition Notes 429
7.5.1 Default Data Types 429
7.5.2 Resolving Data Element Type Conflicts 430
7.5.3 Formats assigned to Input SAS Variables 430
7.5.4 Account Codes 430
7.5.5 File Sort Sequences 431
7.5.6 @@FIRST and @@LAST 431
7.5.7 KEEP and DROP Processing 432
7.5.8 CA MICS Chargeback and Accounting Interface 432
7.5.9 Data Dictionary Aliases 433
7.6 MSI Unit Level Parameters 433
7.6.1 FDA Processing Options (fdaOPS) 434
7.6.2 Input Data set Specification (INPUTfda) 440
7.6.3 Original System Id Assignment (fdaSYSID) 440

Chapter 8: CA MICS NSM Integration (MTI) 441

8.1 NSM Systems Performance 441
8.2 CA MICS NSM Integration (MTI) 442
8.3 Features 444
8.3.1 MTI Dictionary 444
8.3.2 MTI Component Creator 445
8.3.3 CA MICS NSM Data Transfer (MICSXFR) 446
8.4 MTI Reporting 446
8.4.1 MTILD1 - Defined Resource Types 446
8.4.2 MTILD2 - Defined Resource Subtypes 448
8.4.3 MTILD3 - Machine to ORGSYSID Table 450
8.4.4 MTILD4 - MTI Defined Machine Types 451
8.4.5 MTILD5 - Defined MTI Components 451
8.4.6 MTILD6 - MTI Components Detail 452
8.4.7 MTILD7 - MTI Component Element Detail 453
8.5 Integration Checklists 454
8.5.1 Activating MTI in the CA MICS Complex 454
8.5.2 Creating an MTI Component 456
8.5.3 Updating an existing MTI Component 459
8.5.4 Recovering the MTI Dictionary 466
8.6 Considerations 469
8.6.1 SYSIDs vs. Machine IDs 469

8.6.2 Data Types 471
8.7 MTI Unit Level Parameters 472
8.7.1 Component Processing Options (cccOPS) 473
8.7.2 Input Data set Specification (INPUTccc) 483
8.7.3 Original System Id Assignment (cccSYSID) 484
8.8 MTI Operational Jobs 486
8.8.1 Initialize the Dictionary (DICINIT) 486
8.8.2 Load the MTI Dictionary (DICLOAD/cccLOAD) 486
8.8.3 Create an MTI Component (MTICREAT) 487
8.8.4 Create the CA MICS Data Dictionary Entries(DICDSHEL) 488
8.8.5 Repair the MTI Dictionary (DICREPAR) 489
8.8.6 Update an MTI Component (cccUPSCN/cccUPDAT) 489

Appendix A: MSI FDA Checklist 491

Appendix B: MTI Control Statement Reference 499

B.1 Initialize the Dictionary (DICINIT) 500
B.2 Load the MTI Dictionary (DICLOAD) 500
B.3 Create the CA MICS Data Dictionary Entries(DICDSHEL) 501
B.4 Repair the MTI Dictionary (DICREPAR) 502
B.5 Create or update an MTI Component (MTIGENER) 502

Chapter 1: Introduction

The System Modification Guide relies on information contained in other CA MICS guides. Before attempting to modify your CA MICS system, you should have a good working knowledge of the content of the other guides. In fact, the content of some of these manuals may be affected by modifications you make to your CA MICS system since some of the text in those books is generated.

This manual is the CA MICS System Administrator's primary reference for tailoring and modifying a CA MICS system.

NOTE: Please record all modifications made to CA MICS, whether vendor maintenance or user modification, in the CA MICS System Administrator Guide (SAG). To ensure the smooth operation of CA MICS, the SAG should contain an accurate record of changes applied to CA MICS.

Due to the number of references made to the CA MICS Planning, Installation, Operation, and Maintenance Guide, it is referred to as the PIOM or PIOM Guide.

```
This guide is part of the documentation for the CA MICS
System.  The complete list of guides in the CA MICS
documentation series is:

**************************
* GUIDES FOR THE END-USER *
**************************

CA MICS Database Structure and Content Guide
CA MICS Document Access Guide
CA MICS MICF Reference Guide
CA MICS MICF User Guide

*************************************
* GUIDES FOR THE SYSTEM ADMINISTRATOR *
*************************************

CA MICS How to Use the PSP
CA MICS Planning, Installation, Operation, and Maintenance
   Guide (PIOM)
CA MICS Standard Reports Guide
CA MICS System Administrator Guide
CA MICS System Modification Guide


******************
* PRODUCT GUIDES *
******************

CA MICS Accounting and Chargeback Option Concepts and
   Overview Guide
CA MICS Accounting and Chargeback Option User Guide
CA MICS Analyzer for TSO Guide
CA MICS Analyzer Option for CA IDMS Guide
CA MICS Analyzer Option for CICS Guide
CA MICS Analyzer Option for DB2 Guide
CA MICS Analyzer Option for IMS Guide
CA MICS Analyzer Option for MeasureWare Guide
CA MICS Analyzer Option for MQSeries Guide
```

CA MICS Analyzer Option for VAX/VMS Guide
CA MICS Analyzer Option for VM/CMS Guide
CA MICS Analyzer Option for VSE/Power Guide
CA MICS Batch and Operations Analyzer Guide
CA MICS CA ASTEX Option Guide
CA MICS Capacity Planner Option Guide
CA MICS Data Transfer Option for VM/CMS Guide
CA MICS Hardware and SCP Analyzer Guide
CA MICS IMS Log Data Extractor Option Guide
CA MICS Network Analyzer Option Guide
CA MICS Performance Manager Option Guide

CA MICS Space Analyzer Option Guide
CA MICS Space Collector Option Guide
CA MICS StorageMate Option Guide
CA MICS System Reliability Analyzer Option Guide
CA MICS Tandem Option Guide
CA MICS Tape Analyzer Option Guide
CA MICS Web Analyzer Option Guide

SAS (R) is the registered trademark of SAS Institute Inc., Cary, North Carolina 27511, U.S.A. SAS/GRAPH, SAS/FSP, and any other SAS products referenced in this documentation are trademarks of SAS Institute Inc.

IDMS is a registered trademark and IDMS/R is a trademark of CA, Inc.

The Monitor for CICS is a trademark of Landmark Systems, Inc., Vienna, VA 22180.

This section contains the following topics:

1.1 Modification Philosophy (see page 12)
1.2 Document Organization (see page 19)

1.1 Modification Philosophy

The diverse nature of z/OS systems and the applications for which they are used creates equally diverse measurement requirements. CA MICS was designed as a measurement tool capable of effectively meeting these diverse requirements. The approaches provided to do this can be divided into three groups, as follows:

1. Standard Features. Many needs specific to a given z/OS site are met with no modification to CA MICS. The principal vehicle for this type of flexibility is the set of CA MICS parameters specified during installation. This type of solution to a special requirement should always be considered first and used if such a solution is possible.

2. User Modification Features. This group of features includes the provision of user exits in the standard CA MICS code, the CA MICS Database Tailoring Facility, and the CA MICS User Component Generator. These facilities provide the tools and methodology for controlled modification and extension of CA MICS by the user. Modifications of this type are the primary topic of this guide.

3. User Modifications to CA MICS Source Code. Most of the source code for CA MICS is shipped with the product, and this policy makes major changes to the source code possible. However, this is NOT recommended, and seldom is necessary. Modifications of this type should not be undertaken without consulting the CA MICS Product Support Group first and investigating all possible alternatives.

If you have any special measurement requirement, and are considering making some modification to CA MICS, we strongly recommend that you adhere to the following general principles:

    1. Always use the most direct approach available to meet your need. Use features in group 1 above before those in group 2. This will reduce the time needed to make and maintain the modification.

2. If a modification of CA MICS source code as described in group 3 above is necessary, then modify as few lines of CA MICS code as possible. If a change requires a significant amount of coding, use the SAS MACRO facility to include the code (through a standard user exit or user-defined user exit) in CA MICS code.

3. Follow the guidelines in this manual for designing modifications and evaluate the benefits and impacts of the change before making it.

4. Test all changes before implementation. CA MICS provides standard facilities for testing via Test database units. Test methods are described in this guide.

5. Store changes in IEBUPDTE format. The LOCALMOD change library provided for doing this is described in following chapters.

6. Keep a complete record of all changes to CA MICS. Have the records available at the time you call us for product support.

7. Do not modify the source code for the CA MICS generator facilities. This includes the CA MICS Component Generator,the CA MICS JCL Generator, the CA MICS parameter generators, and the CA MICS algorithm generator.

8. Contact the CA MICS Product Support Group to discuss modifications as needed. This is especially appropriate during the planning and evaluation of modifications. A brief contact at this point may help you avoid major difficulties.

These topics are discussed in following sections:

1 - Vendor Product Maintenance and Support
2 - Tailoring CA MICS to the User's Special Needs

1.1.1 Vendor Product Maintenance and Support

Maintenance for CA MICS is distributed on a semi-annual basis, through the CA MICS Product Support Program. This maintenance will enhance CA MICS performance or useability, or correct problems. Maintenance we distribute can be applied to all unmodified CA MICS systems with little or no impact on the existing configuration. Some users may not be affected at all by vendor-supplied CA MICS maintenance, which may apply to some part of CA MICS your site is not currently using.

While some of the maintenance may seem unnecessary for your site, it is to your immediate and long-term benefit to apply the maintenance to your CA MICS system in a timely manner (eg. upon receipt). There are several reasons for this:

1. Future maintenance may be based on the premise that previous maintenance has been applied.

2. It may not be obvious which changes apply to your site and which do not.

3. As the operating system environment and other system monitors are enhanced with new versions, CA MICS is also upgraded to process new and reformatted data.

4. As you upgrade your operating system environment, you are likely to begin using more and more CA MICS features.

Before you undertake to modify the CA MICS system, you must have a good understanding of how vendor-supplied product maintenance will apply to your modifications. Thoroughly read section 3.3 of this manual, and examine the CA MICS System Administrator Guide and CA MICS Product Support Program materials in detail.

1.1.2 Tailoring CA MICS to the User's Special Needs

As discussed in section 1.1, CA MICS provides several approaches to meeting the special needs of different system measurement configurations. The approaches most commonly used for meeting special requirements are:

Using the Standard CA MICS Parameters

These parameters are used to specify normal system environmental descriptions to CA MICS. The parms provide for such things as:

- Naming and describing the CPUs from which data is gathered.
- Describing the monitoring methods, such as CICS monitor used (CMF or Monitor), and whether to use RMF trace data or not.
- How to title reports, and which standard reports to generate.

Tailoring the Database

The CA MICS database can be changed by keeping additional data elements, deleting inappropriate data elements, enabling or disabling archiving for selected CA MICS files, or modifying the sequence of certain files.

Adding User Components to the Database

These components may be used to add another input data source to CA MICS, or create additional files from an existing data source.

System Modification Using Standard EXITs

The exits may be used to modify the content of existing CA MICS data elements, to select input records by your own algorithms, or to introduce your accounting algorithms.

Modification of CA MICS Source Code

Modifications to CA MICS source code implemented outside the structure of the standard CA MICS modification tools fall into this category.

1.1.2.1 Using Standard CA MICS Parameters

The purpose of the CA MICS parameter library construction is to provide user definitions for the CA MICS execution environment. The CA MICS database complex, and each database unit installed within it, have a parameter library used for this specification.

The parameter library is an OS partitioned data set that can be conveniently edited and modified by any means at the user's disposal. Simple changes in one or more members in the library may cause major changes to occur in the operation of CA MICS. The parameter structure is defined in the CA MICS Planning, Installation, Operation and Maintenance Guide.

Most of the members are used by CA MICS during the installation process. One (or more) of the CA MICS generator facilities reads the parameter members, bases decisions on the user-provided coding, and constructs JCL or SAS logic based on the specifications. An example of installation time parameter definitions is found in prefix.MICS.PARMS(SITE). Statements in this member define the processing options for unit database identification and date processing to be used as input to the BASPGEN process.

CA MICS uses some of the parameter library members during the execution of various operational jobs. This execution time use of user-specified parameters is similar to the use of some SYS1.PARMLIB functions when an operating system is loaded. These definitions can affect CA MICS execution from run to run, without rerunning a generation process. An example of execution time parameter definitions may be found in prefix.MICS.PARMS(EXECDEF). Some of the statements in this member choose the management objective reports and exception analysis routines to execute in a DAILY process.

A great deal of tailoring flexibility is built into the CA MICS parameter library mechanism. When considering a CA MICS user modification, review the parameter definitions in the CA MICS Planning, Installation, Operation and Maintenance Guide. A standard CA MICS parameter specification may already exist to cover the situation at hand.

1.1.2.2 Tailoring the Database

Database tailoring is a broad issue. Many kinds of tailoring, in various levels of complication, are available to the CA MICS user. Chapter 6 of this manual describes certain database tailoring techniques. By following the outlines in that chapter, the user may easily perform such functions as:

- Adding data elements to a file

- Deleting data elements from a file

- Changing existing data element attributes

- Modifying attributes or names of common data elements

- Adding a file to a timespan

- Deleting a file from a timespan

- Changing a file's sequence in a timespan

- Enabling or disabling archiving of a file

Such modifications are not to be taken lightly. There are appropriate caveats in that chapter on the compromises (if any) associated with each modification type. However, such modifications produce very satisfactory results when the changes are done according to the guidelines given and are thoroughly tested before implementation.

1.1.2.3Adding User Components to the Database

CA MICS products process data from the commonly used z/OS and distributed system measurement data sources. However, there are many subsystems implemented at user sites that, while of intense local interest, are not treated by CA MICS products. This category of data sources may be included in your CA MICS system by developing a CA MICS user component.

Chapter 7 of this manual describes the methods a user can employ in writing a CA MICS component. There are many advantages to implementing a user reporting system under the CA MICS structure, within the existing CA MICS database complex. For one thing, CA MICS provides almost all of the database manipulation and maintenance functions. The user does not have to code functions such as backup jobs, scheduling subsystems, archiving programs, and online database aging techniques.

Also, the CA MICS method of component generation requires the user to generate the least possible code to develop the component. Several debugging aids are provided to make the development process even easier.

The existing CA MICS operational framework includes full checkpointing of the database, update run audit history files, and a run status reporting mechanism.

Undertaking to write a CA MICS user component is not a trivial matter. However, the guidelines and checklists contained in chapter 7 of this manual make the process much more attractive than implementing a stand-alone system.

1.1.2.4 System Modification Using Standard EXITs

There is a large number of standard exit points within the CA MICS complex. These exit points exist for the sole purpose of allowing the user to make effective changes to the CA MICS processing logic without requiring that user to modify CA MICS source code. While modification of source code is possible, and is discussed in this manual, such modifications reduce the release independence of the user's system definition.

The points at which user exits are available can be grouped into these categories:

- Standard CA MICS exits. These exit routines are required for all CA MICS users, and are used to provide code that MUST be included for CA MICS user-dependent functions. For example, these exits provide for the determination of cost center identifiers and the grouping of batch data into categories suitable for summarization. These exits reside in the database complex or unit parameter libraries, as separate members in the library. See the CA MICS Planning, Installation, Operation and Maintenance Guide for information on these standard exits.

- File exits. There is an exit in CA MICS that is taken before an observation is written to the database file. This type of exit exists for every file of every component in the CA MICS complex. It is possible to use these exits to do such things as implementing accounting algorithms or changing the values of CA MICS-calculated data elements in the files.

- Operational exits. There are several points within the CA MICS complex at which operational decisions are likely to need installation-dependent logic. These points have been determined on the basis of experience at many CA MICS sites. Code which affects the criteria for input record selection and logic for unusual file manipulation may be implemented in these exits.

Chapter 4 of this manual describes the exit types that were noted and gives guidelines for using them.

1.1.2.5 Modification of CA MICS Source Code

Replacing parts of CA MICS logic, prototype members used for generation, or other material with installation-unique coding is a type of modification that should not be undertaken lightly.

Chapters 4 and 7 of this manual contain guidelines for users who want to investigate the possibilities of implementing such modifications. If such changes to the actual source code of CA MICS are considered, pay close attention to the discussions of how this type of modification increases your dependence on a particular release of CA MICS.

However, given the above, it is quite possible to modify the CA MICS system at the source code level. Many CA MICS users have had excellent results in meeting unusual requirements by carefully controlled modifications of this type. Examples of requirements that necessitated source code modifications are:

- Job step accounting

- Unusual requirements that affect the calculation of the ZONE data element

These unusual processing requirements cannot be treated through other CA MICS tailoring or exit facilities. The key to the success of such modifications is to follow the instructions in chapters 4 and 7 and to keep complete and accurate records of the changes made.

1.2 Document Organization

The remaining chapters in this document cover the following topics:

Chapter 2, System Overview, gives a tutorial on the structure of a CA MICS system. It describes the functions performed in the installation and operation of CA MICS, explains how to identify modules in the CA MICS libraries that perform any particular function, and tells how a CA MICS database complex is structured.

Chapter 3, Maintenance Policies and Procedures, describes the difference between changes to CA MICS that CA provides and the modifications a user would make to CA MICS. Each type of vendor maintenance is explained, and a procedure for creating and applying user changes is given.

Chapter 4, CA MICS Facilities, explains the parts of CA MICS that enable system changes to be applied quickly and accurately. It explains the CA MICS generator subsystems, standard CA MICS exits, vendor-supplied utilities, and the CA MICS Information Center Facility.

Chapter 5, CA MICS/SAS Coding Conventions, shows a set of considerations for SAS coding within CA MICS that all users must be aware of, before any system modifications are undertaken.

Chapter 6, Database Tailoring and Retrofitting Techniques, explains the mechanism CA MICS uses to describe CA MICS databases and how the standard descriptions can be modified to add, change, or delete individual data elements, groups of data elements, and files within timespans.

Chapter 7, Writing Field-developed Applications, is a tutorial on the use of the CA MICS component generation facility to add your own component to the CA MICS system. Discussion of two utilities available for adding your own components, MSI, is included at the end of this chapter. The other utility for adding your own components, MTI, is the subject of the next chapter.

Chapter 8, CA MICS NSM Integration (MTI), describes the features and utilities of the MTI package which you may use to integrate data from NSM into CA MICS. The chapter also provides instructions for modifying your installation to take advantage of these features and utilities.

Chapter 2: System Overview

The design of CA MICS is based on a modular construction that allows modifications to be made easily. This chapter presents an overview of the external and internal structure of CA MICS for users who may need to modify the system. It is necessary that a user have a good appreciation of this structure before any attempt is made to modify CA MICS. In addition, you should be familiar with the material presented in the other CA MICS guides before undertaking any modifications.

This section contains the following topics:

2.1 Installed CA MICS System (see page 21)
2.2 CA MICS Naming Conventions (see page 24)

2.1 Installed CA MICS System

In the simplest sense, CA MICS provides a single, common analytical process to analyze many unlike measurement sources. It enables you to systematically track and report z/OS SCP, hardware, TSO, batch, and many other activities within time-synchronized intervals. Its database structure, which is described below, enables CA MICS to contribute to enterprise systems management.

Database Structure

The highest logical unit in CA MICS is a database complex. This is a complete and independent CA MICS system. It consists of shared complex libraries and one or more database units. Each database unit consists of database unit libraries and data files.

Many organizations have a single database complex containing at least one production database unit and one test unit. Organizations that run CA MICS at multiple sites typically have a database complex at each site. Figure 2-1 depicts a database complex with three database units.

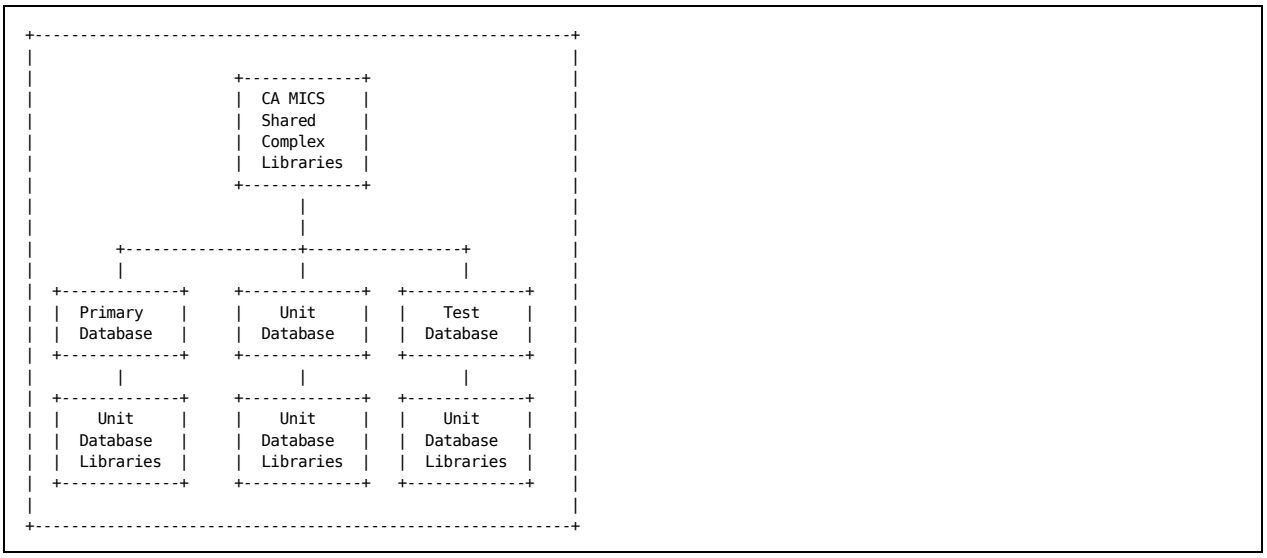


Figure 2-1. Sample CA MICS Database Complex

Database Complex

The CA MICS system is installed as a group of one or more database units which share some libraries and data areas. The entire structure of data and code libraries is called a database complex.

The shared libraries contain programs, JCL, macros, parameter definitions, and documentation. The shared data areas include TABLES (used mostly by CA MICS Accounting and Chargeback), MCOLIB (a system control area), CAPACITY, etc. They contain SAS data files which apply to the entire complex and not just a particular database.

Database Unit(s)

In addition to the shared libraries, a database complex consists of database units. Each database unit contains data files and libraries of code, JCL, and parameters needed to define, update, and access these data files.

Database units may be one of four types: primary, unit, test, or special. Each type of unit is treated differently by the JCLGEN process and by CA MICS Accounting and Chargeback. The purpose of each is described below.

Primary - Every database complex contains only one primary database unit. The distinguishing feature of a primary database unit is that its backup process backs up the MCOLIB, SCREENS, TABLES, ISPTLIB, etc., data sets that exist at the complex level.

Unit - A normal production database is called a unit database. Every unit database contains CA MICS Platform (BAS) plus one or more other products. Any combination of products is allowed, with the exception that CA MICS Accounting and Chargeback, if installed, must accompany at least one product that supports accounting.

Test - A test database unit is used for developing user components and testing user modifications to the CA MICS system. It is also used when installing maintenance and new versions of the system. Test database units differ from unit databases in that they are treated specially by CA MICS Accounting and Chargeback and the CA MICS JCL generation process. Jobs run on a test unit use special test libraries that are concatenated ahead of the standard libraries.

Special - Special databases are temporarily added to the CA MICS complex to perform special studies involving data not available in the production databases. A special unit can also be used for certain configuration planning and testing functions (through modifications to parameter library modifications).

More Information

For more information about the data sets that comprise a CA MICS system, see the section 4.3 in the PIOM.

2.2 CA MICS Naming Conventions

This section discusses the naming conventions used for the members in the CA MICS libraries. The intent of these conventions is to identify the contents of the member by the name used. If you are familiar with the conventions described in the sections that follow, you will be able to identify most CA MICS modules by member name. There are other modules which do not follow the standard naming conventions, but being familiar with the conventions will minimize the number of members which can not be recognized by name.

The naming conventions are discussed in the following sections:

1 - General Naming Conventions
2 - Source Modules
3 - MACAUTOS Modules
4 - PROTOLIB Modules
5 - Generated Jobs/CLISTs
6 - INCLLIB Modules
7 - Documentation Members

2.2.1 General Naming Conventions

The CA MICS libraries contain a large number of members. To help keep the list of member names from becoming too bewildering, several general naming conventions are followed. In addition to these general conventions, there are other conventions that are used within particular libraries or sets of libraries. Those are discussed in the sections that follow this one. None of the conventions that are described below are followed in every case. However, being familiar with the general rules will make a large number of CA MICS members identifiable just by looking at the member name.

The general naming conventions used in CA MICS are:

- Macro member names begin with '$' or '#' in the source library and with a generic name or file/component ID in the MACAUTOS library.

- The component identifier, ccc, is used in names of members that occur once for each component.

- The file identifier, fff, is used in names of members that occur once for each file.

- Members named cccPGEN are used in parameter generation.

- Members named cccCGEN are used in component generation.

- Members used for one of the CA MICS operational jobs contain some reference to the job name, e.g., DAY, WEEK, MONTH, YEAR, or DY, WK, MN, YR.

2.2.2 SOURCE Modules

The CA MICS SOURCE library contains the SAS logic executed by CA MICS. This library is used at execution time, when executing any CA MICS process. SAS modifications to this library take effect with the execution of the next job that uses the module.

The modules in this library are sequenced in statement columns 73-80. It is necessary for distributed sequence numbers to remain intact for future CA MICS standard maintenance.

Members in the CA MICS SOURCE library are generally named according the following conventions.

```
   $cccxxxx - contains SAS macros or logic to relate to the
or #cccxxxx   'ccc' component's execution in all database
              units.

   $xxxxxxx - contains generic SAS macros that apply to all
or #xxxxxxx   CA MICS components in all database units.

   #ffftSUM - contains SAS code that creates specific time
              span summarization of a CA MICS file, where
              "t" is D for the DAYS time span, W for the
              WEEKS time span, M for the MONTHS time span,
              and Y for the YEARS time span.
```

```
    #DBMccc  - contains SAS code that produces database
               models for specific components.

    xxxxxxxx - contains general purpose CA MICS SAS logic.
               The name of the member is usually an
               abbreviation of the module's function.  One
               example is "CHECK", the routine that reads the
               CA MICS daily update checkpoint file and
               abends the update step if the required
               predecessor step was not the last step run.

    xxxxxGEN - contains CA MICS SAS logic which is executed
               in a parameter generation process.

    GENxxxxx - contains CA MICS SAS logic which is executed
               in a CA MICS Component Generator process.

    cccxxxxx - contains support logic for a specific CA MICS
               component.

    ttcccxxx - contains source code for component modules
               that are executed in CA MICS operational jobs.
               These modules are named according to the
               job in which they are used, where the 'tt'
               part of the name takes the values:

                  DY - module is used in the DAILY job
                  WK - module is used in the WEEKLY job
                  MN - module is used in the MONTHLY job
                  YR - module is used in the YEARLY job

Legend:
   ccc      - a component identifier
   fff      - a file identifier
   xxx to
   xxxxxxxx - some label describing the function of the
              module.
```

2.2.3 MACAUTOS Modules

The CA MICS MACAUTOS library contains SAS %MACROS which resolve to logic executed by CA MICS. These macros are used at execution time, and may be used in any CA MICS process. Changes to this library take effect in execution of the next job that uses the module.

Normally SAS resolves a %MACRO reference from the list of all previously compiled %MACROs. %MACROS may be included by SAS into programs by explicit coding in the SYSIN DD, or may be %INCLUDEd. If a %MACRO reference cannot be resolved from the list of previously compiled %MACROs, SAS will examine data sets in the SASAUTOS concatenation for a member of the same name as the macro referenced. If it finds such a member, and the member contains a %MACRO statement defining a macro of the same name, SAS will use that macro definition automatically, without "preloading" the macro.

The CA MICS MACAUTOS library is concatenated into the SASAUTOS DD statement in all CA MICS PROCs. This allows CA MICS macros in this library to be automatically included into the program logic without preloading.

Members in the CA MICS MACAUTOS library are generally named according the following conventions.

```
    cccxxxx  - contains SAS macros or logic to relate to the
               'ccc' component's execution in all database
               units (eg.  SMFACCT)

    fffxxxx  - contains SAS macros or logic to relate to the
               'fff' file's definition in all database
               units (eg.  JOBKEEP)

    xxxxxxxx - contains generic SAS macros or general purpose
               CA MICS SAS logic that apply to all CA MICS
               components in all database units (eg. DERVST)
               The name of the member is usually an
               abbreviation of the module's function.  One
               example is "LOGUP", the routine that updates
               the CA MICS input history log.

Legend:
   ccc      - a component identifier
   fff      - a file identifier
   xxx to
   xxxxxxxx - some label describing the function of the
              module.
```

2.2.4 PROTOLIB Modules

The CA MICS prototype library contains the skeleton of CA MICS executable JCL library members. These skeletons are not used at execution time, but are input to the CA MICS JCL Generation process. Modifications to this library do not take effect immediately. Rather, the modified library must be input to a new run of the appropriate JCLGEN run, and the output of the JCLGEN run will produce JCL members with the intended modifications.

Members in the CA MICS prototype library are generally named according the following conventions.

xxxxxxxx - contains skeletons of CA MICS installation jobs, operational jobs, or support function jobs. The name of the member is an abbreviation of the function of the generated unit. One example is the CA MICS database backup utility, whose skeleton member's name is BACKUP.

ttcccsss - contains component-related skeleton members needed for building CA MICS operational jobs. These modules are named according to the job in which they are used, where the 'tt' part of the name takes the values:

DY - module is used in the DAILY job
WK - module is used in the WEEKLY job
MN - module is used in the MONTHLY job
YR - module is used in the YEARLY job

cccxxxxx - contains skeletons for JCL and logic related to the 'ccc' component's execution in all database units (e.g. SMFIUALC).

Legend:
ccc - a valid component identifier
sss - the number of a step in a CA MICS operational job
xxxxxxxx - an abbreviation of the function of the module.

2.2.5 Generated Jobs

The CA MICS CNTL library contains JCL for the CA MICS batch jobs. Most of the members in this library are generated, and should not be modified directly. Rather, the corresponding PROTOLIB member should be modified, and then the JCL should be generated from the PROTOLIB member.

Members in the CA MICS CNTL library are generally named according to the following conventions.

```
    xxxxxxxx - contains CA MICS installation jobs,
               operational jobs, and support function jobs.
               The name of the member is an abbreviation of
               the function of the job.  One example is the
               CA MICS database backup utility, which is
               named BACKUP.

    ttcccsss - contains component-related CA MICS operational
               jobs.  These modules are named according to
               the job in which they are used, where the 'tt'
               part of the name takes the values:

               DY - module is used in the DAILY job
               WK - module is used in the WEEKLY job
               MN - module is used in the MONTHLY job
               YR - module is used in the YEARLY job

    INCRccc  - contains component related jobs for
               incremental update processing.

    cccPGEN  - is the naming convention used for the
               parameter generation jobs

    JCLGENx  - is the naming convention used for the JCL
               generation jobs.

    cccCHECK - is the naming convention used for the
               component check-out jobs.

    cccxxxxx - contains other component related jobs.


Legend:
   ccc      - a component identifier
   x        - a valid character
   xxxxxxxx - some label describing the function of the
              module.
```

2.2.6 INCLLIB Modules

The CA MICS INCLLIB library contains executable SAS code. The code in an INCLLIB member is step execution control code that %INCLUDEs a predetermined list of other SAS modules that actually do the work of the step in the CA MICS job. This library is used at execution time, when executing any CA MICS process. SAS modifications to these libraries take effect with the execution of the next job that uses the module.

Members in the CA MICS source library are named according the following conventions.

```
    #cccMACS - %INCLUDEs SAS macros necessary to perform
               summarization for a component.

    xxxxxxxx - %INCLUDEs general purpose CA MICS SAS logic.
               The name of the member is usually an
               abbreviation of the module's function.  One
               example is "CYCLEGEN", which includes all the
               CA MICS modules necessary to execute the
               database cycle building process.

    DAYnnn
    WEEKnnn
    MONTHnnn
    YEARnnn  - %INCLUDEs the SAS members for the correspond-
               ing step of a CA MICS operational job.

    cccxxxxx - includes support logic for a specific CA MICS
               component.


Legend:
   ccc      - a component identifier
   fff      - a file identifier
   nnn      - a job step step number
   xxxxx to
   xxxxxxxx - a label which describes the function of the
              module.
```

2.2.7 Documentation Members

This section describes the naming conventions used for the members of the CA MICS documentation libraries. The text for the CA MICS documents is stored in three libraries:

- sharedprefix.MICS.DOC.TEXT
- sharedprefix.MICS.INC.TEXT
- sharedprefix.MICS.DIC.TEXT

The TEXT libraries contain records up to 133 bytes in length (LRECL=137). The DOC.TEXT library contains the text of all CA MICS manuals except the element Data Dictionary members. The Data Dictionary text is contained in the DIC.TEXT library.

The INC.TEXT library contains members of text that are generated at your site or are distributed but likely to be included in more than one place in the CA MICS documentation. Members in the INC.TEXT library are copied into CA MICS manuals using the functions of the CA MICS DOCUMENT program. Much of the copied text is generated by the CA MICS Component Generator.

Each section of the document, figure, and title page may be stored as a separate member in the appropriate library. The document assembly commands for each document and chapter are grouped in short record members, called picklists, which are stored in sharedprefix.MICS.DOC.TEXT.

MEMBER NAMES

Each CA MICS document is assigned a one or two letter document identifier. This identifier is incorporated into each member name to identify the document of which that member is a part. The exceptions to this are noted below.

The letter identifiers for CA MICS manuals are:

```
AH - CA MICS Tandem Option Guide
AI - CA MICS CA ASTEX Option Guide
BE - CA MICS How to Use the PSP
BS - CA MICS System Administrator Guide
C  - CA MICS Network Analyzer Option Guide
CK - CA MICS Space Collector Option Guide
CL - CA MICS IMS Log Data Extractor Option Guide
CM - CA MICS Data Transfer Option for VM/CMS Guide
FB - CA MICS Analyzer for TSO Guide
FC - CA MICS Hardware and SCP Analyzer Guide
FD - CA MICS Batch and Operations Analyzer Guide
FE - CA MICS Analyzer Option for CICS Guide
FG - CA MICS Analyzer Option for IMS Guide
FH - CA MICS System Reliability Analyzer Option Guide
FI - CA MICS Analyzer Option for VM/CMS Guide
FJ - CA MICS Space Analyzer Option Guide
FM - CA MICS Analyzer Option for MeasureWare Guide
FN - CA MICS Analyzer Option for DB2 Guide
FO - CA MICS Analyzer Option for MQSeries Guide
FQ - CA MICS Analyzer Option for VSE/Power Guide
FT - CA MICS Tape Analyzer Option Guide
FV - CA MICS Analyzer Option for VAX/VMS Guide
FW - CA MICS Web Analyzer Option Guide
H  - CA MICS Standard Reports Guide
I  - CA MICS Database Structure and Content Guide
JD - CA MICS Document Access Guide
JR - CA MICS MICF Reference Guide
JU - CA MICS MICF User Guide
L  - CA MICS Planning, Installation, Operation, and
       Maintenance Guide
M  - CA MICS System Modification Guide
PA - CA MICS Capacity Planner Option Guide
PD - CA MICS Performance Manager Option Guide
PG - CA MICS StorageMate Option Guide
PK - CA MICS Accounting and Chargeback Option
       Concepts and Overview Guide
PL - CA MICS Accounting and Chargeback Option User Guide
W  - CA MICS Analyzer Option for CA IDMS Guide
```

```
The member naming conventions, by library, follow.

LIBRARY:  sharedprefix.MICS.DOC.TEXT

  EXCnnnnn - Text for exception members.  'EXC' is the
             character string that identifies the member as
             containing an exception description, and 'nnnnn'
             is a five digit exception number.  These members
             are referenced by the picklist for each
             component guide.

             Example:  EXC01001 contains the text for
                       exception 01001, "CPU Usage Imbalance
                       Between MP/AP Processors."

  innnnnnn
   - or -
  iinnnnnn - For text members.  'i' or 'ii' is the document
             identifier, 'nnnnnnn' is one to seven digits and
             identifies the section number.  If a level of
             the section number exceeds 9, letters are used
             in the member name (e.g., 'A' for '10', 'B' for
             '11', etc.).

             Example:  L232 contains the text for section
                       section 2.3.2 of document 'L'.  LB31
                       contains the text for section 11.3.1
                       of document 'L'.

  PICi     - For the text member containing the picklist for
             document 'i'.  Section 4.8.1 describes the
             CA MICS document programs and explains
             picklists.

             Example:  PICL contains the picklist for
                       document 'L'.

  PICin    - For the text member containing the picklist for
             document 'i', chapter 'n'.

             Example:  PICL3 contains the picklist for
                       document 'L', chapter 3.

  iTITLE   - For the text member containing the document
             title for document 'i'.

             Example:  ATITLE contains the title page text
                       for document 'A'.
```

```
  iTITLEnn - For the text member containing the document
             title for document 'i', chapter 'nn'.

             Example:  ATITLE01 contains the title page text
                       for chapter 1 of document 'A'.

  icc$nnn  - For the text member containing a figure for
             document 'i', chapter 'cc', figure number 'nnn'.
             The '$' separates the chapter and figure
             numbers; the number of digits in 'cc' and 'nnn'
             is not fixed.  The only restrictions are that
             both numbers be present and that the two numbers
             plus the '$' not total more than 7 characters.

             Example:  L2$36 is figure 2-36 of document 'L'.
                       L14$2 is figure 14-2 of document 'L'.

LIBRARY:  sharedprefix.MICS.INC.TEXT

  #INSnnnn - For the text members which are copied into other
             members.  'nnnn' is a sequence number.

  #INSfffE - For the generated text members that contain data
             element lists for file fff.

  #INSfffS - For the generated text members that contain the
             sequence keys for file fff.

  #INSiiiA - For the text members that contains the file
             abstracts for the files in information area iii.

  #INScccF - For the generated text members that contain the
             portion of the file matrix for component ccc.

  #INScccQ - For the generated text members that contain the
             portion of the sequence chart for component ccc.

  #INLnnnn - For the text members which are copied into
             other members.  'nnnn' is a sequence number.
```

Chapter 3: Maintenance Policies and Procedures

CA MICS is a large system, with millions of lines of code and thousands of modules in several libraries. Maintenance of such a large system would be impossible without strict guidelines on the method and content of maintenance application.

This chapter presents the policies and procedures that have been defined for maintenance of the CA MICS product.

This section contains the following topics:

3.1 Product Maintenance Policies (see page 35)
3.2 User Maintenance Procedures (see page 37)

3.1 Product Maintenance Policies

CA MICS is regularly maintained to provide the most quality, utility and performance to our customers, and to keep up with a constantly advancing technological base. The CA MICS Product Support Program exists for the purpose of distributing maintenance in the most efficient and understandable way, with appropriate failsafe and fallback mechanisms.

Because of the evolving nature of the CA MICS product and its maintenance subsystems, there can never be a complete list of maintenance procedures. Procedures are added, changed, or dropped, as necessary, to keep CA MICS positioned as the most comprehensive I/S management product in the industry. However, we present this list of Product Maintenance Policies here for your consideration as the foundation of CA MICS maintenance policy, not as a complete list. Any user feedback on these policies will be greatly appreciated.

The following policies govern CA MICS maintenance and user modifications:

- Product Changes delivered through the Product Support Program (PSP) should be reviewed and, if applicable, be applied by the user within 120 days of receipt.

- All user written changes to CA MICS modules should be applied through the IEBUPDTE utility.

- A CA MICS module should NEVER be modified directly through an online editor such as EDIT, SPF EDIT, or the QED EDITOR. An attempt to edit a CA MICS source module will result in forcing the EDITOR into NONUM mode. The module's sequence numbers must be MANUALLY ENTERED in this mode. This is done in order to avoid the accidental renumbering of the source module, and is accomplished by setting the sequence number field of the first statement (columns 73 through 80) to 'MICS6'.

- All changes to CA MICS modules are to be noted in the Change Logs at the front of each module. Separate areas are allocated for standard CA MICS maintenance (i.e., supplied by CA) and user modifications. Each change to the module is noted by one comment line. Modifications distributed via component upgrades, releases, and the CA MICS Product Support Program automatically update the Change Logs.

- Standard CA MICS maintenance should be applied via PSP Online.

  If necessary, full module replacements will be supplied. The resequencing of module numbers will be avoided wherever possible.

- Standard CA MICS maintenance is distributed for the following libraries, and in most cases, will NOT be applied to the library automatically. This is to safeguard user modifications that have been made to CA MICS.

  CA MICS.USER.SOURCE
  CA MICS.PARMS

- Integration of CA MICS maintenance with user modifications (i.e., retrofitting) is the responsibility of user.

The CA MICS design provides the user with a high degree of flexibility in tailoring a system through its standard options, its generation processes, and the provision to enable user modification of system code. If this latter provision is used prudently and with good change control practices, the results will be advantageous to all.

3.2 User Maintenance Procedures

It is the user's responsibility to maintain the CA MICS complex as installed at the user's site or sites. This can be a complicated process due to the fact that there are many individual modules in the typical CA MICS installation, many libraries in which they reside, and thousands of lines of code.

Maintenance of such a system would be impossible without strict adherence to an organized maintenance scheme. The scheme chosen for CA MICS employs complete change documentation and the use of the standard IBM utility IEBUPDTE. The reason for this strict approach is to ensure that necessary maintenance can be applied quickly and accurately. For unmodified CA MICS systems, only vendor-supplied maintenance is involved. For modified CA MICS complexes, the application of vendor-supplied maintenance can become a very large task if modifications are not properly developed.

3.2.1 Applying Supplied Maintenance

CA MICS system maintenance is provided in the following ways:

- Product Service Pack (PSP) maintenance

  You receive this maintenance twice a year, once in the late spring or early summer, and then in late fall or early winter. The PSP comprises both technical enhancements and scheduled maintenance for CA MICS products. Refer to the CA MICS How to Use the PSP Guide for information about applying PSP maintenance.
- Product Information Bulletins (PIBs)

  Product Information Bulletins contain official news about a specific CA MICS product. PIBs are normally distributed with maintenance, but can be distributed immediately, if appropriate.
- Program Temporary Fixes (PTFs)

  Program Temporary Fixes (also referred to as APARs) are developed to enhance product functionality or correct a software error.

  - Published PTFs

    PTFs are corrections to problems or product enhancements that were determined to be important enough to make available immediately. Published PTFs have been certified by QA and will be integrated into the next PSP. Published PTFs are available for download at http://ca.com/support, or can be requested by contacting CA Technical Support.

  - Test PTFs

    Occasionally, clients will report a problem where an official fix has not yet been developed. In this case, a Test PTF will be developed to address the reported problem. Test PTFs have not been certified by QA and are only available by contacting CA Technical Support.

Note: The urgency of a PIB or a PTF is indicated by its HYPER classification as follows:

HYPER NO	The PIB or PTF is of normal importance.
HYPER YES	The PIB or PTF is of high importance.
HYPER FLASH	The PIB or PTF is of high importance and has critical impact. These bulletins and fixes are automatically mailed to you.

All of the hyper categories indicated above are available at http://ca.com/support, or can be requested by contacting CA Technical Support. If you register for hyper notification (Hyper Solution Delivery) using the CA website at http://ca.com/support, you will automatically receive any PIBs or PTFs classified as HYPER YES or HYPER FLASH by email.

3.2.2 User Change Control Procedure

Changes applied to the CA MICS system must be strictly controlled. This is essential if the chance of making an error in applying a modification is to be minimized.

Strict control involves constant attention to several matters:

- Is it convenient and cost-effective to make the intended modification, considering both time expended and the run-time resources necessary?
- Will the modification increase your dependence on a particular maintenance level of CA MICS?
- Would the intended modification interfere with some other CA MICS process, or with the process of another system?
- How can information on the intended modification be gathered, documented, and disseminated?

3.2.2.1 Identification

The goal of change control procedures is to ensure a tested modification that produces the desired results. The desired results are usually improvements in the performance or application of CA MICS at your site, or the introduction of some new input source into the CA MICS database.

Source of Data

If the modification involves treating a new data source, document the source. Some examples of new data sources are user-generated appendages to SMF records, entire special purpose SMF records, or manually-generated operations data. Regardless of the source, documentation is essential to the success of CA MICS modifications that use the inputs.

The documentation should include:

- The source of the input, such as the SMF log, or a data entry function.
- The format of each type of record, including the location and SAS INFORMAT specifications for each data element in each record type.
- The availability of the input source, including the time of day the data can be input to a CA MICS update cycle, and what corporate entity is responsible for the integrity of the input.

Database Output

If the modification involves adding new data elements or whole files to the CA MICS database, the output of the daily update cycle must be documented. The examples noted above could cause data elements to be added to existing CA MICS files, or could cause entire user components to be added to CA MICS.

The documentation should include:

- The list of files to be modified, if the files currently exist.

- The description of files to be added to the CA MICS database, and the structure of the user component to which these files will belong.

- The list of data elements added by the process, including the SAS data element length and output format of each.

Tangible Output

Usually a modification involves the production of some form of tangible output, such as a report or an extracted output file to be used as input to some further process. The examples noted above might enable resource utilization or billing reports to be created using extended accounting information, or may enable site-unique exceptions to be added to the CA MICS administrative Information Area.

The documentation should include:

- The list of existing reports into which the new material must be integrated, and considerations for doing the integration.

- The description of new reports or other CA MICS output that will contain information from the new source, including report/record content descriptions.

Benefit Analysis

If a modification involves improvement in the performance or application of CA MICS, the benefits of such a modification should be examined. For example, some extension of the normal CA MICS input checkpointing mechanism might inform the computer operator of a possible data loss condition. Such a modification should be evaluated for benefits: Does the operator communication serve the useful purpose of helping the operator locate missing CA MICS input? Or would such communication merely slow down the daily update process with a needless operator message with every update?

The documentation should include:

- Identification of the CA MICS user or CA MICS support person who would benefit.

- A description of the benefit.

- The list of CA MICS operational considerations necessary to use the new procedure.

Impact Analysis

Any modification involves actually changing CA MICS code or adding code to CA MICS. A complete list of CA MICS modules affected must be made and evaluated.

Some CA MICS modules lend themselves to modification more readily than others, and some CA MICS modules should not be modified at all. For example, if you are considering modifying the CA MICS JCL Generator, there is generally a way to modify the CA MICS prototype library instead to produce the desired result.

The documentation should include:

- The identification of the CA MICS module or functional area to be modified.

- The description of the modification.

- The list of CA MICS operational dependencies seen at this stage. That is, if an update were made in module A, then module B must be changed to accommodate the modification.

3.2.2.2 Authorization

It is the responsibility of the user to integrate changes into CA MICS. This applies to vendor-supplied maintenance and user-generated modifications.

The final responsibility at most CA MICS installations for authorizing the production implementation of CA MICS changes rests with the CA MICS System Administrator, or some member of the organization's management staff at an even higher level.

Before arranging such authorization, you should ascertain that:

- The changes will not affect the operation of CA MICS, except in the documented way.

- The changes will not affect the content of CA MICS data, except in the documented way.

- The changes will not increase the dependence of your site on a particular CA MICS or operating system release to a level that you think is unreasonable.

The integrity of the CA MICS database is the prime consideration. That, after all, is a major reason behind the acquisition of the product. The criticality of the integrity of data in the CA MICS database is self-evident, especially if the CA MICS data is used in chargeback applications. Lost data can mean lost revenue.

The CA MICS Product Support Group is available to present objective arguments you may use in formulating the decision to implement or not implement a change. Any negative response from the CA MICS Product Support Group should be considered carefully.

3.2.2.3 Design and Review

The authorization phase of modification development includes an implied feasibility study. Having done that groundwork, the next step is the design of the modification and a thorough review of the design.

General design of the modification is usually a very short process. This is true because most modifications to CA MICS are small in nature, even though the impact of the intended change may be great.

The most important factor in general design of a CA MICS modification is identifying all the areas affected by the change. For example, adding an element to one file of the CA MICS database may involve coding a piece of CA MICS file exit logic to construct contents of the data element. This, in turn, may involve additional record selection and data element verification logic. It may also require that additional records be passed through the intermediate files, and that the intermediate data sets be resized.

The general design of larger projects, such as adding user-written components, requires more effort. Besides the "ripple" effect noted above, an entire spectrum of design must be done. This covers the input data through the output reports and database, and identification of exception conditions.

The general design should be reviewed before detail design takes place. The CA MICS Product Support Group is well able to comment on summaries of such general designs. Also, user involvement at this stage in modification development can be valuable in terms of short-term design refinement and long-term acceptance of CA MICS (if user involvement is needed for the change).

Detail design of the modification should be done next. Detail design involves, at a minimum:

- Identifying each point in all modules to be modified.
- Producing logic diagrams or pseudocode for each modification point.
- Producing a code implementation plan, defining the order in which module modifications have to be made.
- Producing a test plan. This defines what debugging tools will be used, what debugging output must be examined to determine if the modification is working, and what checking must be done to determine that the modification is not interfering with normal system operation.

3.2.2.4 Coding and Testing

Having an implementation and test plan is prerequisite to coding the user modification. The implementation and test plans should be followed and reviewed during testing to ensure that testing is adequate.

Certain coding conventions should be followed in coding user modifications to CA MICS. Besides the guidelines presented in each section of this manual, chapter 5 of this manual gives certain standards for CA MICS modification coding and SAS coding techniques. That chapter contains performance notes and structure guidelines, as well.

It is important to have documented the design of the modification. It is absolutely critical for long-term maintenance considerations to have accurate and complete documentation of changes that were actually coded. This comes from entries in the CA MICS System Administrator Guide, and from comments in the code provided.

The normal tendency in actually entering code is to use a standard TSO-compatible editor. Such direct changes are not compatible with the application of standard CA MICS maintenance. ALL MODIFICATION CODING MUST BE DONE IN IEBUPDTE FORMAT TO PROVIDE COMPATIBILITY OF YOUR CODE WITH STANDARD CA MICS MAINTENANCE. Use the LOCALMOD.CNTL library provided in your CA MICS complex for this purpose.

Testing should be done on the Test database unit. This database unit type provides all the facilities necessary to perform unit and parallel testing, interference testing, and separation of the tested logic from production libraries.

Coding should be applied to the libraries in the CA MICS Test database unit. Applying changes to CA MICS sharedprefix libraries will enable the modification immediately, and may cause an interruption in production CA MICS database units or production database integrity problems.

3.2.2.5 Implementation

Implementation of the modification may be done after successful testing on the CA MICS TEST database unit. The nature of the modification dictates the method of implementation.

It is important to have an integration plan. The modules changed and their locations must be listed, and the module movement order documented.

Tested, modified modules may be moved back to the CA MICS sharedprefix libraries, if such is the nature of the modification. All affected modules must be moved at once, and the change will be enabled in the next run of a production CA MICS database unit.

If modules on database unit-oriented libraries have been modified, they must be moved to all applicable database unit libraries. If those changes accompany changes at the complex level, all the database unit changes must be activated at once. If they do not accompany such changes, the database unit modules may be moved one database unit at a time, or all at once.

After implementation, each database unit in the complex should be reviewed. The first DAILY update cycle of all database units to run after the module moves should be examined. Check the SAS logs of each step of each update output for unexpected messages. List a few observations from each database unit's database for every file that could have possibly been affected by the modification, and examine the contents of the data elements for correctness.

3.2.2.6 Notification and Documentation

The modifications implemented in the previous section may affect CA MICS complex operation, or the visible output of CA MICS database units, or both. Often a modification may involve adding some data element to the CA MICS database as a response to a user request. Users may begin to use the data element as soon as the modification has been integrated. Occasionally, a modification may affect the record selection algorithm to such a degree that modifications in input preparation procedures must be made.

Either of the cases above requires some communication on the part of the CA MICS System Administrator to some other group in the organization. Such notification of a CA MICS modification is usually better received if made in advance. That is, announce the day the implementation will be done with several days "warning," so that changes in operation or reporting can be made on a timely basis.

The CA MICS System Administrator Guide should be the focus of documenting the modification. The documentation format and content used to disseminate information on the change within the organization depends on the nature of the change. However, the format and content of change documentation in the System Administrator Guide is constant.

The System Administrator Guide's chapter 4, User Modification Descriptions, contains blank User Modification Description log sheets. These sheets provide a convenient skeleton for documenting the minimum amount of necessary information on user-developed changes. The information requested on this form is useful and necessary, but may actually be a small part of the complete documentation picture.

Some of the identifiers on the User Modification Description form are specifically designed to refer to a user-oriented filing system. The documentation and working notes generated during approval and design phases of the modification development can be saved. Such information typically fits in a small packet, and can be filed directly in the System Administrator Guide. An attractive alternative filing method is to use the user modification number to index a complete file on the modification. Such a file could easily contain not only the design material, but also code listings (before and after), IEBUPDTE listings, and test results. This scheme has advantages in being able to reconstruct the implementation process.

3.2.3 Change Control Test Database

CA MICS database units can be defined as a particular processing type, where the type of database unit describes the function and structure of the unit itself.

A UNIT database unit is a production database unit that performs update and reporting cycles on a regular basis. The PRIMARY database is a UNIT database which also performs some functions for complex-level data sets. A SPECIAL database unit is used for special, one-time analyses of selected input sources, especially in backloading.

The TEST database unit has all the facilities of a UNIT database. Its specific design target is to provide the facilities needed to test modifications to CA MICS before the modification is put into production.

It is absolutely necessary that user modifications be tested in a TEST database unit. Any modification can be an unsuspected liability to CA MICS database integrity. Conscientious use of the TEST database in testing user modifications, as described in this manual, can yield reliable modifications that produce the intended results with minimal production database vulnerability.

3.2.3.1 Purpose

The TEST database unit exists to test modifications to CA MICS before such modifications are put into production. That is the sole purpose for which this type of database unit is designed.

The TEST database unit is fully capable of running in a production mode. Production processing is NOT the purpose of a TEST database unit, but the ability to operate in such a mode is necessary to provide comprehensive testing facilities.

The TEST database unit has full database functions. If generated, all timespans are maintained. Modifications to any file in any timespan can be tested, printed, deleted, and retested without affecting any other database unit.

All archiving functions are supported. The effect of enabling or disabling file audit or history archiving can be quickly and easily tested, using manufactured data or data copied from any other database unit.

Disjoint source files are maintained. Modifications that must be made to complex-level source files may be tested on the TEST database unit without affecting any other database unit or the target complex libraries. The following special libraries exist for that purpose, and are concatenated in front of their complex-level counterparts for testing in the TEST unit:

```
sharedprefix.MICS.TEST.INCLLIB
sharedprefix.MICS.TEST.LOAD
sharedprefix.MICS.TEST.SOURCE
```

Another advantage of the TEST database unit is that, if it is used properly (not in production), it can be deleted during periods of no modification activity, and can be reinstalled when needed.

3.2.3.2 Definition and Generation

The TEST database unit definition and generation is accomplished through the standard CA MICS installation checklist. Since the TEST database is installed as a standard database unit type, such a unit can be installed through normal methods.

3.2.3.3 Controlled Input Data Sources

One advantage of a TEST database unit is the ability to test a database and logic configuration with a controlled amount of test data. It is possible to select and process data for testing, and keep the data entirely separate from any production CA MICS database unit.

The term selection usually brings to mind the concept of discarding unwanted data. A larger view of data selection is to pick data for a specific purpose. In the TEST database unit, such selection has great benefit. Test data that represents usual processing can be kept on the TEST unit. It is also to the user's advantage to search out data that represents unusual events and monitoring conditions. It may not be possible, for example, to find the sequence of test data that represents the SMF log contents during a full JES orderly shutdown.

If such a case is noted in a production process, and the data might be valuable to modification testing sometime in the future, the data can be saved on the TEST database unit. This may be on a DASD data set associated with the TEST database unit's prefix, or catalogued on tape through the TEST unit's tapeprefix.

It is also possible to group data in the TEST unit into meaningful intervals of time. It is often useful to have a small amount of data, say one hour's worth, available to test a change. If the change has a problem, a lot of processing time need not be expended in error detection.

Volume test data can be catalogued and saved on tape. Such data may be necessary for benchmarking possible optimized code or operational changes.

End of month and end of year test data can also be difficult to locate. As such conditions arise, or other variations from normal time stamp contents, the data may be saved through the TEST database unit. Examples of other such periods are the transition to and from daylight saving time, and February 29th of a leap year.

Creative test data management can greatly reduce the effort involved in testing and verifying a modification to CA MICS.

Chapter 4: CA MICS Facilities

A CA MICS system includes several facilities that ease the installation tasks and help provide an optimum execution environment. These facilities provide powerful tools for both standard and advanced installation tailoring of a CA MICS system.

Using these facilities as described will allow you to perform a broad range of modifications to CA MICS in a way that will cause minimal difficulty in applying maintenance and in moving to new releases of the product.

This section contains the following topics:

4.1 Overview (see page 51)
4.2 CA MICS Component Generator (MCG) (see page 54)
4.3 User Exit Facilities (see page 151)
4.4 SAS Functions and Call Routines (see page 181)
4.5 User-defined Informats and Formats (see page 184)
4.6 Documentation Programs (see page 199)
4.7 Calendar Derivations (see page 219)
4.8 Account Code Determination Routine Coding (see page 258)

4.1 Overview

CA MICS has been designed to accept a wide range of data sources in a number of information areas. No single user installation may ever use all the functions built into CA MICS, and the likelihood of any two CA MICS installations being identical is very small.

CA MICS installations can be tailored to individual sites by a variety of standard means, through execution-time or installation-time CA MICS parameters. This "tailoring" answers questions such as:

- Is JES2 or JES3 data to be kept by CA MICS, or are both types to be kept?
- What type of CICS monitor is used?
- What is the description of each processor that CA MICS will process?

Documentation for such standard specifications appears in the PIOM, as well as in the individual data integration product guides.

There is another level of tailoring applicable to CA MICS. Installations have the option of deciding to make some modification to the data source itself (such as changing the content of a particular SMF record type). Also, there are data elements that may not be used by CA MICS, even though they appear in the data sources, because such use is not generally applicable to the user community and creates levels of complication that are distracting to the normal CA MICS installation. This level of tailoring involves more than just parameter specifications. Some modification of the standard CA MICS process must be done.

CA MICS provides a flexible approach to modification. A considerable amount of CA MICS code and its environment is generated. This allows user modifications to be inserted by changing as few lines as possible of existing CA MICS code to perform the intended function.

This section discusses facilities for CA MICS installation and user modification. The purpose of each facility is outlined here. Subsequent sections discuss each facility in detail.

CA MICS Component Generator (MCG)

The MCG accepts encoded descriptions of components and their associated information areas, files, and data elements within each file. For example, it produces SAS code and other material that builds files, summarizes data elements, and ages files. Typical user modifications that require a knowledge of MCG operation include adding a data element to a file or activating a file in a normally inactive timespan.

CA MICS JCL Generator (JCLGEN)

The JCLGEN accepts encoded descriptions of JCL. It produces executable JCL library members. Typical user modifications that require a knowledge of JCLGEN operation include changing a data set name from the standard CA MICS convention.

User Exit Facilities

The User Exit Facilities provide points for easy insertion of user SAS code in frequently modified parts of CA MICS logic. Typical user modifications that require a knowledge of user exit facility operation include extending input data selection criteria or implementing data element content changes at the detail level.

CA MICS Information Center Facility (MICF)

MICF is the online CA MICS inquiry and documentation tool. MICF provides efficient access to CA MICS data and documentation by experienced CA MICS users and provides useful tutorial functions for novices. Typical user modifications that require a knowledge of MICF operation include integrating MICF into user SPF dialogs or rewording MICF screens or messages to accommodate unusual requirements (like translating some messages into a language other than English).

CA MICS Utilities

The CA MICS utilities include print formatting, cross reference, and library maintenance programs. Typical user modifications that require a knowledge of CA MICS utilities operation include modifying the DOCUMENT program to an alternate document page structure or creating versions of the cross reference utilities for special purposes.

CA MICS Functions

The CA MICS functions include common database control functions, calendar data element derivations, and CA MICS standard formats. Typical user modifications that require a knowledge of CA MICS functions operation include modifying the content and use of common data elements, such as MONTH, or adding special user information into standard CA MICS status reports.

4.2 CA MICS Component Generator (MCG)

The CA MICS Component Generator provides the following important capabilities:

- as described in Chapter 6 of this guide, the MCG simplifies database tailoring
- as described in Chapter 7 of this guide, the MCG simplifies writing a user component
- as described in Chapter 8 of this guide, the MCG facilitates integration of NSM data into the CA MICS database

The CA MICS Component Generator (MCG) minimizes the efforts in initial definition, SAS coding, MICF coding, JCL coding, and system integration tasks involved with the design and implementation of a database component. Figure 4-1 illustrates the operational flow.

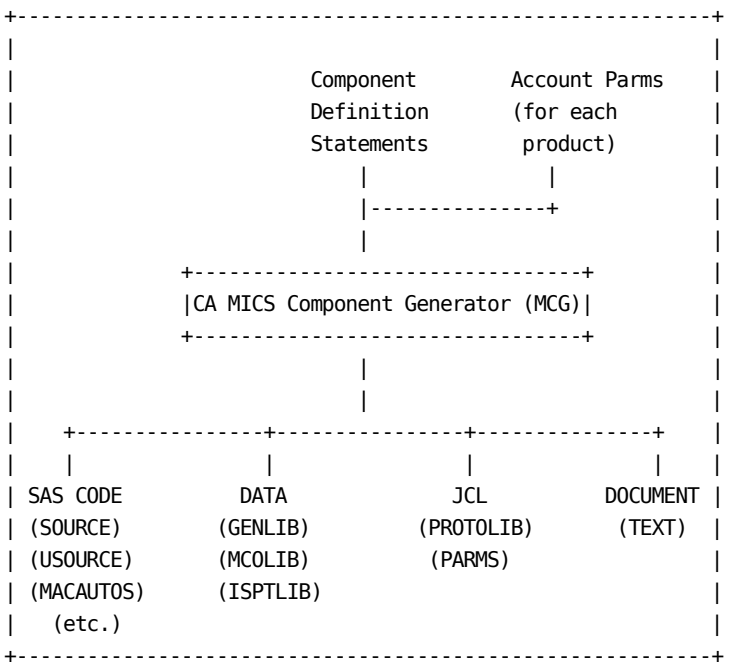


Figure 4-1. MCG Operational Flow

All of the required SAS, MICF, JCL, and documentation members for a component are produced, but additional user code (such as the detail format routine DYcccFMT) must be provided using predetermined MACRO definitions.

4.2.1 Generation Dependencies

The MCG is one of several generation processes that are standard in a CA MICS system. Consequently, it is important that the dependencies between the MCG and the other generators be understood.

The five types of generations that take place in CA MICS are divided into two categories:

Database Complex Generations

- CA MICS JCL Generation (JCLGEN) for complex-wide jobs and batch PROCs
- CA MICS Component Generation (MCG) for individual components. These jobs are identified by the jobname cccCGEN, where ccc is the component identification (e.g., SMF).

  These jobs are submitted from the sharedprefix.MICS.CNTL library.

Database Unit Generations

- CA MICS Parameter Generation for individual database unit operational parameters that use the database unit parameter library as input. These jobs are identified by the jobname cccPGEN, where ccc is the component identification (e.g., SMF).
- CA MICS JCL Generation (JCLGEN) for individual database unit jobs and PROCs (JCLGENA through JCLGENH).
- CA MICS Database Cycle Generation for individual database unit data retention limits (CYCLES and archive limits). This job is named CYCLEGEN.

  These jobs are submitted from the prefix.MICS.CNTL library.

Implications

The CA MICS Component Generator (MCG) generates JCL PROTOLIB members that are used as input to the JCL generations for database units. Therefore, any component generation changes MUST BE accompanied by JCL generations for component or file definition-sensitive JCL (JCLGENB-JCLGEND) and cycle generations (CYCLEGEN). These generations must be performed for each database unit in which the component is defined.

Because the component generator produces complex-wide outputs, the definitions of all components in the generation library apply to ALL DATABASE UNITS in which the generated component is installed.

4.2.2 Generated Outputs

Figure 4-1 above illustrated that the MCG produced outputs for a number of different CA MICS libraries. A maintenance report is also produced: the Generation Definition Statements Report, which displays the statements that have been defined. If any errors were detected in editing the statements, error messages appear immediately after the statement's display or at the end of the entire list.

The MCG produces most of the required SAS, MICF, JCL, and documentation members for a component. Additional user code (for example, for the detail format routine DYcccFMT) must be provided to complete the component. The following list describes the members that are produced by the MCG. Figure 4-3 provides a cross-reference between the generated member and the MCG member that produced it.

The conventions used in naming the different types of members are:

ccc - Three-character component identifier
iii - Three-character information area identifier
fff - Three-character file identifier
sss - Three-character component job step number.

CA MICS Member Overview

The following is a brief description of the purpose of each of the SAS, PROTOLIB, and document members involved in a component implementation.

```
DATABASE TAILORING

1. SAS Members - 'sharedprefix.MICS.SOURCE'

   $cccCYC1:  Test Aging and Delete Cycle Macros

              Contains macros defining the file deletes and
              PROC DATASET aging statement for cycle
              maintenance in the MICSGT update test process.

   #cccSFD    Selective File Definition Macros
    and
   $cccSFD:   Contains one macro for each file, defining the
              file as active or inactive (fffSFD).

   cccCHECK:  Database Check Routine

              Contains executable code to print the first 10
              observations of all files in the DETAIL and
              TABLES timespans and to perform a TABLES list
              frequency distribution (PROC FREQ) according to
              the defined sequence pattern for all files in
              DAYS, WEEKS, MONTHS, and YEARS timespans.

   DYcccAGE   Daily File Aging Routine
     and
   DYcccRGE:  Contains executable code to update the
              Checkpoint and Input History Log Files for this
              component's input data.  It also performs
              database cycle maintenance daily by aging for
              DETAIL and DAYS files, and WEEKS week-to-date
              and MONTHS month-to-date files using the
              generated macros in the
              prefix.MICS.USER.SOURCE($cccCYCS) member.

   DYcccSUM:  Daily Summarization/Update Routine

              Contains executable code to insert the new file
              in the DAYS timespan, and to update the
              week-to-date and month-to-date files in the
              WEEKS and MONTHS timespans respectively for
              the files supported in the applicable
              timespans.
```

```
DYcccIUA:  Incremental Update File Aging Routine

           Contains executable code to update the
           incremental update Checkpoint and Input History
           Log Files for this component's input data.  It
           also performs incremental update database cycle
           aging for DETAIL and DAYS files using the
           generated macros in the
           prefix.MICS.USER.SOURCE($cccCYCS) member.

DYcccIUR:  Incremental Update Daily File Roll-up Routine

           Contains executable code to copy the "today
           to-date" incremental update DETAIL and DAYS
           level files to the appropriate unit database
           DETAIL and DAYS timespan data sets.  This
           processing is executed during the DAILY job to
           "roll-up" the incremental update database to
           the production unit database files.

DYcccIUS:  Incremental Update Summarization/Update Routine

           Contains executable code to combine DETAIL and
           DAYS level files from the current incremental
           update with the "today to-date" files resulting
           from today's prior incremental update
           executions for this product.

WKcccAGE:  Weekly File Aging Routine

           Contains executable code to update the
           Checkpoint File for this component's weekly
           update.  It also performs database cycle
           maintenance weekly by aging the files supported
           in the WEEKS timespan using the generated
           macros in the #cccCYCS member.

WKcccAUD:  Weekly Audit Archive Routine

           Contains executable code to update the archive
           weekly audit files for all files supported in
           the DETAIL or DAYS timespan and defined as
           eligible for the weekly audit archive process.
```

```
WKcccHST:  Weekly History Archive Routine

           Contains executable code to update the archive
           weekly history files for all files supported in
           the WEEKS timespan and defined as eligible for
           the weekly history archive process.

MNcccAGE:  Monthly File Aging Routine

           Contains executable code to update the
           Checkpoint File for this component's monthly
           update.  It also performs database cycle
           maintenance monthly by aging the files
           supported in the MONTHS timespan using the
           generated macros in the #cccCYCS member.

MNcccHST:  Monthly History Archive Routine

           Contains executable code to update the archive
           monthly history files for all files supported
           in the MONTHS timespan and defined as eligible
           for the monthly history archive process.

YRcccAGE:  Yearly File Aging Routine

           Contains executable code to update the
           Checkpoint File for this component's yearly
           update.  It also performs database cycle
           maintenance yearly by aging the files supported
           in the YEARS timespan using the generated
           macros in the #cccCYCS member.

NOTE:      The next five members are maintained for
           compatibility with old-style SAS macros.  The
           old (SAS 79-style) macros defined in this
           member resolve to the new (%MACRO) macros
           described later.  The generated members are:
           #cccFLS, #cccKEEP, #cccLBLS, #cccSEQ, and
           #cccSUM.

#cccFLS:   Element LENGTH, FORMAT, and INFORMAT Macros

           Contains three macros for each file defining
           the LENGTHs, FORMATs, and INFORMATs for all
           elements in the file, with unique definitions
           for the DETAIL (_fffFD), DAYS/WEEKS (_fffFDD),
           and MONTHS/YEARS (_fffFDM) timespans.
```

#cccKEEP: File KEEP Statement Macros

Contains up to six macros for each file defining the elements to be kept by the KEEP statement in the DETAIL (_fffKPDT), DAYS (_fffKPDY), WEEKS (_fffKPWK), MONTHS (_fffKPMN), YEARS (_fffKPYR), and TABLES (_fffKPTB) timespans.

#cccLBLS: File LABEL Statement Macros

Contains one macro for each file defining all the data elements' 40-character labels (_fffLBLS).

#cccSEQ: File Sequence Definition Macros

Contains up to six macros for each file defining the file's unique sequence pattern for the DETAIL (_fffSEQX), DAYS (_fffSEQD), WEEKS (_fffSEQW), MONTHS (_fffSEQM), YEARS (_fffSEQY), and TABLES (_fffSEQT) timespans.

#cccSUM: File Summarization and Computation Macros

Contains two macros for each file defining: (1) the processing logic for summarizing the data in the file to a higher level of granularity (_fffSUM), and (2) SAS expressions for all computed elements (_fffVCE).

2. SAS Members - 'sharedprefix.MICS.INCLLIB'

DAYsss: %INCLUDE List for Daily DB Update

Contains executable code to include all of the CA MICS SAS macro and executable code members required for the daily database update step.

WEEKsss: %INCLUDE List for Weekly DB Update

Contains executable code to include all of the CA MICS SAS macro and executable code members required for the weekly database update step.

MONTHsss: %INCLUDE List for Monthly DB Update

Contains executable code to include all of the CA MICS SAS macro and executable code members required for the monthly database update step.

YEARsss: %INCLUDE List for Yearly DB Update

Contains executable code to include all of the CA MICS SAS macro and executable code members required for the yearly database update step.

cccGTEST: %INCLUDE List for Testing Online Database

Contains executable code to include all of the CA MICS SAS macro and executable code members required to test all online database update and reporting functions. This %INCLUDE list is used by the MICSGT generation and test procedure.

3. JCL PROTOLIB Members - 'sharedprefix.MICS.PROTOLIB'

DYcccsss: Step JCL for Daily Job

Contains PROTOLIB JCL that, when generated by the MJG, produces the step JCL for this component that is used in a database unit's daily update job.

WKcccsss: Step JCL for Weekly Job

Contains PROTOLIB JCL that, when generated by the MJG, produces the step JCL for this component that is used in a database unit's weekly update job.

MNcccsss: Step JCL for Monthly Job

Contains PROTOLIB JCL that, when generated by the MJG, produces the step JCL for this component that is used in a database unit's monthly update job.

YRcccsss: Step JCL for Yearly Job

Contains PROTOLIB JCL that, when generated by the MJG, produces the step JCL for this component that is used in a database unit's yearly update job.

WKccc300: Step JCL Portion for Weekly Archive

Contains PROTOLIB JCL that, when generated by the MJG, produces JUST the input and output JCL data definition statements (DD statements) used to update the audit and weekly history archive files (if not being generated in the optional stand-alone AUDIT and/or HISTW jobs).

MNccc300: Step JCL Portion for Monthly Archive

Contains PROTOLIB JCL that, when generated by the MJG, produces JUST the input and output JCL data definition statements (DD statements) used to update the monthly history archive files (if not being generated in the optional stand-alone HISTM job).

AUDIT: JCL for Stand-Alone Weekly Audit Archive

Optionally contains PROTOLIB JCL that, when generated by the MJG, produces JUST the input and output JCL data definition statements (DD statements) used to update the audit archive files.

HISTW: JCL for Stand-Alone Weekly History Archive

Optionally contains PROTOLIB JCL that, when generated by the MJG, produces JUST the input and output JCL data definition statements (DD statements) used to update the audit weekly history files.

HISTM: JCL for Stand-Alone Monthly History Archive

Optionally contains PROTOLIB JCL that, when generated by the MJG, produces JUST the input and output JCL data definition statements (DD statements) used to update the audit monthly history files.

4. Data Members - 'sharedprefix.MICS.GENLIB'

   cccFCT: File Definition Table (FCT)

            Contains data that defines each file and the
            timespans in which they are supported for the
            component. This member is used to support the
            CA MICS Information Center Facility (MICF) with
            database complex configuration information.

5. Documentation Members - 'sharedprefix.MICS.INC.TEXT'

   #INSfffE: Data Element List

            Contains text for a single file; lists the data
            elements supported. This documentation member
            is stored in the INC.TEXT library and is used
            in the product guides.

   #INSfffS: File Organization Box

            Contains text for a single file; lists the
            sequence/summarization pattern for this file in
            each timespan in which the file is supported.
            This documentation member is stored in the
            INC.TEXT library and is used in the product
            guides.

   #INScccF: General File Matrix Chart

            Contains text for all files in the component,
            listing each file and the timespans in which
            it is supported. This documentation member is
            stored in the INC.TEXT library and is used in
            the product guides.

6. SAS Autocall Macros - 'sharedprefix.MICS.MACAUTOS'

   cccACCT: Component Account Code Definition Macro

            Contains the SAS LENGTHs, FORMATs, and LABELs
            of the account code data elements defined for
            this component.

   cccC: Component Description Macro

            Contains a list of all files in the component,
            the information areas to which they belong,
            and the essential and derived options used in
            their generation.

```
cccCLEN:   Component Element Length Macros

           Contains SAS length descriptors for the common
           data elements used by files in the component.

fffDERV:   File Derived Element Computation Macros

           Contains SAS statements necessary to calculate
           every derived data element in the file.

fffFMT:    File Format and Informat Macros

           Contains SAS format and informat descriptors
           for each data element in the file, in all
           timespans.

fffKEEP:   File Keep Macros

           Contains the data element KEEP list for the
           file in all timespans.

fffLEN:    File Length Macros

           Contains SAS length descriptors for each data
           element in the file, in all timespans.

fffLBL:    File Label Macros

           Contains SAS label statements for each data
           element in the file in any timespan.

fffSEQ:    File Sequence Macros

           Contains the list of data elements, by
           timespan, that describe the key structure and
           summarization break of the file.

fffSUM:    File Summarization Macros

           Contains the SAS statements necessary to
           summarize the file into a higher timespan.
           Derived data elements are recalculated by
           the fffDERV macro, referenced by the fffSUM
           macros.
```

```
7.  Data members -

    sharedprefix.MICS.ISPTLIB

              These members are ISPF tables used in support
              of the CA MICS online applications and end-user
              facilities.

    sharedprefix.MICS.MCOLIB

              These SAS files and SAS views support CA MICS
              generation and operational processes and also
              provide information to the CA MICS end-user
              applications.


COMPONENT SHELL GENERATION - ONE TIME ONLY EXECUTION
1. SAS Members - 'sharedprefix.MICS.SOURCE'

   $cccMSTR:  Master Options

              Contains macros that define component options
              or special flags.  The member is initially
              empty following a shell generation.

   #cccEXIT:  User Exit Macros

              Contains one macro for each file (_USRSfff)
              that provides a CA MICS exit (Null SAS Macro)
              immediately before the file is written in the
              daily format routine.

   DYcccFMT:  Daily Input Format Routine
```

```
NOTE:       Items followed by an asterisk contain
            executable code, which must be completed by the
            user, to perform the following functions:

            o  Input raw input data (*)
            o  Edit raw input data (*)
            o  Verify input data against checkpoint
               low/high dupdate ranges for duplicate data
               test
            o  Use checkpoint processing facilities for
               NODATE, SELECT, and/or FORCE options
            o  Insert files in the DETAIL timespan
            o  Support Selective File Definition (SFDs)
            o  Support User File Exits
            o  Update Checkpoint File low/high update
               ranges
            o  Update Input History Log File with update
               activity, record counts, and update ranges
            o  Sort all DETAIL files and drop duplicate
               data.

DYcccMBO:   Daily MBO Report Routine

            Contains executable code to produce this
            component's daily management objective routine.
            This member is initially empty following a
            shell generation.

WKcccMBO:   Weekly MBO Report Routine

            Contains executable code to produce this
            component's weekly management objective
            routine.  This member is initially empty
            following a shell generation.

MNcccMBO:   Monthly MBO Report Routine

            Contains executable code to produce this
            component's monthly management objective
            routine.  This member is initially empty
            following a shell generation.
```

```
   DYcccEXC:  Daily Exception Analyzer Routine

              Contains executable code, which must be
              completed by the user, to perform the exception
              analysis that has been defined for this
              component.  This member contains only standard
              DATA, SET, initialization, and termination
              code.  You must add the actual exception
              analyzers.

   cccEVA:    Exception Value Analyzer Routine

              Contains executable code, which must be
              completed by the user, to perform the exception
              value analysis for those elements used in this
              component's exception analyzer.  This member
              contains only standard DATA, SET,
              initialization, and termination code.  You must
              add the actual exception value analyzers.
2. Documentation Members - 'sharedprefix.MICS.INC.TEXT'

   PICccc:    Data Dictionary PIC List

              Contains the PIC list necessary to generate the
              data dictionary appendix for this component.

   eeeeeeee:  Data Dictionary Shell

              Contains the outline or "shell" member for the
              data dictionary member for this data element.
```

4.2.3 MCG Operation

```
The MCG is invoked through one of two standard
complex-wide procedures listed below:

    MICSGU - CA MICS Generation and Libraries Updated
    MICSGT - CA MICS Generation and Test (reports and
             test only)

Standard jobs have been generated by JCLGEN0 for the standard
CA MICS components using the MICSGU PROC. These jobs are
named cccCGEN, where ccc is the component identifier (for
example, SMFCGEN).
```

We suggest using the PROCs in the following manner:

- Use MICSGU for all functions for standard CA MICS components.
- Use MICSGT to test the operational fitness of generated user components.

Remember that whenever a MICSGU is executed, it is necessary also to execute the JCL and cycle generations for the applicable database units (reference Section 4.2.1).

COMPONENT GENERATOR EXECUTION

The component generator may be executed to generate and test a component on a temporary basis or to generate the final version of the component and load the production libraries.

A standard procedure, MICSGT, has been provided to enable the user to generate and test a user component. The only symbolic that is required to execute the procedure is &COMPT, which is used to define the component that is being generated. See Chapter 7 for more information.

COMPONENT GENERATOR EXECUTION FLOW

The component generator reads generator definition statements and, based on the options and definitions input, dynamically invokes the required generator subroutines.

Phase 1:

The component generator supervisor, GENCOM, is the first module executed and serves the purpose of the SAS INCLUDE member for the generation job. Prior to actually beginning execution, GENCOM %INCLUDEs two members, #BASMSTR and GENSASMC.

#BASMSTR is the standard CA MICS step initialization process. It dynamically builds a number of required Macros.

GENSASMC contains all of the standard MACROs used in the generation process and is not an executable module.

Phase 2:

The component generator supervisor, GENCOM, then invokes the generator initialization routine, GENINIT.

GENINIT performs a pre-pass of the generator definition statements to identify the type of generation to be performed and the component to be generated. This information, with the date and time of generation, is stored by the CA MICS Global Data Facility for reference by other generator subroutines.

Phase 3:

The component generator supervisor, GENCOM, then invokes the common element definition generator, GENSASCE.

GENSASCE processes the member $DEGENIN from GENLIB to create input to be used by the GENINPUT routine in processing common data elements. $DEGENIN contains generator definition statements which identify common data elements, their processing attributes (retained, accumulated, maximum, etc.), LENGTH and FORMAT definitions, SAS LABEL, Data Dictionary ALIAS, and alternate field name definitions.

Phase 4:

The component generator supervisor, GENCOM, then invokes the generator definition statement input parser, GENINPUT.

GENINPUT performs the following operations:

1) Creates a work file (GENFIL) which contains a record for each FILE defined.

2) Creates a work file (GENINF) which contains a record for each INFORMATION AREA defined.

3) Creates a work file (GENDED) which contains a record for each DATA ELEMENT/TIMESPAN defined.

4) Creates a work file (GENDEP) which contains a record for each DATA ELEMENT/TIMESPAN DEPENDed on by a derived data element in that timespan.

5) Creates a work file (GENCED) which contains a record for each computed EXPRESSION required to build the code for computed data elements.

6) Inputs and edits each of the generator definition statements.

7) Builds SELECT indicators which are input by GENCOM to determine which modules are to be executed.

8) Processes @@ACCT statements and dynamically invokes the account code generator. This enables GENINPUT to then substitute the actual account codes into the appropriate locations.

9) Processes @@FIRST and @@LAST statements and outputs the information to the computed expression (GENCED) file.

10) Verifies all sequence patterns to ensure that: (1) each file has at least one sequence field, (2) the sequence fields are ascending in rank and contiguous, and (3) sequences are not defined more than once within a file.

11) Verifies logical data conditions to ensure that (1) an element is not defined more than one time within a file, and (2) computed elements must have computed expressions.

Phase 5:

The component generator supervisor, GENCOM, then invokes the required generator subroutines, based on the options that have been defined, to generate the component. If the component being generated is the Base (BAS) Component, then the select indicators are set to indicate that a number of modules are not to be generated.

1) Invokes component-unique generators to create specialized SAS code for the respective component.

2) Invokes the required modules to perform GENFILES generation (File Tailoring).

Phase 6:

The component generator supervisor, GENCOM, then invokes the update mechanism, %GENUPDT, that stores generated members into CA MICS libraries.

4.2.4 Generator Definition Statements

Components are defined using several types of generator definition statements, as discussed below. These statements are located in several members of sharedprefix.MICS.GENLIB, the generation library.

Distribution material includes one member in the GENLIB library for each CA MICS component licensed. The members are named in the form cccGENIN, where ccc is the three letter component identifier. The specific members on the library may include:

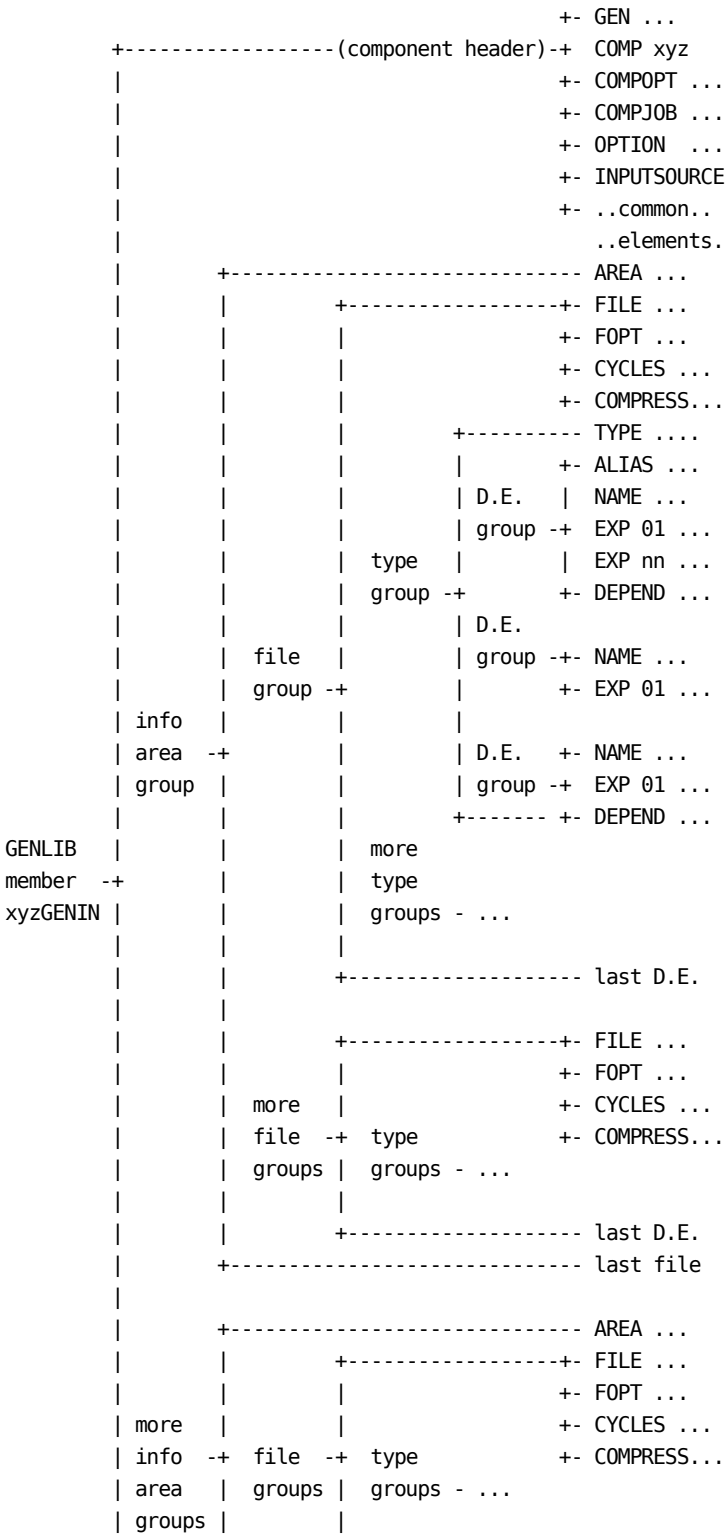
Member Name	Component Described
BASGENIN	CA MICS Platform
RMFGENIN	Hardware and SCP Analyzer
SMFGENIN	Batch and Operations Analyzer
cccGENIN	ccc CA MICS component (where ccc is the three letter component identifier. An example is CICGENIN, which is the member containing definition statements for CICS Analyzer.)
UxxGENIN	User-Written Component
ZxxGENIN	User-Written Component (where one member is coded for each user-written component installed.)

Common data elements are elements that are used by more than one CA MICS file. If a common element is used by files in more than one product, the common element is defined in the special generation library member called $DEGENIN. If a common element is used by files in a single product, the common element is defined in that product's cccGENIN member prior to the first AREA statement.

```
Components are defined through component definition
statements that contain the following keywords:

    GEN:          Defines the generation process to be
                  performed.

    OPTION:       Defines the data element clusters to be
                  included in the generation.

    COMP:         Defines the component name, identifier, job
                  step number, and input file attributes.

    COMPOPT:      Defines the component-specific Essential and
                  Derived Data Element options.

    COMPJOB:      Defines the operational JOBs that will
                  contain a component-specific step.

    INPUTSOURCE:  Defines the component's input source and the
                  record types that will be processed by the
                  component.

    common        Defines elements that are available to more
    elements:     than one of this component's files.  These
                  elements must appear prior to the first AREA
                  statement.

    AREA:         Defines the information area within which a
                  file or set of files belongs.

Files are defined through file definition statements that
contain the following keywords:

    FILE:         Defines a file's name, identifier,
                  timespans supported, and archive levels
                  supported.

    FOPT:         Defines the file's specific Essential and
                  Derived Data Element options.

    CYCLES:       Defines a file's default database cycle
                  retention values.
```

```
    COMPRESS:    Defines whether or not the file is eligible
                 to be compressed when the compression
                 facility is activated.

Standard and common data elements are defined through
standard and common data element definition statements that
contain the following keywords:

    TYPE:        Defines a data element's type, file sequence
                 position, length, and format.

    ALIAS:       Defines a data element's data dictionary
                 alias.

    ALTNAME:     Defines a common data element's alternate
                 name, as saved on all files.

    NAME/NAMX:   Defines a data element's name and label and,
                 for standard data elements, the timespans
                 in which the data element exists.

    EXP:         Defines a standard data element's expression
                 to compute the element's value, if the data
                 element is a computed element (type C).

    DEPEND:      Defines dependencies between standard data
                 elements on the file.


The following terms will help you understand how these
statements relate to each other hierarchically.  It is
useful, and sometimes necessary, to refer to clusters of
Component Generator (MCG) control statements by their group
function.  The following groups are defined:

Component Group:  The set of all MCG control statements that
    define a component in the cccGENIN member.  This group
    consists of one GEN statement, an optional OPTION
    statement, one COMP statement, an optional COMPOPT
    statement, an optional COMPJOB statement, an optional
    INPUTSOURCE statement, and one or more AREA statements.

File Group:  The set of all MCG statements that define a file
    within an information area.  This group consists of one
    FILE statement, one FOPT statement, one CYCLES statement,
    an optional COMPRESS statement, and one or more Type
    Groups.
```

```
Type Group:  The set of all MCG statements that define one or
    more data elements within a file, such that all the data
    elements in the group share common summarization
    algorithms and common SAS characteristics of format and
    length in all timespans of the file.  This group
    consists of a TYPE statement followed by one or more Data
    Element Groups.

Data Element Group:  The set of all MCG statements that
    completely defines one single data element within the
    Type Group.  This group consists of different statements
    based on the data element's type:

    Derived summarization elements (type "C" or computed)
    consist of:

    o one ALIAS statement, optional
    o one NAME or NAMX statement, mandatory
    o one EXP statement, mandatory
    o additional EXP statements, optional
    o one or more DEPEND statements, optional

    Common elements (type "X") consist of:

    o one ALIAS statement, optional
    o one ALTNAME statement, optional
    o one NAME statement, mandatory

    All other summarization types consist of:

    o one ALIAS statement, optional
    o one NAME or NAMX statement, mandatory
    o one or more DEPEND statements, optional

    ** THE ORDER OF STATEMENTS IS CRITICAL **

If optional statements are used, they must appear in the
order shown in the diagram that follows.  For example, if an
OPTION statement is being used, it MUST appear after the
COMP statement.

    The statements are documented in the following sections:

    1 - Component Definition Statements
    2 - File Definition Statements
    3 - MSI Definition Statements
    4 - Standard Data Element Definition Statements
    5 - Common Data Element Definition Statements
```


+- GEN ...
(component header)-+ COMP xyz
+- COMPOPT ...
+- COMPJOB ...
+- OPTION ...
+- INPUTSOURCE
+- ..common..
..elements.
AREA ...
FILE ...
FOPT ...
CYCLES ...
COMPRESS...
TYPE ....
ALIAS ...
NAME ...
EXP 01 ...
EXP nn ...
DEPEND ...
info
area
group
file
group
type
group
D.E.
GENLIB
member
xyzGENIN
more
type
groups - ...
last D.E.
more
file
groups
last file
last D.E.

```
|       |        +------------------- last D.E.
|       +----------------------------- last file
+--------------------------------------- last area
```

4.2.4.1 Component Definition Statements

The generation definition statements used to define the component and information area to be generated and to control the MCG generation processing options are GEN, OPTION, COMP, COMPOPT, COMPJOB, INPUTSOURCE, and AREA.

The GEN statement controls the extent of component generation. Only one GEN statement is permitted in each cccGENIN member.

The optional OPTION statement defines optional processing and data element selection parameters unique to this component. User-written components typically do not use this statement. If present, only one OPTION statement is permitted per cccGENIN member.

The COMP statement defines the component. Only one COMP statement is permitted per cccGENIN member.

The optional COMPOPT statement defines the Essential and Derived data element options to be used for this component's files.

The optional COMPJOB statement defines whether or not the component is included in or excluded from operational processing.

The optional INPUTSOURCE statement defines the component's input source and the record types it processes.

The AREA statement defines the information areas within the defined component. A minimum of one AREA statement per cccGENIN member is required. Each AREA statement must be followed by one or more file statement groups. One AREA statement is coded per information area in the component. For example, the IMS Analyzer has one information area, and that information area's name is also IMS. The SCP and Hardware Analyzer, however, has two information areas, HAR and SCP.

4.2.4.1.1 Generation Type Statement (GEN)

```
This statement defines the type of generation to be
performed, allowing for a statement syntax check, a database
definition generation, or a "shell" module generation.
STATEMENT FORMAT

This statement is coded with the KEYWORD starting in
column 1, free-form, and continuing through column 72.
Continuation is not supported.  The following describes the
format and coding rules for the Generation Type Statement.

  GEN XXXXXXXXXX
         |
         |
         +---GENTYPE - Defines the type of generation to be
             performed, including:
STATEMENT OPTION DEFINITIONS

  GENTYPE - Defines the type of generation to be performed,
            with the valid options including:

            SYNTAX - Perform first-level syntax check of
              input statements.  No generation is performed.

            GENFILES - Generate system code and documentation
              for data elements and file definitions.

            GENSHELL - Generate shell documentation members
              for data dictionary and exception entries, and
              shell SAS source code for management objective
              reports, user exit member, exception analyzers,
              and exception value evaluators.  This option is
              ONLY used to build user components.  It is NOT
              to be used for CA-distributed components.
STATEMENT DEFINITION EXAMPLE

    GEN GENFILES

This generation statement is requesting a component
generation to generate system code and documentation for data
elements and file definitions.
```

4.2.4.1.2 Option Statement (OPTION)

```
The OPTION statement identifies which data element
clusters are being included or excluded when a file's data
element content is generated.  This statement is optional.
```

```
STATEMENT FORMAT

The OPTION statement is free-form and coded with KEYWORDs
in columns 1 through 72.  Continuation is not supported.  The
following describes the format and coding rules for the
OPTION statement.

  OPTION xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx xxxxxx
         |      |      |      |      |      |
         +------+------+------+------+------+
         |
         +--- KEYWORD(s)

where:

  OPTION  - Defines which data element clusters are to be
            included or excluded when generating the file's
            data element content.

  KEYWORD(s) - Defines which of the data element clusters to
               include in or exclude from the file.
               KEYWORD(s) vary by product and are listed in
               each product's guide.  All data element
               clusters that are shown for a product must be
               present.

See Chapter 7 of the appropriate product guide for OPTION
statement information.
```

4.2.4.1.3 Component Statements (COMP and COMPOPT)

```
The component statements define the component that is
being generated by assigning the required identification
name, job step number, input file attributes, and Essential
and Derived data element options.
STATEMENT FORMAT

The COMP and COMPOPT statements are always specified in a
matched pair.  There is one COMP statement per cccGENIN
member.  The COMPOPT statement may be omitted, in which case
default values for the Essential and Derived options are
taken.  If the COMPOPT statement is present, it MUST
immediately follow the COMP statement in cccGENIN.
```

The COMP statement is coded with the KEYWORD starting in column 1, free-form, and continuing through column 72. Continuation is not supported. The following describes the format and coding rules for the component statements.

```
COMP ccc sss bbbbb rrr AAAA xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
      |   |    |    |    |      |
      |   |    |    |    |      +--- COMPNAME - Name of the
      |   |    |    |    |           component.
      |   |    |    |    |
      |   |    |    |    +--- ACCOUNTING - Definition of
      |   |    |    |         component's use of account
      |   |    |    |         level data elements.
      |   |    |    |
      |   |    |    +--- RECFM - Input file record format
      |   |    |         to be used for input data.
      |   |    |
      |   |    +---BLKSIZE - Input file maximum block size
      |   |        to be used for input data.
      |   |
      |   +---STEPNUM - Job step number used to identify
      |       this component in the CA MICS jobs.
      |
      +---COMPID - Component identification.
```

The COMPOPT statement is coded with the KEYWORD starting in column 1, free-form, and continuing through column 72. Continuation is not supported. The following describes the format and coding rules for the component option statement.

```
COMPOPT eeeeeeee dddddddd xxxxxxxx
           |        |        |
           |        |        |
           |        |        +--- BYPASS - Optional
           |        |             protection bypass for all
           |        |             files in this component.
           |        |
           |        +--- DERIVED - The derived data element
           |             option to be used for all files in
           |             this component.
           |
           +------ ESSENTIAL - The essential  data  element
                   option to be used for all files in  this
                   component.
```

```
STATEMENT OPTION DEFINITIONS

  COMP     - Defines the description of the component that is
             being generated.

       COMPID      - The 3-character component identifier that
                     identifies this component (for example,
                     the CICS Analyzer's COMPID is CIC).  A
                     user component must have a 'U' or 'Z' as
                     its first letter to denote that it is a
                     user component and ensure uniqueness in
                     the component's identification.
                     Additionally, MSI cannot be used as
                     component identifier.

       STEPNUM     - The 3-character job step number that
                     identifies this component's job steps in
                     the DAILY, WEEKLY, MONTHLY, and YEARLY
                     jobs (for example, the CICS Analyzer's
                     STEPNUM is 040).  Step numbers for user
                     components should be restricted to those
                     numbers from 150 through 190.  If there
                     is a concern about eventually using all
                     of the numbers in this range, one should
                     define multiple information areas within
                     user components wherever it is possible.

       BLKSIZE     - This is an obsolete parameter which must
                     be a "." or range from 18 to 32000 for
                     compatibility.

       RECFM       - This is an obsolete parameter which must
                     be ".", F, FB, U, VB, or VBS for
                     compatibility.

       ACCOUNTING - The keywords ACCT or NOACCT define
                     whether this component has files that
                     contain account level data elements.
                     ACCT causes the ACCTGEN process to be
                     performed and is used if the data element
                     name @@ACCT appears on any file in the
                     component's definition.
```

```
                This parameter controls the
                interpretation of the @@ACCT NAME
                statement in this component's files.  If
                files contain @@ACCT NAME statements,
                specifying NOACCT here will cause @@ACCT
                to be ignored.  Specifying ACCT will
                cause @@ACCT to expand into one or more
                account code data elements.  If no @@ACCT
                names appear in any file in the
                component, this parameter is ignored.

                For user components, specify ACCT to
                generate component shells that include
                the user component interface to CA MICS
                Accounting and Chargeback.

COMPNAME    - A 1-40 character name for the component.


COMPOPT - Defines the Essential and Derived data element
     options to be used for this component's files.

ESSENTIAL   - Must take the value of DEFAULT, INITIAL,
              ONLY, or ALL.  If DEFAULT is coded, the
              component will be generated using the
              option for essential data elements set in
              sharedprefix.MICS.PARMS(CPLXDEF), which
              is activated by the CPLXGEN job.  If
              INITIAL, ONLY, or ALL is specified, files
              in the component will be generated using
              this essential option value.  See the
              FOPT statement description for more
              information.

DERIVED     - Must take the value of DISK, COMP, or
              DEFAULT.  If DEFAULT is coded, the option
              for derived data elements set in
              sharedprefix.MICS.PARMS(CPLXDEF), which
              is activated by the CPLXGEN job, will be
              used for all files in the component.  If
              COMP or DISK is specified, the files in
              the component will be generated using
              this derived option value.  See the FOPT
              statement description for more
              information.
```

```
        BYPASS     - This optional parameter is normally left
                     blank.  There are safeguards in the
                     evaluation of the Essential option that
                     help prevent inadvertent data loss.  If
                     you wish to bypass these safeguards and
                     force the Essential option for all files
                     in a component to a value of INITIAL or
                     ONLY unless overridden on the FOPT
                     statement, code the word BYPASS in this
                     position on the COMPOPT statement.
                     Otherwise, for the protection of the data
                     on an existing database, do not code
                     this parameter.
STATEMENT DEFINITION EXAMPLE


    COMP USL 150 .  .  NOACCT User SYSLOG Component
    COMPOPT DEFAULT DEFAULT

This component statement defines a user component (USL)
named User SYSLOG Component, which is to execute as step 150.
The component will not require account level data elements
and will use the default Essential and Derived data element
options set at the complex level in
    sharedprefix.MICS.PARMS(CPLXDEF).
```

4.2.4.1.4 Component Job Statement (COMPJOB)

```
The optional Component Job Statement defines the product's
exclusion or inclusion of component processing in DAILY,
WEEKLY, MONTHLY, and YEARLY jobs.
STATEMENT FORMAT

The COMPJOB statement may be omitted, in which case the
product is not included in any operational job, exception or
MBO report generation, nor archive eligibility.

If the COMPJOB statement is present, it MUST follow the
COMPOPT statement in cccGENIN.

The COMPJOB statement is coded with the KEYWORD starting in
column 1.  Parameters are free-form and can continue through
column 72.  Statement continuation is not supported.  The
following describes the format and coding rules for the
COMPJOB statement.
```

```
  COMPJOB  dwmy  edwm  awm
      |     |      |    |
      |     |      |    +-- Audit/History indicators
      |     |      |
      |     |      +-- Exceptions/MBO report indicators
      |     |
      |     +-- Job indicators
      |
      +-- COMPJOB - statement identification
STATEMENT OPTION DEFINITIONS

       COMPJOB - Define that the product be excluded from or
               included in operational jobs, exception and
               MBO report generation, and archive
               eligibility.

               dwmy - Indicators that show whether the
                      product is processed in the CA MICS
                      DAILY, WEEKLY, MONTHLY, or YEARLY jobs.
                      Code a D, W, M, Y, or inactive (.) in
                      the respective position.

               edwm - Indicators that show whether the
                      product will have exceptions generated
                      and reported, and if management
                      objective reports are to be produced by
                      the DAILY, WEEKLY, or MONTHLY CA MICS
                      reporting processes.  Code an E, D, W,
                      M, or inactive (.), respectively.

               awm  - Indicators that show whether the
                      product is eligible to create AUDIT,
                      WEEKLY History, or MONTHLY History
                      archives.  Code an A, W, M, or inactive
                      (.), respectively.
STATEMENT DEFINITION EXAMPLE

    COMPJOB DWM. EDWM .WM

These statements define a product that will be processed in
the DAILY, WEEKLY, and MONTHLY jobs.  Exceptions and MBO
reports will be generated and reported in the daily, weekly,
and monthly reporting processes.  In addition, the product is
eligible to create weekly and monthly history archives.

For more information about the COMPJOB statement, see
sections 4.3.13.1 and 4.3.13.2 in the PIOM.
```

4.2.4.1.5 Input Source Statement (INPUTSOURCE)

```
The optional input source statements define the product's
input source type and the record types it processes.
STATEMENT FORMAT

If the INPUTSOURCE statement is present, it MUST follow
the COMPJOB statement, if present, or the COMPOPT statement
in cccGENIN.

The INPUTSOURCE statement is coded starting in column 1.
Parameters are free-form and can continue through column 72.
Although statement continuation is not supported, multiple
INPUTSOURCE statements can be specified.  The following
describes the format and coding rules for the INPUTSOURCE
statement.

    INPUTSOURCE  source  rtype ...
        |           |      |
        |           |      +-- Record type
        |           |
        |           +-- Type of input source
        |
        +-- INPUTSOURCE - statement identification


STATEMENT OPTION DEFINITIONS

  INPUTSOURCE - Defines the product's input source type and
                the record types that it processes.

                source    - A value of SMF, NONSMF or MSI
                            for the first INPUTSOURCE
                            statement; a value of
                            inactive (.) for subsequent
                            statements.

               recordtype - Ranges from 0 to 255.  Multiple
                            record types may be defined on a
                            line.
                            If a value of MSI is specified
                            for the input source, record type
                            is ignored.
```

```
STATEMENT DEFINITION EXAMPLE

    INPUTSOURCE SMF 30 35 70
    INPUTSOURCE .  43 66 100
```

These statements define that the input source is SMF data and that record types 30, 35, 43, 66, 70, and 100 are supported.

4.2.4.1.6 Information Area Statement (AREA)

The Information Area statement defines an area identifying the Information Area ID and name.

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1, free-form, and continuing through column 72. Continuation is not supported. The following describes the format and coding rules for the Information Area Statement.

```
  AREA  iii xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
         |     |
         |     +--- INFONAME - Information Area Name.
         |
         +---INFOID   - Information Area identification.
```

STATEMENT OPTION DEFINITIONS

AREA - Defines an Information Area for the product.

This statement is required. If a product has only one Information Area, this Information Area ID may be the same as the component ID. The maximum number of information areas per component is nine.

INFONAME - Defines a 1-40 character name for this information area.

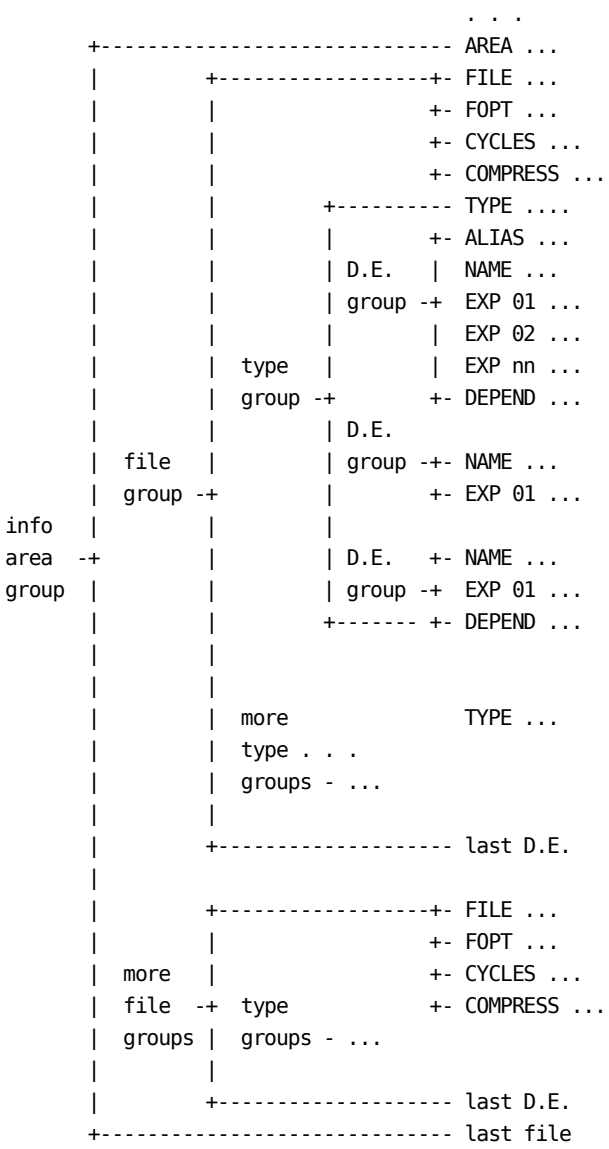
INFOID - Defines a 3-character ID for the information area identifier. A user information area must have U or Z as its first letter to denote that it is a user information area and ensure uniqueness in the information area's identification. Additionally, MSI cannot be used as in information area name.

```
STATEMENT DEFINITION EXAMPLE

    AREA UOD User SYSLOG Operations Data

This statement defines a user information area (UOD) named
User SYSLOG Operations Data.
```

4.2.4.2 File Definition Statements

```
The statements in a file group are used to define a file
within an information area.  One or more file groups may be
contained within an information area definition.

A file group of component generation statements consists
of all statements needed to define the file.  These are the
FILE and optional FOPT, CYCLES, and COMPRESS statements.
```

The grouping and order of the statements is important because of the hierarchical nature of MCG statements. This diagram illustrates the order of statements:



4.2.4.2.1 File Statement (FILE)

This statement defines a CA MICS database file identifying the type of file, the timespans or areas, archive levels in which the file is supported, and the name of the file.

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1, free-form, and continuing through column 72. Continuation is not supported. The following describes the format and coding rules for the FILE Statement.

```
  FILE fff cc ll dt dy wk mn yr aa hw hm xxxxxxxxxxxxxxxxxxx
        |  |  |  |  |  |  |  |  |  |  |  |
        |  |  |  |  |  |  |  |  |  |  |  +---FNAME - Filename
        |  |  |  |  |  |  |  |  |  |  |
        |  |  |  |  |  |  |  |  |  |  +---HISTM -Archive
        |  |  |  |  |  |  |  |  |  |      monthly history
        |  |  |  |  |  |  |  |  |  |      definition.
        |  |  |  |  |  |  |  |  |  |
        |  |  |  |  |  |  |  |  |  +---HISTW - Archive weekly
        |  |  |  |  |  |  |  |  |      history definition.
        |  |  |  |  |  |  |  |  |
        |  |  |  |  |  |  |  |  +---AUDIT - Archive audit
        |  |  |  |  |  |  |  |      definition.
        |  |  |  |  |  |  |  |
        |  |  |  |  |  |  |  +---YEARS - Years timespan
        |  |  |  |  |  |  |      definition.
        |  |  |  |  |  |  |
        |  |  |  |  |  |  +---MONTHS - Months timespan
        |  |  |  |  |  |      definition.
        |  |  |  |  |  |
        |  |  |  |  |  +---WEEKS - Weeks timespan
        |  |  |  |  |              definition.
        |  |  |  |  |
        |  |  |  |  +---DAYS - Days timespan definition.
        |  |  |  |
        |  |  |  +---DETAIL - Detail timespan definition.
        |  |  |
        |  |  +---LEVEL - File expansion level.
        |  |
        |  +---FCODE - File type code.
        |
        +---FID - File Identification (base and parallel IDs
                  if a parallel definition - FCODE >= 50).
```

```
STATEMENT OPTION DEFINITIONS

          FILE    - Defines a data file.  The file contains all of
           the data elements defined following this
           statement.  This statement defines the type of
           file, the timespans and areas, archive levels in
           which the file is supported, and the name of the
           file.

           FID     - A 3-character or 6-character file ID.

                     A 3-character file ID denotes a
                     standard CA MICS file.

                     A user file must have a 'U' or 'Z' as
                     its first letter to denote that it is a
                     user file and ensure uniqueness in the
                     file's identification.  Additionally,
                     MSI cannot be used as a file name.

                     A parallel file is a file that has the
                     same record structure as another file.
                     A parallel file's FID is two
                     3-character file identifications--the
                     first three characters are the ID of
                     the parallel file, and characters 4
                     through 6 are the ID of the database
                     file after which the parallel file is
                     patterned.

            FCODE   - Defines the type of file that is to be
                     processed using the following codes:

                     Standard File

                        00 - defines standard file for
                             DETAIL, DAYS, WEEKS, MONTHS,
                             and YEARS database timespans.

                        01 - defines a standard file for the
                             TABLES database area.

                        02 - defines an interim work file
                             typically used during the DAILY
                             process.  Contact CA MICS
                             Product Support before
                             deactivating an interim work
                             file.
```

```
   03 - defines a CA MICS Accounting
        and Chargeback Journal File.

   04 - defines a standard file that is
        processed as a spin file.

        A spin file maintains
        information spanning a
        specified period of time.The
        specified time period is
        subtracted from the current
        date during a DAILY update
        cycle to determine a cutoff
        date.  Data older than the
        cutoff is dropped from the
        file.  Only cycles 00 and 01
        exist and are aged.

   05 - defines a standard file that
        exists only in the WEEKS,
        MONTHS, and/or YEARS timespans.
        It does not exist in DETAIL or
        DAYS, and requires special
        processing for the first
        summarization.
Parallel File

A parallel file is a file that has the
same record structure as another file.

   50 - defines parallel file for
        DETAIL, DAYS, WEEKS, MONTHS,
        and YEARS database timespans.

   51 - defines a parallel file for the
        TABLES database area.

   52 - defines a parallel interim work
        file.
```

```
LEVEL    - Defines the level in which file
           expansion takes place.  This option
           defines that a file is dependent on a
           file that precedes it in the expansion
           process.  Valid values range from 1 to
           n.

           For example, if FileA must be processed
           before FileB, FileA's level is set to 1
           and FileB's level is set to 2.

DETAIL   - Defines if the file is to be
           maintained in the DETAIL timespan.  Y
           if yes and N if no.

DAYS     - Defines if the file is to be
           maintained in the DAYS timespan.  Y
           if standard support, N if no, and E
           for special processing support.

WEEKS    - Defines if the file is to be
           maintained in the WEEKS timespan.
           Y if standard support, N if no, and
           E for special processing support.

MONTHS   - Defines if the file is to be
           maintained in the MONTHS timespan.
           Y if standard support, N if no, and
           E for special processing support.

YEARS    - Defines if the file is to be
           maintained in the YEARS timespan.
           Y if standard support, N if no, and
           E for special processing support.

AUDIT    - Defines if the file is to be included
           in the weekly audit archive.  Y if
           yes and N if no.

HISTW    - Defines if the file is to be included
           in the weekly history archive.  Y if
           yes and N if no.

HISTM    - Defines if the file is to be included
           in the monthly history archive.  Y if
           yes and N if no.
```

```
            FNAME    - The 40-character name used to describe
                       the file.


Special consideration for online timespans:

A file is active in a timespan if its timespan
definition contains a Y or E and is inactive if its timespan
definition contains an N.

The difference between the Y and E options is that Y
results in standard summarization code being generated to
update the file, and E requires the use of an exit to perform
special processing on the file.  The exit member used with
these files is contained in sharedprefix.MICS.SOURCE members
named #ffftSUM, where "fff" is the three-character file
identifier and "t" is either X, D, W, M, or Y depending on
the timespan being processed.
STATEMENT DEFINITION EXAMPLE

  Standard Database File

    FILE UDE 00 1 Y Y Y N N N Y N SYSLOG Device Error File

This statement defines the "SYSLOG Device Error File" as
a standard file (00) with the 3-character file ID of "UDE".
The file is to be maintained in the DETAIL (Y), DAYS (Y), and
WEEKS (Y) timespans and is only supported in the archive
weekly history (Y).
  Parallel Database File

    FILE _STPGM 50 1 Y N N Y N N N Y System Task Pgm Activity

This statement defines the "System Task Pgm Activity"
file as a parallel file (50) whose 3-character file ID is
"_ST" and whose structure is patterned after "PGM" file.  The
_ST file is to be maintained in the DETAIL (Y), MONTHS (Y),
and archive monthly history (Y) timespans.
```

4.2.4.2.2 File Option Statement (FOPT)

This optional statement defines the Essential and Derived data element options to be associated with a CA MICS database file. Defaults are used when the statement is absent. If the statement is present, it must immediately follow the FILE statement.

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1, free-form, and continuing through column 72. Continuation is not supported. The following describes the format and coding rules for the FOPT Statement.

```
  FOPT  eeeeeeee ddddddddd tttttttt xxxxxxxx
           |        |        |        |
           |        |        |        |
           |        |        |        |
           |        |        |        +---BYPASS -Optional
           |        |        |            protection bypass
           |        |        |
           |        |        +---TYPE - File option type
           |        |
           |        +---DERIVED - Derived data element
           |            option
           |
           +---ESSENTIAL - Essential data element option
```

STATEMENT OPTION DEFINITIONS

FOPT - Defines the Essential and Derived data element options to be used for the database file defined in the preceding FILE statement.

ESSENTIAL - Defines the Essential data element option to be used for the file. The value specified for this option must be DEFAULT, INITIAL, ONLY, or ALL as described below.

DERIVED - Defines the Derived data element option to be used for the file. The value specified for this option must be DEFAULT, COMP, or DISK as described below.

```
          TYPE      - Controls the activation or deactivation
                      of files in timespans.  Valid values
                      are:

                      o STD - does not deactivate the file in
                              any timespan

                      o NONESSL - deactivates the file in all
                                  timespans

                      o APPL - deactivates the file in the
                               WEEKS, MONTHS, and YEARS
                               timespans

                      o USER - synonym for APPL

          BYPASS    - This optional parameter is normally
                      left blank.  There are safeguards in
                      the evaluation of the Essential option
                      that help prevent inadvertent data
                      loss.  If you wish to bypass these
                      safeguards and force the Essential
                      option for all files in a component to
                      a value of INITIAL or ONLY unless
                      overridden on the FOPT statement, code
                      the word BYPASS in this position on the
                      COMPOPT statement.  Otherwise, for the
                      protection of the data on an existing
                      database, do not code this parameter.


Actions of the Specifications

    Definitions necessary for the discussion:

    o A derived data element is a data element defined in a
      TYPE group whose DATATYPE (as defined in the TYPE
      statement documentation that follows) is C, CN, XC, or
      XCN.  Its value results from a calculation.

    o An essential data element is a data element defined in
      a TYPE group by a NAME statement.  (Data elements that
      are not essential are defined by the NAMX statement.)
      "Essential" indicates that the element is needed either
      to calculate the values of other elements or is used in
      the standard reports provided with the product.

+-------------+-----------------------------------------+
```

If ESSENTIAL is set to ...	then ...
ALL	all data elements defined in the component's cccGENIN member on sharedprefix.MICS.GENLIB are stored on the disk (when DERIVED is DISK) or are available through the derived element macros (when DERIVED is COMP).
ONLY	only the data elements essential for file key structures, standard reports, and CA MICS management support applications are stored on disk. (These data elements are defined using NAME statements.) All other data elements are dropped from CA MICS processing and are not available without reprocessing raw DETAIL data. When ONLY is specified, the DERIVED option determines whether essential computed elements are stored on disk (when DERIVED is DISK) or are available through the derived element macros (when DERIVED is COMP).
INITIAL	only essential data elements are stored on disk, which is similar to the ONLY option. What differs between ONLY and INITIAL is that INITIAL forces the DERIVED statement's value to COMP, meaning essential computed elements are available only through the derived element macros. Moreover, the WEEKS, MONTHS, and YEARS timespans of all user and application unit identified files are disabled.
DEFAULT	essential data element values are taken from the definition in the CPLXDEF member of the sharedprefix.MICS.PARMS library.

If DERIVED is set to ...	then ...

```
| DISK         | derived data element values are stored on  |
|              | auxiliary storage.                         |
|              |                                            |
| COMP         | derived data element values are            |
|              | recalculated when needed rather than being |
|              | stored on auxiliary storage.               |
|              |                                            |
| DEFAULT      | derived data element values are taken from |
|              | the definition in sharedprefix.MICS.PARMS  |
|              | member CPLXDEF.                            |
+--------------+--------------------------------------------+
```

Essential and Derived data element option values can be specified at three levels:

- the complex level, which applies to all products in the complex
- the product level, which applies to all files in the product
- the file level, which applies to a single file

Normal specifications for the component and file option values are "DEFAULT", indicating that the complex definitions found in sharedprefix.MICS.PARMS(CPLXDEF) are to be used. This relationship is illustrated by the following table:

CPLXDEF Options	COMPOPT Options	FOPT Options	Generated File Options
ALL DISK	DEFAULT DEFAULT	DEFAULT DEFAULT	ALL DISK
ALL COMP	DEFAULT DEFAULT	DEFAULT DEFAULT	ALL COMP
ONLY DISK	DEFAULT DEFAULT	DEFAULT DEFAULT	ONLY DISK
ONLY COMP	DEFAULT DEFAULT	DEFAULT DEFAULT	ONLY COMP
INITIAL	DEFAULT DEFAULT	DEFAULT DEFAULT	INITIAL
any	ALL DISK	DEFAULT DEFAULT	ALL DISK
any	ALL COMP	DEFAULT DEFAULT	ALL COMP
any	ONLY DISK	DEFAULT DEFAULT	ONLY DISK
any	ONLY COMP	DEFAULT DEFAULT	ONLY COMP
any	INITIAL	DEFAULT DEFAULT	INITIAL
any	any	ALL DISK	ALL DISK
any	any	ALL COMP	ALL COMP
any	any	ONLY DISK	ONLY DISK
any	any	ONLY COMP	ONLY COMP
any	any	INITIAL	INITIAL

STATEMENT DEFINITION EXAMPLE

```
  Standard Database File

    FOPT DEFAULT DEFAULT STD
```

This statement defines that the file to which this file option statement belongs uses standard (STD) Essential data element options, and the Essential and Derived data element options will be taken from previous specifications (either the COMPOPT statement in the cccGENIN member itself, or from the CPLXDEF parameter member specifications as generated by the CPLXGEN job).

4.2.4.2.3 Database Cycle Retention Statement(Cycles)

The optional Database Cycle Retention statements define the default number of cycles that will be retained for a file in each respective timespan if the DBMODEL FILE statement is missing or if a value has not been coded for a timespan.

STATEMENT FORMAT

If the CYCLES statement is present, it MUST follow the FILE or FOPT statement in cccGENIN.

The CYCLES statement is coded with the KEYWORD starting in column 1, free-form, and continuing through column 72. Continuation is not supported. The following describes the format and coding rules for the CYCLES statement.

```
CYCLES  retx retd retw retm rety rett hretw hretm
   |     |    |    |    |    |    |     |     |
   |     |    |    |    |    |    |     |     +-- Months
   |     |    |    |    |    |    |     |         History
   |     |    |    |    |    |    |     |
   |     |    |    |    |    |    |     +-- Weekly History
   |     |    |    |    |    |    |
   |     |    |    |    |    |    +-- Tables
   |     |    |    |    |    |
   |     |    |    |    |    +-- Years
   |     |    |    |    |
   |     |    |    |    +-- Months
   |     |    |    |
   |     |    |    +-- Weeks
   |     |    |
   |     |    +-- Days
   |     |
   |     +-- Detail
   |
   +-- CYCLES - statement identification.
```

STATEMENT OPTION DEFINITIONS

See the FILE statement documentation in Section 2.3.4.1.2 of the PIOM for information about coding this statement. Section 4.3.13.2 of the PIOM contains additional information about the hretw and hretm parameters.

4.2.4.2.4 Compression Statement (COMPRESS)

The Compression statement defines whether or not the file is eligible to be compressed when the compression facility is activated. It is an optional statement; if it is not present, the file is not eligible for compression.

```
STATEMENT FORMAT

If the COMPRESS statement is present, it MUST follow the
CYCLES statement in cccGENIN.

The COMPRESS statement is coded with the keywords starting in
column 1.  Parameters are free-form and can continue through
column 72.  Statement continuation is not supported.  The
following describes the format and coding rules for the
COMPRESS statement.

  COMPRESS  x   d   w   m   y   (text)
     |      |   |   |   |   |   |
     |      |   |   |   |   |   +-- Never Compress
     |      |   |   |   |   |       Designation
     |      |   |   |   |   |
     |      |   |   |   |   +-- Years
     |      |   |   |   |
     |      |   |   |   +-- Months
     |      |   |   |
     |      |   |   +-- Weeks
     |      |   |
     |      |   +-- Days
     |      |
     |      +-- Detail
     |
     +-- COMPRESS - statement identification

STATEMENT OPTION DEFINITIONS

Valid values for the keywords are Y (yes) and N (no) for the
timespan keywords.  A value of Y indicates that the file's
data can be compressed in that timespan and a value of N
indicates that the file's data cannot be compressed in that
timespan.

If the file should not be compressed, then the timespan
keywords are set to N and the never compress text is present.
You should NOT override this designation because files are
usually designated as not eligible for compression for the
following reasons:

 - compressing the file negatively affects performance.

 - the file uses options that are not compatible with
   compression, such as POINT.
```

```
SAMPLE  STATEMENTS

The statement

COMPRESS Y Y Y Y N

means that the data in the file's DETAIL, DAYS, WEEKS, and
MONTHS timespans is eligible for compression when the
compression facility is activated.  Data in the YEARS
timespan files are not eligible for compression.

The statement

COMPRESS N N N N N (FILE COMPRESSION NOT SUPPORTED)

means that the file is not a candidate for compression.  Its
data should not be compressed, and this specification should
not be changed.
```

4.2.4.3 MSI Definition Statements

MSI is an extension to the CA MICS Component Generator (MCG), that creates Field Developed Applications (FDAs) that use SAS data sets as input. Since MSI generates the code, it allows for rapid development of FDAs to integrate new data into the CA MICS database. The MCG supports the generation of MSI FDAs using the special MSI statements listed below.

After reviewing this section as well as Section 7, you may refer to Appendix A for a checklist on installing an MSI generated FDA.

```
    ** THE ORDER OF STATEMENT APPEARANCE IS CRITICAL **

If optional statements are used, they must appear in the
order shown in the list which follows.  For example, if a
COMPUTE statement is being used, it MUST appear after the
RENAME statement.

These statements are documented in the following sections:

    1 - MSI Options Statement (MSIOPTS)
    2 - Input Data Set Statement (INPUTSAS)
    3 - Variable Rename Statement (RENAME)
    4 - Required Data Element Definition
    5 - Common Exit Definition Statement (COMMONEXIT)
    6 - Data Type Definition Statement
    7 - Data Element Initialization Statement(INITIALIZE)
    8 - File Sequence Statement (SEQUENCE)
    9 - Data Element Retention Statement (KEEP/DROP)
   10 - Timespan Modification Statements
        (DETAIL, DAYS, WEEKS, MONTHS, YEARS)
```

4.2.4.3.1 MSI Options Statement (MSIOPTS)

```
This statement defines the special options for MSI Field
Developed Applications.

STATEMENT FORMAT

The MSIOPTS keyword is coded starting in column 1, followed
by one or more option keywords, and continuing through column
72.  Multiple MSIOPTS statements can be coded, but
continuation of a single MSIOPTS statement is not supported.

  MSIOPTS options ...

STATEMENT OPTION KEYWORDS

 The MSIOPTS keywords specify options unique to MSI Field
 Developed Applications.  MSIOPTS statements coded before the
 first FILE statement specify global options used by all
 files in the FDA.  MSIOPTS statements coded after a FILE
 statement specify options for that file only.  A FILE level
 option overrides the same option specified in a global
 MSIOPTS statement.
```

```
SPLITCHAR character

Some input data sets contain a special character in the
variable label used by SAS to control line breaks when
printing the label.  The SPLITCHAR option is provided as a
means to instruct MSI to remove the special character from
data element labels during MSI FDA file creation.

FILTER filtername | NONE

Certain data sources contain SAS variables that have no
meaning under MSI.  Generally, these are date and time
sequences.  MSI currently supports filter types of MXG
(the MXG data source), and USR (user defined).

Note: For each file in fdaGENIN that has an input SAS
dataset that comes from MXG, if ENDTS, STARTTS, and ORGSYSID
statements are not specified, you must use the following:

  MSIOPTS FILTER MXG

CREATEWITH datafilename

This option is coded on an MSIOPTS statement which follows
a FILE statement.  That preceeding file statement
specifies a data file that will be created at the same
time as datafilename.  Datafilename was specified as as
the FID of a data file defined on an earlier FILE
statement in this GENLIB library member.

Createwith files are changed from INPUTSAS to INPUTMODEL.
This means that a separate input step will not be created
for this file.

Detail summarization will be accomplished after the
datafilename detail files have been sorted.

MULTWORK | NOMULTWORK

Determines if the file is eligible for Generalized Work
File Support.  This feature allows CA MICS to distribute
work space over several work data sets.
```

```
STATEMENT DEFINITION EXAMPLE

* SET DEFAULT OPTIONS

COMP ....
INPUTSOURCE MSI
* SET DEFAULT OPTIONS
MSIOPTS SPLITCHAR * FILTER MXG
MSIOPTS NOMULTWORK
AREA ...
FILE ...
* OVERRIDE DEFAULTS AND SET CREATEWITH
MSIOPTS SPLITCHAR NONE
MSIOPTS MULTWORK
MSIOPTS CREATEWITH FFF
```

4.2.4.3.2 Input Data Set Statement (INPUTSAS)

```
The INPUTSAS and INPUTMODEL statements are used to specify
the SAS dataset that will be used as input for a file in an
MSI FDA.

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported.  The following describes the format and coding
rules for the INPUTSAS statement.

 INPUTSAS   libname.member FILTER fltrname
 INPUTMODEL        |         |
                   |         +--- FILTER
                   |
                   +--- SASDS
```

```
STATEMENT OPTION DEFINITIONS

 INPUTSAS      - Defines the input SAS dataset for this
 INPUTMODEL      file.

                 SASDS    - The name of the input SAS dataset.
                            This dataset name becomes the
                            default input data for all
                            installed units.  The libname must
                            be specified in the cccCGEN JCL.

                 FILTER   - The MSI defined filter used to
                            exclude variables from the SASDS.
                            This is an optional parameter.

                 fltrname- NONE - no filtering

                           MXG  - filter variables for the
                                  MXG data source.

                           USR  - user defined filtering.
                                  Please refer to the
                                  MSIFILTR macro for
                                  instructions on how to add
                                  or modify filters.

                           This is an optional parameter and
                           only needs to be specified if
                           FILTER is specified.

STATEMENT DEFINITION EXAMPLE

INPUTSAS PDB.TYPE50
```

4.2.4.3.3 Variable Rename Statement (RENAME)

```
This statement allows the user to change the name of a
variable in the INPUTSAS data set. Renames are applied as the
input data is read from the SAS data sets via the RENAME
option.  Note that all subsequent MSI statements must use the
RENAMEd value.
```

```
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported.  The following describes the format and
coding rules for the RENAME statement.

  RENAME orgname  newname
    |      |         |
    |      |         |
    |      |         +--- NEWNAME - New variable name
    |      |
    |      +---ORGNAME - original name of the variable on
    |                    the input SAS data set.
    |
    +-----Statement identifier

STATEMENT OPTION DEFINITIONS

  RENAME  - Defines the renaming of an input SAS data set
            variable.

            ORGNAME - The eight-character name of the
                      variable on the INPUTSAS or INPUTMODEL
                      data set. If ORGNAME is the same as an
                      existing common data element, COMMON
                      status is revoked.

            NEWNAME - The eight character new variable name.
                      Note that all subsequent statements
                      must refer to NEWNAME.
```

4.2.4.3.4 Required Data Element Definition

For any MSI file, the CA MICS data elements ENDTS, STARTTS, and ORGSYSID must be defined and have values. The required data element statements (ENDTS, STARTTS, and ORGSYSID) are used to assign a SAS variable (from the INPUTSAS data set) to the particular data element.

```
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported.  The following describes the format and
coding rules for the statement.

  ENDTS       SASvar
  STARTTS        |
  ORGSYSID       |
    |            |
    |            |
    |            |
    |            |
    |            |
    |           +---SAS variable name
    |
    +-----Statement identifier

STATEMENT OPTION DEFINITIONS

  ENDTS    - Required data element definition statement
  STARTTS
  ORGSYSID
            SASvar  - The eight-character name of the
                      variable on the INPUTSAS or INPUTMODEL
                      data set.
```

4.2.4.3.5 Common Exit Definition Statement (COMMONEXIT)

```
The COMMONEXIT statement provides a special exit, coded in
SAS, that defines the required data elements (ENDTS, STARTTS,
and ORGSYSID).  This exit is executed after the required data
elements are set (using the MSI ENDTS/STARTTS/ORGSYSID
statements) and before user exit cccSYSID.

Do not use a RETURN, STOP, ABORT, or DELETE statement in
COMMONEXIT code.  This will produce unpredictable results
during processing.

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported.  Multiple COMMONEXIT statements can be
specified and will be presented to SAS in the order they
appear in the cccGENIN.  The following describes the format
and coding rules for the COMMONEXIT statement.
```

```
COMMONEXIT  SAScode
    |          |
    |          |
    |          |
    |          |
    |          +SAScode - SAScode
    |
    +-----Statement identifier


STATEMENT OPTION DEFINITIONS

COMMONEXIT- Defines the SAS code for defining the values of
            required MSI variables (ENDTS, STARTTS, ORGSYSID)

            SAScode - The remaining lines are used to define
                      SAS code used for the definition of
                      required MSI data elements (ENDTS,
                      STARTTS, and ORGSYSID).  Do not use
                      RETURN, LINK, ABORT, DELETE, or STOP
                      statements in this exit, as it can
                      interfere with normal CA MICS
                      processing.
```

4.2.4.3.6 Data Type Definition Statements

```
When SAS variables are converted to CA MICS data elements
during MSI processing, MSI assigns a default data type of
Accumulated for numeric variables, and Retained for character
variables.  MSI provides several statements that allow the
data type of data elements to be changed.

Many of the Data Type Definitions allow the specification of
generic data elements.  The colon is used to indicate a
generic element.  For example, ABC: indicates all data
elements starting with ABC.

    The various data type definition statements are described
    in the following sections:

    1 - Data Type Definition (MIN/MAX)
    2 - Data Type Definition (RETAIN)
    3 - Data Type Definition (COMPUTE)
    4 - Data Type Definition (AVE/PCT)
    5 - Alias Definition (ALIAS)
```

4.2.4.3.6.1 Data Type Definition(MIN/MAX)

These statements are used to define a data element as type MINIMUM or MAXIMUM. Both MIN/MAX support the definition of an optional initial value. Note that an automatic dependency is created between the data element and the seed value.

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1, free-form, and continuing through column 72. Continuation is not supported and multiple MINIMUM/MAXIMUM statements can be specified. The following describes the format and coding rules for the MIN/MAX statement.

```
MINIMUM   element/seed ...
  MIN         |     |
MAXIMUM       |     |
  MAX         |     |
   |          |     +----initial value element name prefixed
   |          |          with a slash (/).
   |          |
   |          +----data element name or generic name
   |
   +-----Statement identifier
```

STATEMENT OPTION DEFINITIONS

MINIMUM - Redefines the type of selected data elements to
MAXIMUM either MINIMUM or MAXIMUM.

element - Specifies which data element name will have its type changed to MINIMUM or MAXIMUM. Generic names are allowed and you can specify multiple elements and/or element/seed pairs.

/seed - The name of a data element that is used to set the initial value of the element. This is an optional parameter. If specified, you must specify the element/seed pair with the slash (/) and with no imbedded blanks.

4.2.4.3.6.2 Data Type Definition (RETAIN)

This statement allows a data element to be defined as RETAIN, and overrides the default specification as determined by MSI.

```
 STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported and multiple MINIMUM/MAXIMUM statements can be
specified.  The following describes the format and coding
rules for the RETAIN statement.

 RETAIN     element ...
    |          |
    |          +----data element name or generic name
    |
    |
    +-----Statement identifier
 STATEMENT OPTION DEFINITIONS

RETAIN    - Redefines the type of selected data elements to
            RETAIN.

            element - Specifies which data element name will
                      have its type changed to RETAIN.
                      Multiple elements can be specified per
                      RETAIN statements.  Generic
                      specification of elements is allowed.
```

4.2.4.3.6.3 Data Type Definition (COMPUTE)

```
MSI fully supports the definition of COMPUTEd data
elements.  To change a data element as COMPUTED, specify the
variable on the COMPUTE statement immediately followed by
the EXP and DEPEND statements.  EXP defines the actual SAS
code that will perform the computation.  DEPEND defines the
data elements that are required to be activated in order to
perform the computation.
```

```
 STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported and multiple COMPUTE statements can be
specified.  Only one element per statement is allowed.  The
following describes the format and coding rules for the
COMPUTE statement.

 COMPUTE    element
    |          |
    |          |
    |          |
    |          |
    |          |
    |          +----data element name
    |
    +-----Statement identifier
 STATEMENT OPTION DEFINITIONS

COMPUTE   - Redefines the type of selected data elements to
            COMPUTE.

            element - Specifies which data element name will
                      have its type changed to COMPUTE.
                      Generic specification of elements is
                      not allowed.

STATEMENT EXAMPLES

*
* BUSYTIME IS ASSUMED TO BE IN THE INPUTSAS DATASET
*
COMPUTE BUSYTIME
EXP 01 BUSYTIME = SUM(BUSYCPU0-BUSYCPU3, 0);
DEPEND BUSYCPU0 BUSYCPU1 BUSYCPU2 BUSYCPU3
```

4.2.4.3.6.4 Data Type Definition (AVG/PCT)

```
MSI can automatically generate data element DEPENDencies,
and the SAS code to perform either an average (A = B / C) or
a percent (A = B / C * 100) equation.

Note that the parameters for the AVERAGE or PERCENT statement
can only be data element names.  No constants or SAS
expressions are allowed.  If a more complex average is
needed, (e.g., A = (B+(C/47.8))/D), then the COMPUTE and its
companion statements (EXP, DEPEND) must be used.
```

```
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported.  The following describes the format and coding
rules for the statement.

 AVERAGE    element-A element-B element-C
   AVG         |       |          |
 PERCENT       |       |          |
   PCT         |       |          +---denominator data
    |          |       |              element name
    |          |       |
    |          |       +---numerator data element name
    |          |
    |          +----data element name
    |
    +-----Statement identifier
STATEMENT OPTION DEFINITIONS

AVERAGE   - Defines a data element as type AVERAGE or PERCENT
AVG
PERCENT
PCT
            element-A - data element that will contain the
                        result of the calculation.

            element-B - data element that is the numerator
                        for the expression.  Note that only
                        data elements are valid here.

            element-C - data element that is the denominator
                        for the expression.  Note that only
                        data elements are valid here.
STATEMENT EXAMPLES

In this example, the data element BUSYPCT contains the
percent busy CPU, and in the INPUTSAS data set is represented
as a value from 0 to 100.  It is necessary to provide an
equation so that BUSYPCT can be properly computed at higher
timespans.

FILE fff
INPUTSAS SAMPLE.DSN
PERCENT BUSYPCT BUSYTIME DURATION
```

4.2.4.3.6.5 Alias Definition (ALIAS)

```
MSI supports an alternate form of the ALIAS statement that
allows multiple data elements to be assigned to a single Data
Dictionary Alias.  This form can only be used with MSI FDAs.

STATEMENT FORMAT

This statement is coded with the keyword starting in column
1, free-form, and continuing through column 72.  Continuation
is not supported, although multiple ALIAS statements can be
specified.  The following describes the format and coding
rules for the ALIAS statement.

 ALIAS element...  name
    |   |          |
    |   |          |
    |   |          +----dictionary member name
    |   |
    |   +----data element name or generic name
    |
    +-----Statement identifier

STATEMENT OPTION DEFINITIONS

ALIAS     - Defines the name by which the data elements
            associated Data Dictionary description is
            stored.

            element - Specifies the data element names to
                      assign the ALIAS.  Generic
                      specification of elements is allowed.

            name    - The Data Dictionary Alias.  This is the
                      name of the sharedprefix.MICS.DIC.TEXT
                      member that provides dictionary
                      information for the listed elements.

STATEMENT EXAMPLES

*
ALIAS UCUPCTR1 UCUPCTR2 UCUPCTRX
```

4.2.4.3.7 Data Element Initialization Statement (INITIALIZE)

```
Some CA MICS data elements need to be initialized with a
value before they can be effectively used.  For example, data
element TRANSCNT needs to be initialized to one to show that
each detail observation represents a single transaction.  At
higher timespans, it contains the sum of TRANSCNT within the
BY group.

Initialization is performed prior to output to the work
data sets.

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported.  The following describes the format and
coding rules for the INITIALIZE statement.

INITIALIZE   element  SASexpr
    |           |       |
    |           |       |
    |           |       +--- SASExpr - SAS expression
    |           |
    |           +---element - data element name
    |
    +-----Statement identifier

STATEMENT OPTION DEFINITIONS

INITIALIZE - Specifies the initialization expression for a
             data element.

            element - Data element name.  Generic
                      specification is not allowed.

            SASexpr - The SAS expression that is used to
                      initialize the data element.  This can
                      be any SAS variable name, constant, or
                      expression.  Only one INITIALIZE
                      statement is allowed per data element.

STATEMENT EXAMPLE

INITIALIZE INTERVL 1
* SEMI-COLON IS OPTIONAL
INITIALIZE DURATION ENDTS-STARTTS;
$SCOL
```

4.2.4.3.8 File Sequence Statement (SEQUENCE)

```
All CA MICS files must have a SORT sequence.  In MSI, the
sort sequence is composed of two segments, the User Sequence,
and the Timespan Sequence.  The User Sequence is optional and
consists of user-specified data elements.  The Timespan
Sequence is automatically generated by the MCG and cannot be
modified (except for some minor tailoring).  It consists of
common data elements describing the date and time the file is
summarized and sorted.

The SEQUENCE statement allows the specification of the User
Sequence.  If no SEQUENCE is specified, then no User Sequence
is applied to the sort sequence.

**********************  IMPORTANT  ************************
*                                                         *
*  NOTE:  In addition to the SEQUENCE statement, you must  *
*         insert a NAME @@ACCT statement in each file in   *
*         which you want cccACTn data elements.           *
*                                                         *
**********************  IMPORTANT  ************************

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported, although multiple SEQUENCE statements can be
specified.  The following describes the format and coding
rules for the SEQUENCE statement.

  SEQUENCE elements ...
           |
           +---ELEMENTS - Data Elements

STATEMENT OPTION DEFINITIONS

  SEQUENCE- Defines the data elements in this file which
            will specify the data elements of the User
            Sequence.

            ELEMENTS - The data element names of the User
                       Sequence.  Data Elements can also
                       include @@ACCT, which is the account
                       code structure.  Generic specification
                       is not allowed.
```

```
STATEMENT EXAMPLE

SEQUENCE SYSID DEVNAME @@ACCT
```

4.2.4.3.9 Data Element Retention Statement (KEEP/DROP)

MSI has simplified the process of activating and deactivating data elements through the use of the new KEEP and DROP statements.

KEEP and DROP can be applied either Globally (affecting all timespans) or locally (affecting individual timespans). The following rules apply when using KEEP/DROP:

- All sequence (user or timespan) elements are always kept, and cannot be dropped.
- If there is no KEEP/DROP statement, the default action is to KEEP.
- Dropping a data element from the detail timespan causes MSI to drop it in all timespans.
- KEEP/DROP statements for individual timespans override the Global definition.

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1, free-form, and continuing through column 72. Continuation is not supported, although multiple KEEP/DROP statements can be specified. The following describes the format and coding rules for the KEEP/DROP statement.

```
KEEP   elements ...
DROP    |
 |      +---ELEMENTS - The names of elements which are
 |                     to be kept or dropped. Maximum of
 |                     eight characters each.
 |
 +----keyword- Statement keyword
```

```
STATEMENT OPTION DEFINITIONS

  keyword - Defines the data elements that will be kept or
            dropped from the file.  KEEP and DROP can be used
            in the same file.

            ELEMENTS - Specifies the name of the data
                       elements.  Generic specification is
                       allowed, and is indicated by the
                       colon(:) character.

STATEMENT EXAMPLES

KEEP BUSYTIME CPU:  GOPHER
DROP TUNA:  WINPROXY
```

4.2.4.3.10 Timespan Modification Statements

All of the previous statements affected data elements in all timespans. MSI extends functionality by allowing certain modifications to be made that only affect specific timespans.

The various ways MSI allows the sort sequence and data element retention modification for individual timespans are described in the following sections:

1 - Timespan Selection
2 - Sequence Specification
3 - Timespan Sequence Modification (AFTERSEQUENCE)
4 - User Sequence Modification (APPENDSEQUENCE)
5 - Data Element Retention

4.2.4.3.10.1 Timespan Selection

This statement is used to select the timespan for modification.

```
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported.  The following describes the format and coding
rules for the statement.

  DETAIL
  DAYS
  WEEKS
  MONTHS
  YEARS
    |
    |
    |
    +-----Statement identifier
STATEMENT OPTION DEFINITIONS

  (NONE)

STATEMENT EXAMPLES

  (NONE)
```

4.2.4.3.10.2 Sequence Specification

```
This statement is syntactically the same as the file sequence
statement. See section 4.2.4.3.8 of this guide.
If a timespan specific SEQUENCE is specified, the Global
sequence cannot be specified.  This implies that SEQUENCE
would have to be specified for all timespans.
```

```
STATEMENT EXAMPLES

DETAIL
SEQUENCE SYSID DSNAME VOLUME
*
DAYS
SEQUENCE SYSID DSNAME
*
WEEKS
SEQUENCE SYSID DSNAME
*
MONTHS
SEQUENCE SYSID DSNAME
*
YEARS
SEQUENCE SYSID DSNAME
*
```

4.2.4.3.10.3 Timespan Sequence Modification (AFTERSEQUENCE)

```
This statement allows for the file's sequence to be modified
by adding data elements to the end of the Time Sequence,
i.e., they are added LAST after all other data elements.
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported, although multiple statements are allowed.  The
following describes the format and coding rules for the
statement.


AFTERSEQUENCE  element ...
    |             |
    |             |
    |             +---element - data element name (no generic
    |                           specification).
    |
    +-----Statement identifier
STATEMENT OPTION DEFINITIONS

  AFTERSEQUENCE - Defines the data elements to be added after
                  the Time sequence (at the end of all
                  sequence elements).

             element - The name of data elements added after
                       the Timespan Sequence.  Generic
                       specification is not allowed.
```

```
STATEMENT EXAMPLES

SEQUENCE SYSID DSNAME
MONTHS
AFTERSEQUENCE DAY
```

4.2.4.3.10.4 User Sequence Modification (APPENDSEQUENCE)

```
This statement allows for the file's sequence to be modified
by inserting data elements after the User Sequence, i.e.,
they are inserted AFTER the User Sequence elements but BEFORE
the Time Sequence elements.
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported, although multiple statements are allowed.  The
following describes the format and coding rules for the
statement.

APPENDSEQUENCE  element ...
    |             |
    |             |
    |             +---element - data element name (no generic
    |                           specification) to be added
    |                           AFTER the user sequence and
    |                           BEFORE the Time sequence
    |
    +-----Statement identifier
STATEMENT OPTION DEFINITIONS

  APPENDSEQUENCE - Defines the data elements to be added
                   after timespan sequence.

            element - The name of data elements added to the
                      timespan Sequence.  Generic
                      specification is not allowed.
STATEMENT EXAMPLES

SEQUENCE SYSID DSNAME
DETAIL
APPENDSEQUENCE JOBID
```

4.2.4.3.10.5 Data Element Retention

```
This statement is syntactically the same as the file sequence
statement. See section 4.2.4.3.8 of this guide.
```

```
STATEMENT EXAMPLES

DETAIL
KEEP  SYSID DSNAME VOLUME BUSY:
DROP BUSYTIME
*
```

4.2.4.4 Standard Data Element Definition Statements

Five of the generation definition statements define data elements within a file. These statements are grouped in clusters, according to function. The statement types are TYPE, ALIAS, NAME, EXP, and DEPEND.

The TYPE statement defines summarization characteristics for one or more data elements. A minimum of one TYPE statement per file must be defined, with no maximum number of TYPE statements per file.

The following apply to the definition of an individual data element:

- The ALIAS statement is optional. An ALIAS statement defines a data element name, other than the real name of the data element, under which MICF will look for documentation. For example, data elements TSOSDST1 through TSOSDST8 will be documented on the CA MICS data dictionary by the member called TSOSDSTX. An ALIAS statement applies only to the data element defined by the NAME statement following the ALIAS statement.

- The NAME (essential) or NAMX (non-essential) statement gives the characteristics of the data element itself. Included on this statement are the short name and title of the data element, the optional group to which the data element belongs (if any), and the timespans of the file on which the data element is kept or is part of the file key structure.

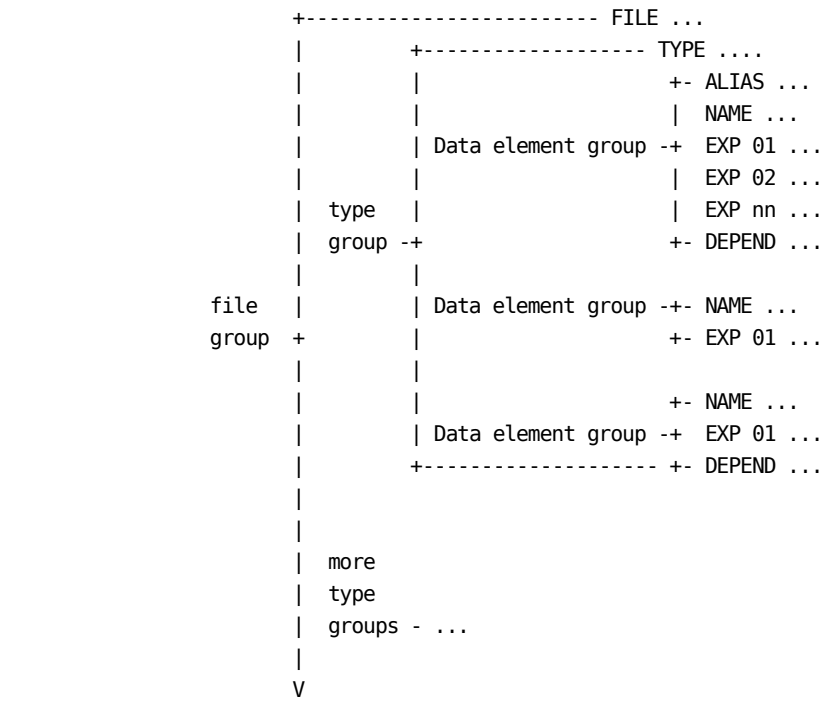
- The EXP statement is used for data elements whose summarization TYPE is C, for computed. The EXP statements following each NAME or NAMX statement for C-type data elements contain SAS code that calculates the value of the data element.

- The optional DEPEND statement relates the data element defined by the previous NAME or NAMX statement to one or more other data elements on the same file. Each DEPEND statement may list one or more data element names, and one or more DEPEND statements may be used for any data element.

  Note: DEPEND statements CANNOT refer to elements on other files.

  For example, the average response time for CICS transactions is CSYAVTTM. It would not be appropriate to have CSYAVTTM on the file in any timespan that does not contain the data elements necessary to calculate CSYAVTTM. These data elements are the total number of transactions CSYTRANS and the total response time CSYTRSTM.

The grouping and order of the statements are important because of the hierarchical nature of MCG statements. This diagram illustrates the order of statements:



4.2.4.4 Standard Data Element Definition Statements

Five of the generation definition statements define data elements within a file. These statements are grouped in clusters, according to function. The statement types are TYPE, ALIAS, NAME, EXP, and DEPEND.

The TYPE statement defines summarization characteristics for one or more data elements. A minimum of one TYPE statement per file must be defined, with no maximum number of TYPE statements per file.

The following apply to the definition of an individual data element:

- The ALIAS statement is optional. An ALIAS statement defines a data element name, other than the real name of the data element, under which MICF will look for documentation. For example, data elements TSOSDST1 through TSOSDST8 will be documented on the CA MICS data dictionary by the member called TSOSDSTX. An ALIAS statement applies only to the data element defined by the NAME statement following the ALIAS statement.

- The NAME (essential) or NAMX (non-essential) statement gives the characteristics of the data element itself. Included on this statement are the short name and title of the data element, the optional group to which the data element belongs (if any), and the timespans of the file on which the data element is kept or is part of the file key structure.

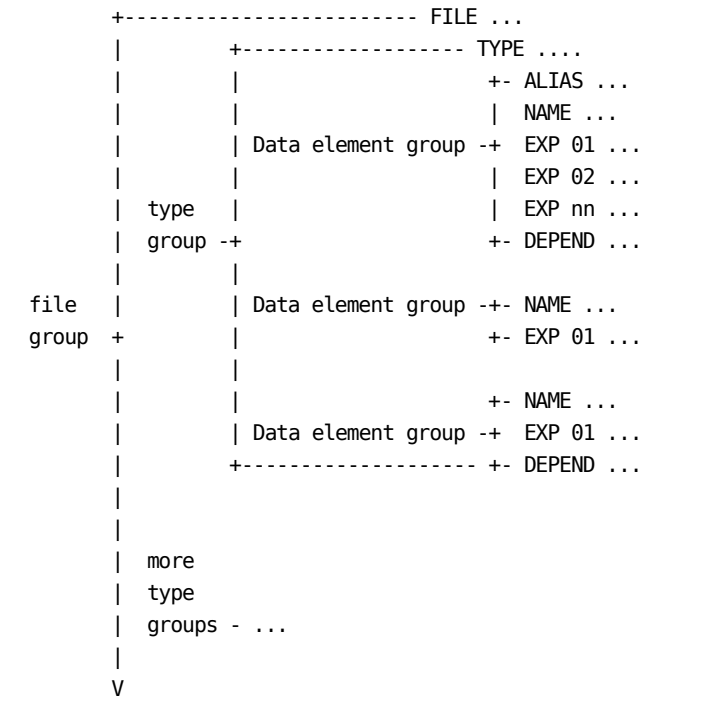
- The EXP statement is used for data elements whose summarization TYPE is C, for computed. The EXP statements following each NAME or NAMX statement for C-type data elements contain SAS code that calculates the value of the data element.

- The optional DEPEND statement relates the data element defined by the previous NAME or NAMX statement to one or more other data elements on the same file. Each DEPEND statement may list one or more data element names, and one or more DEPEND statements may be used for any data element.

  Note: DEPEND statements CANNOT refer to elements on other files.

  For example, the average response time for CICS transactions is CSYAVTTM. It would not be appropriate to have CSYAVTTM on the file in any timespan that does not contain the data elements necessary to calculate CSYAVTTM. These data elements are the total number of transactions CSYTRANS and the total response time CSYTRSTM.

The grouping and order of the statements are important because of the hierarchical nature of MCG statements. This diagram illustrates the order of statements:



4.2.4.4.1 Type Statement (TYPE)

This statement defines the type, length, and SAS format attributes for data elements.

```
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column
1, free-form, and continuing through column 72.  Continuation
is not supported.  The following describes the format and
coding rules for the TYPE Statement.

  TYPE tt lll.l ffffff lll.l fffffff lll.l fffffff
       |   |     |       |     |       |     |
       |   |     |       |     |       |     +---YMF - SAS
       |   |     |       |     |       |         FORMAT for
       |   |     |       |     |       |         YEARS and
       |   |     |       |     |       |         MONTHS
       |   |     |       |     |       |
       |   |     |       |     |       +---YML - SAS LENGTH
       |   |     |       |     |           for YEARS and
       |   |     |       |     |           MONTHS
       |   |     |       |     |
       |   |     |       |     +---DWF - SAS FORMAT for
       |   |     |       |         WEEKS and DAYS
       |   |     |       |
       |   |     |       +---DWL - SAS LENGTH for WEEKS and
       |   |     |                 DAYS
       |   |     |
       |   |     +---DTF - SAS FORMAT for DETAIL
       |   |
       |   +---DTL - SAS LENGTH for DETAIL
       |
       +---DATATYPE - Type of data element defined as
           described below.

STATEMENT OPTION DEFINITIONS

  TYPE    - Defines data elements' type, length, and SAS
            format attributes.  The definitions provided in a
            TYPE statement will be used for all data elements
            following the statement (NAME/NAMX statements)
            until a new TYPE statement is encountered.

            Note:  The length for any data element is forced
            to 8 if the MONEY format is used on the TYPE
            statement.
```

```
DATATYPE - Defines the type of data element and
           the manner in which this element is
           operated on by the CA MICS
           summarization process.  The valid
           codes include:

           R  - Retained data element
           A  - Accumulated data element
           M  - Maximum data element
           N  - Minimum data element
           C  - Derived data element with
                expressions
           CN - Derived data element with no
                expressions
           X  - Common data element
           XR - Common Retained data element
           XA - Common Accumulated data element
           XM - Common Maximum data element
           XN - Common Minimum data element
           XC - Common Derived data element
           XCN - Common Derived data element
                 with no expressions

DTL      - Defines the number of bytes that is to
           be used for the data element's LENGTH
           when used in the DETAIL timespan.
           The LENGTH is required.  Any valid SAS
           numeric or character LENGTH may be
           used.  The maximum length of this
           value is 5 characters.  Reference the
           recommendations on LENGTH definition
           provided below.

DTF      - Defines the display mask that is to be
           used as the data element's FORMAT and
           INFORMAT when used in the DETAIL
           timespan.  The format is not required
           and is coded as a (.) to specify no
           format.  Any valid SAS FORMAT may be
           used.  The maximum length of this
           value is 12 characters.  Reference the
           recommendations on FORMAT definition
           provided below.
```

```
DWL      - Defines the number of bytes that is to
           be used for the data element's LENGTH
           when used in the WEEKS or DAYS
           timespans.  The LENGTH is required.
           Any valid SAS numeric or character
           LENGTH may be used.  The maximum
           length of this value is 5 characters.
           Reference the recommendations on
           LENGTH definition provided below.

DWF      - Defines the display mask that is to be
           used as the data element's FORMAT and
           INFORMAT when used in the WEEKS or
           DAYS timespans.  The format is not
           required and is coded as a (.) to
           specify no format.  If no format has
           been specified but a FORMAT was
           defined for the DETAIL timespan
           (DTF), then that FORMAT will be used
           in place of the "no format"
           specification.  Any valid SAS FORMAT
           may be used.  The maximum length of
           this value is 12 characters.
           Reference the recommendations on
           FORMAT definition provided below.

YML      - Defines the number of bytes that is to
           be used for the data element's LENGTH
           when used in the YEARS or MONTHS
           timespans.  The LENGTH is required.
           Any valid SAS numeric or character
           LENGTH may be used.  The maximum
           length of this value is 5 characters.
           Reference the recommendations on
           LENGTH definition provided below.
```

```
YMF      - Defines the display mask that is to be
           used as the data element's FORMAT and
           INFORMAT when used in the YEARS or
           MONTHS timespans.  The format is not
           required and is coded as a (.) to
           specify no format.  If no format has
           been specified but a FORMAT was
           defined for the lower timespans (WDF
           or DTF), then that FORMAT will be used
           in place of the "no format"
           specification.  Any valid SAS FORMAT
           may be used.  The maximum length of
           this value is 12 characters.
           Reference the recommendations on
           FORMAT definition provided below.
```

```
LENGTH definition recommendations

   Integer values:       Maximum Value      SAS LENGTH

                                   255           2
                                65,535           3
                            16,777,215           4
                         4,294,967,295           5
                     1,099,511,627,775           6
                   281,474,946,710,655           7
                720,557,594,037,928,000           8

   Fractions to 0.1      Maximum Value      SAS LENGTH

                                 255.9           3
                              65,535.9           4
                          16,777,215.9           5
                       4,294,967,295.9           6
                   1,099,511,627,775.9           7
                 281,474,976,710,656.9           8


   Fractions to 0.01     Maximum Value      SAS LENGTH

                                255.99           3
                             65,535.99           4
                         16,777,215.99           5
                      4,294,967,295.99           6
                  1,099,511,627,775.99           7
                281,474,976,710,656.99           8

   Times in 0.01 Seconds Maximum Value      SAS LENGTH

                            HH:MM:SS.HS
                             18:12:15.99           4
                          4,660:20:15.99           5
                        ??????????????????         6
```

FORMAT definition recommendations

- Integer, fraction, and character data elements should not have FORMATs defined. Instead, enter a period (missing value) in the format field.

- Percentage data elements should have a LENGTH of 3 and a FORMAT of MAPCT, which will display the percentage as 999.99%.

Times in 0.01 Seconds	FORMAT	SAS LENGTH
	TIME11.2	5
	TIME15.2	6
	TIME18.2	7
	TIME18.2	8

- Timestamp data elements should have a LENGTH of 7 and a FORMAT of DATETIME19.2 (DDMMMYY:HH:MM:SS.HS) or DATETIME21.2 (DDMMMYYYY:HH:MM:SS.HS).

STATEMENT DEFINITION EXAMPLE

```
TYPE XM 7 DATETIME19.2 7 DATETIME19.2 7 DATETIME19.2
```

The data elements defined by the NAME statements to follow are common data elements that are to operated on as maximum values (XM). The value is a timestamp having a length of 7 bytes and a format of DATETIME19.2 (e.g. 09APR00:11:17:23.22 for April 9, 2000) for the DETAIL, DAYS/WEEKS, and MONTHS/YEARS definitions.

4.2.4.4.2 Alias Statement (ALIAS)

This statement defines the name by which this data element's associated data dictionary description is stored. The majority of data elements have their data dictionary elements stored as members with names that are the same as that of the element. There are cases where a number of different data elements have a common definition. In these cases, this definition need only be stored one time and an alias assigned to each of the applicable elements.

```
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column
1, free-form, and continuing through column 72.  Continuation
is not supported.  The following describes the format and
coding rules for the ALIAS statement.

   ALIAS xxxxxxxx
            |
            +---ALIASNAM - Data element name alias used to
                           name this element's data


STATEMENT OPTION DEFINITIONS

   ALIAS    - Defines the name by which this data element's
              associated data dictionary description is
              stored.

              ALIASNAM - Assigns a data dictionary alias for
                         the data element name instead of
                         using the standard data element name
                         in the data dictionary.
STATEMENT DEFINITION EXAMPLE

    ALIAS TSMDIST

This statement assigns an alias data dictionary name
"TSMDIST" to the data element defined in the NAME statement
immediately following this ALIAS statement.
```

4.2.4.4.3 Name Statement (NAME or NAMX)

```
This statement identifies a data element and defines its
program tag, 40-character label, group cluster code (which is
discussed with the OPTION statement), and the status of the
data element in each of the database's timespans.  Special
name definitions are used by the generator to indicate
different generation facilities.  These are explained under
the description for TAG.
```

```
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column
1, free-form, and continuing through column 72.  Continuation
is not supported.  The following describes the format and
coding rules for the statement.

   name nnnnnnnn c x d w m y xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
    |      |     | | | | | | |
    |      |     | | | | | | +---LABEL - element description
    |      |     | | | | | |
    |      |     | | | | | +---YEARS - Timespan data element
    |      |     | | | | |     active status
    |      |     | | | | |
    |      |     | | | | +---MONTHS - Timespan data element
    |      |     | | | |       active status
    |      |     | | | |
    |      |     | | | +---WEEKS - Timespan data element
    |      |     | | |       active status
    |      |     | | |
    |      |     | | +---DAYS - Timespan data element
    |      |     | |       active status
    |      |     | |
    |      |     | +---DETAIL - Timespan data element
    |      |     |       active status
    |      |     |
    |      |     +---CLUSTER - Data element cluster
    |      |           definition
    |      |
    |      +---TAG - Data element name (4-8 characters).   If
    |          the data element is not a Common data  element
    |          (X or Xx), then positions 1-3 must contain the
    |          file ID defined in the FILE statement.
    |
    +-----Statement identifier - NAME for essential data
          elements, or NAMX for non-essential data elements.

STATEMENT OPTIONS DEFINITION

  NAME    - Defines an essential data element.
  - or -
  NAMX    - Defines a non-essential data element.  Typical
            user component definitions do not use the NAMX
            statement because non-essential data elements
            would not be defined.
```

```
TAG      - Data element name (4-8 characters).  If
           the data element is not a Common data
           element, then positions 1-3 must
           contain the file ID defined in the FILE
           statement.

           There are several names that have
           special functions:

            @@ACCT - Specifies that the generated
                     account codes are to be
                     inserted in place of this
                     action name.  For example, if
                     the Batch and Operations
                     Analyzer was being generated
                     and the user defined two
                     account codes, then two
                     fields (ACCTNO1 and ACCTNO2)
                     would be substituted for the
                     @@ACCT field.

            @@FIRST- Specifies that computation
                     expressions (EXP) immediately
                     following this statement are
                     not to be associated with a
                     field, but are to be the
                     first computations made on
                     this file's content, prior to
                     calculating the standard
                     computed elements.  A TYPE
                     statement for computed (C)
                     elements must precede this
                     statement.

            @@LAST - Specifies that computation
                     expressions (EXP) immediately
                     following this statement are
                     not to be associated with a
                     field, but are to be the last
                     computations made on the
                     contents of this file after
                     calculating the standard
                     computed elements.  A TYPE
                     statement for computed (C)
                     elements must precede this
                     statement.
```

```
CLUSTER - Data element cluster definition.
          Normally set to 00, unless one of
          the following:

          Note:  See the individual Product
                 Guides for information on
                 cluster codes used in each data
                 integration product.

          Common Definition Cluster Codes

          Code Keyword Description
          ---- ------- --------------------------
          99   none    Common definition data
                       element

          99xx none    Common definition  data
                       element that is also a
                       CLUSTER of one of the
                       above clusters.

          For example, if the CLUSTER is defined
          as 01, then the element will be
          included if RACF is defined in the
          OPTION statement for the Batch and
          Operations Analyzer (SMFGENIN).  It
          will be excluded if NORACF is defined.

          Code 99 is used to point the MCG logic
          to a common definition of this data
          element, located in either cccGENIN or
          $DEGENIN.  If 99 is coded in CLUSTER,
          the LABEL field on the NAME statement
          and all description from the preceding
          TYPE statement is ignored.  That
          information is taken from the
          description of the data element with
          the same TAG in member cccGENIN or
          $DEGENIN.
```

```
                    Code 9901 would define a common data
                    element (99) that is controlled by the
                    RACF (01) option.

          DETAIL  - Timespan data element active status
                    (Refer to the status definition below.)

          DAYS    - Timespan data element active status
                    (Refer to the status definition below.)

          WEEKS   - Timespan data element active status
                    (Refer to the status definition below.)

          MONTHS  - Timespan data element active status
                    (Refer to the status definition below.)

          YEARS   - Timespan data element active status
                    (Refer to the status definition below.)

          LABEL   - A 40-character element description
                    (Reference CLUSTER 99, above).

Recommendation on timespan status definition

This definition determines an element's active and
sequence/summary status according to the following
guidelines:

    N - Inactive; the data element is not to be kept in the
        file for the designated timespan.

    0 (zero) - Active; the data element is to be kept in the
               file for the designated timespan.

    n - Active sequence/summary element; the data element is
        to be kept in the file for the designated timespan.

        "n" defines the order in which the data element is
        used to sequence or summarize the file (n=1 is the
        major sequence, n=2 the second most major sequence,
        and so forth).  Up to 20 sequence/summarization
        elements are allowed within a file's timespan
        definition.
```

The @@FIRST and @@LAST elements may not be defined as sequence/summary fields.

@@ACCT elements are generally defined as sequence/summary fields, but you do not need to consider the number of account codes that will replace this special name because the generator inserts them and increments the sequence numbers for you.

For example, if the sequence of a file is SYSID, account codes, and ENDTS, then SYSID would be defined as the most major field (n=1), @@ACCT the second most major field (n=2), and ENDTS the last field (n=3). If there is more than one account code, say ACCTNO1 and ACCTNO2, the generator inserts them correctly and increments the sequence definitions of those following to reflect the number of account numbers that were inserted. In this example, the sequences after generation are SYSID (first), ACCTNO1 (second), ACCTNO2 (third), and ENDTS (fourth).

Dn - Active sequence/summary element; the data element is to be kept in the file for the designated timespan. The sequence pattern of this data element will be descending.

4.2.4.4.4 Computation Statement (EXP)

This statement defines SAS statements necessary to calculate the value of a derived data element. The expression statement must follow the NAME statement (normal, @@FIRST, @@LAST) to which it applies and have a TYPE statement defined as compute (C). Note that an expression statement is not to be provided if the TYPE statement is defined as compute with no expression (CN).

This statement is used to define the computation of averages, percentages, and special computations (for example, avg working set size).

Note: The validity of the SAS statements is not checked during component generation. Errors in the expression will not be realized until derived elements are computed during the daily update or reporting.

```
STATEMENT FORMAT

   EXP nn xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
       |   |
       |   +---EXPSTAT - SAS statement(s).
       |
       +--- EXPNO - Expression sequence number.


STATEMENT OPTION DEFINITIONS

EXP - Defines the computation expression.

      EXPNO - Expression sequence number.  If there is only
              one statement to the expression this value is
              01.  If multiple statements are required then
              this sequence number defines the order that the
              statements appear and is incremented by one.

      EXPSTAT - SAS statements defining derivation formulas
                (columns 8-69).  Multiple statements are
                concatenated as a single expression.  The
                expression in a single statement is a maximum
                of 61 characters in length.

STATEMENT EXAMPLE

The following examples illustrate the recommended way
to compute an average, percentage, and special value.

Average Concurrent TSO Users
----------------------------

EXP 01 IF INTERVLS > 0 THEN TSOAVUSR=TSONOUSR / INTERVLS;


Percent of Batch Jobs Meeting Target
------------------------------------

EXP 01 IF JOBCOUNT > 0 THEN JOBPCMET=(JOBSRVMT*100)/JOBCOUNT;


Control Performance Group Average Working Set Size
--------------------------------------------------

EXP 01 IF PGAMSD*PGACTS > 0 THEN
EXP 02   PGAAVRSF=(PGAMTS*PGACSD*50)/(PGAMSD*PGACTS);
EXP 03 ELSE PGAAVRSF=0;
```

4.2.4.4.5 Data Dependency Statement (DEPEND)

```
This statement defines the data elements that are
required to compute the data element defined by the previous
NAME or NAMX statement or by the general computed definitions
of @@FIRST and @@LAST.  The DEPEND statement causes an
additional edit to be performed to ensure that an element
required in a computation is not inadvertently marked
inactive.

STATEMENT FORMAT

   DEPEND XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX
           |
           +---ELEMENTS - One to seven elements which are
                          required to compute the element(s)
                          in NAME, @@FIRST, or @@LAST.

STATEMENT OPTION DEFINITIONS

  DEPEND  - Defines the data elements in this file on which
            the computation of the element(s) in NAME,
            @@FIRST, or @@LAST depends.

            ELEMENTS - The 4-8 character field tags of the
                       fields required.  If an element is
                       defined on a DEPEND statement and has
                       been deleted from the file or marked
                       inactive (N) an error message will be
                       generated and the run aborted.

STATEMENT EXAMPLE

The following examples illustrate the use of the DEPEND
statement in conjunction with a computed expression.

NAME PGAAVRSF 00 0 0 0 0 0 Average Working Set Size
EXP 01 IF PGAMSD*PGACTS > 0 THEN
EXP 02   PGAAVRSF=(PGAMTS*PGACSD*50)/(PGAMSD*PGACTS);
EXP 03 ELSE PGAAVRSF=0;
DEPEND PGAMSD PGACTS PGAMTS PGACSD
```

4.2.4.5 Common Data Element Definition Statements

Common data elements are elements that are used by more than one CA MICS file. If a common element is used by files in more than one product, the common element is defined in the special generation library member called $DEGENIN. If a common element is used by files in a single product, the common element is defined in that product's cccGENIN member prior to the first AREA statement.

The characteristics of common data elements are defined in a single place (the GENIN member) and they are included in a file's definition through reference to the common definition (using the TYPE and NAME statements). These statements are grouped in clusters, according to function. The statement types used to describe common data elements are TYPE, ALIAS, ALTNAME, and NAME.

The TYPE statement defines the summarization characteristics of one or more data elements. Each file in the information area can have multiple TYPE statements.

Each of the other three statement types applies to the definition of an individual data element:

- The ALIAS statement is optional. An ALIAS statement is used to define an alternate data element name. The alternate name is used by MICF when it looks for the data element's documentation. For example, data elements IMSRVAL1 through IMSRVAL8 will be documented on the CA MICS data dictionary by the member called IMSRVALX. One ALIAS statement applies only to the data element defined by the NAME statement following the ALIAS statement.

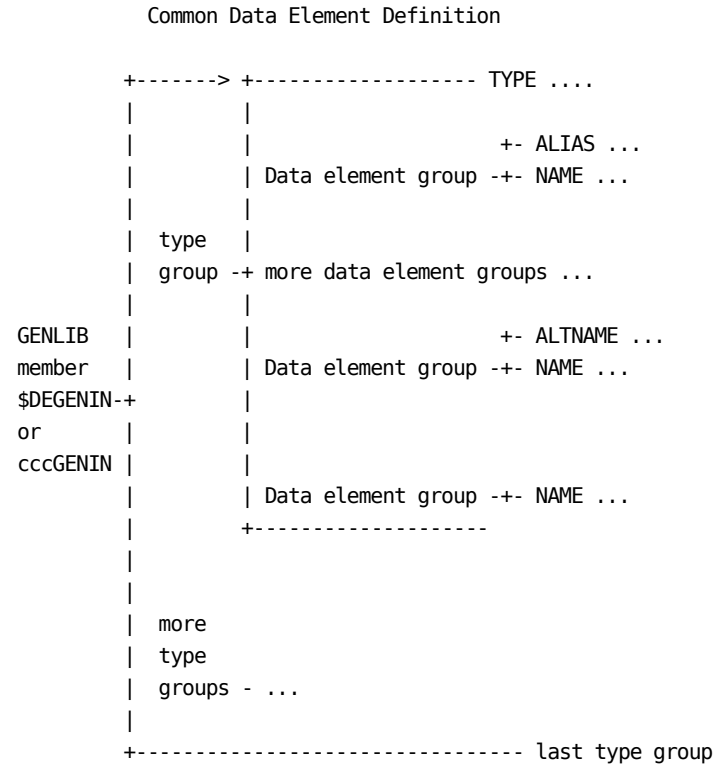
- The ALTNAME statement is used to replace the name of a common data element on all files in which that data element is kept.

- The NAME statement gives the name and title of the data element itself.

These statements are described in the following sections:

1 - Type Statement (TYPE)
2 - Alias Statement (ALIAS)
3 - Alternate Name Statement (ALTNAME)
4 - Name Statement (NAME)

The grouping and order of the statements is important because of the hierarchical nature of MCG statements. This diagram illustrates the order of statements:



4.2.4.5.1 Type Statement (TYPE)

This statement defines the type, length, and SAS format attributes for common data elements.

```
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1,
free-form, and continuing through column 72.  Continuation is
not supported.  The following describes the format and coding
rules for the TYPE Statement.

  TYPE tt lll.l ffffff lll.l fffffff lll.l fffffff
       |   |     |       |     |       |     |
       |   |     |       |     |       |     +---YMF - SAS
       |   |     |       |     |       |         FORMAT for
       |   |     |       |     |       |         YEARS and
       |   |     |       |     |       |         MONTHS
       |   |     |       |     |       |
       |   |     |       |     |       +---YML - SAS LENGTH
       |   |     |       |     |           for YEARS and
       |   |     |       |     |           MONTHS
       |   |     |       |     |
       |   |     |       |     +---DWF - SAS FORMAT for
       |   |     |       |         WEEKS and DAYS
       |   |     |       |
       |   |     |       +---DWL - SAS LENGTH for WEEKS and
       |   |     |                 DAYS
       |   |     |
       |   |     +---DTF - SAS FORMAT for DETAIL
       |   |
       |   +---DTL - SAS LENGTH for DETAIL
       |
       +---DATATYPE - Type of data element defined as
           computed with generic, common retained, common
           accumulated, common maximum, or common minimum.

STATEMENT OPTION DEFINITIONS

  TYPE    - Defines data elements' type, length, and SAS
            format attributes.  The definitions provided in a
            TYPE statement will be used for all data elements
            following the statement (NAME statements) until a
            new TYPE statement is encountered.
```

```
DATATYPE - Defines the type of data element and
           the manner in which this element is
           operated on by the CA MICS
           summarization process.  The valid
           codes include:

           XC - Common Derived element
           XR - Common Retained element
           XA - Common Accumulated element
           XM - Common Maximum element
           XN - Common Minimum element
           XCN - Common Derived data element
                 with no expressions

DTL      - Defines the number of bytes to use for
           the data element's LENGTH in the
           DETAIL timespan.  The LENGTH is
           required.  Any valid SAS numeric or
           character LENGTH may be used.  The
           maximum length of this value is 5
           characters.  Refer to the LENGTH
           definition recommendations below.

DTF      - Defines the display mask that is to be
           used as the data element's FORMAT and
           INFORMAT when used in the DETAIL
           timespan.  The format is not required
           and is coded as a (.) to specify no
           format.  Any valid SAS FORMAT may be
           used.  The maximum length of this
           value is 12 characters.  Refer to the
           FORMAT definition recommendations
           below.

DWL      - Defines the number of bytes to use for
           the data element's LENGTH when used in
           the DAYS or WEEKS timespans.  The
           LENGTH is required.  Any valid SAS
           numeric or character LENGTH may be
           used.  The maximum length of this
           value is 5 characters.  Refer to the
           LENGTH definition recommendations
           below.
```

DWF - Defines the display mask that is to be used as the data element's FORMAT and INFORMAT when used in the DAYS or weeks timespans. The format is not required and is coded as a (.) to specify no format. If no format has been specified but a FORMAT was defined for the DETAIL timespan (DTF), then that FORMAT will be used in place of the "no format" specification. Any valid SAS FORMAT may be used. The maximum length of this value is 12 characters. Refer to the FORMAT definition recommendations below.

YML - Defines the number of bytes to use for the data element's LENGTH in the YEARS or MONTHS timespans. The LENGTH is required. Any valid SAS numeric or character LENGTH may be used. The maximum length of this value is 5 characters. Refer to the LENGTH definition recommendations below.

YMF - Defines the display mask that is to be used as the data element's FORMAT and INFORMAT when used in the YEARS or MONTHS timespans. The format is not required and is coded as a (.) to specify no format. If no format has been specified but a FORMAT was defined for the lower timespans (DWF or DTF), then that FORMAT will be used in place of the "no format" specification. Any valid SAS FORMAT may be used. The maximum length of this value is 12 characters. Refer to the FORMAT definition recommendations below.

```
LENGTH definition recommendations

    Integer values:     Maximum Value      SAS LENGTH

                                  255           2
                               65,535           3
                           16,777,215           4
                        4,294,967,295           5
                    1,099,511,627,775           6
                  281,474,946,710,655           7
               720,557,594,037,928,000          8

    Fractions to 0.1    Maximum Value      SAS LENGTH

                                255.9           3
                             65,535.9           4
                         16,777,215.9           5
                      4,294,967,295.9           6
                  1,099,511,627,775.9           7
               281,474,976,710,656.9            8


    Fractions to 0.01   Maximum Value      SAS LENGTH

                              255.99           3
                           65,535.99           4
                       16,777,215.99           5
                    4,294,967,295.99           6
                1,099,511,627,775.99           7
             281,474,976,710,656.99            8

    Times in 0.01 Seconds Maximum Value      SAS LENGTH

                          HH:MM:SS.HS
                           18:12:15.99          4
                        4,660:20:15.99          5
                    ??????????????????          6
```

```
FORMAT definition recommendations

    o Do not specify formats for integer, fraction, and
      character data elements.  Instead, enter a period
      (missing value) in the format field.

    o Percentage data elements should have a LENGTH of 3 and
      a FORMAT of MAPCT, which will display the percentage as
      999.99%.

    Times in 0.01 Seconds      FORMAT        SAS LENGTH

                               TIME11.2           5
                               TIME15.2           6
                               TIME18.2           7
                               TIME18.2           8

    o Timestamp data elements should have a LENGTH of 7 and a
      FORMAT of DATETIME19.2 (DDMMMYY:HH:MM:SS.HS) or
      DATETIME21.2 (DDMMMYYYY:HH:MM:SS.HS).

STATEMENT DEFINITION EXAMPLE

    TYPE XM 7 DATETIME19.2 7 DATETIME19.2 7 DATETIME19.2

The data elements defined by the NAME statements to follow
are common data elements that are to be operated on as
maximum values (XM).  The value is a timestamp having a
length of 7 bytes and a format of DATETIME19.2
(09APR82:11:17:23.22) for the DETAIL, DAYS/WEEKS, and
MONTHS/YEARS definitions.
```

4.2.4.5.2 Alias Statement (ALIAS)

```
This statement defines the name by which this data
element's associated data dictionary description is stored.
The majority of data elements have their data dictionary
elements stored as members with names that are the same as
that of the element.  There are cases where a number of
different data elements have a common definition.  In these
cases, this definition need only be stored one time and an
alias assigned to each of the applicable elements.

The ALIAS statement for common data elements coded in
$DEGENIN is identical in content to the ALIAS statement for
ordinary data elements.
```

```
STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column
1, free-form, and continuing through column 72.  Continuation
is not supported.  The following describes the format and
coding rules for the alias statement.

   ALIAS xxxxxxxx
            |
            +---ALIASNAM - Data element name alias used to
                           name this element's data

STATEMENT OPTION DEFINITIONS

   ALIAS    - Defines the name by which this data element's
              associated data dictionary description is
              stored.

              ALIASNAM - Assigns a data dictionary alias for
                         the data element name instead of
                         using the standard data element name
                         in the data dictionary.

STATEMENT DEFINITION EXAMPLE

    ALIAS TSMDIST

This statement assigns an alias data dictionary name
"TSMDIST" to the data element defined in the NAME statement
immediately following this ALIAS statement.
```

4.2.4.5.3 Alternate Name Statement (ALTNAME)

```
This statement is used in situations when an alternate name
needs to be assigned to a common data element.  Please
consult the CA MICS Product Support Group before attempting
to use this MCG statement type.
```

4.2.4.5.4 Name Statement (NAME)

This statement identifies a data element and defines its program tag and 40-character label. The NAME statement used to describe common data elements differs from the NAME statement used for ordinary data elements:

- the element can be defined in either $DEGENIN or in the cccGENIN member prior to the first AREA statement
- the format of the NAME statement is simplified as illustrated below

Common elements are not defined using NAMX statements because they are essential.

STATEMENT FORMAT

This keyword is coded with the KEYWORD starting in column 1, free-form, and continuing through column 72. Continuation is not supported. The following describes the format and coding rules for the NAME statement for common data elements.

```
   NAME nnnnnnnn xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
           |                |
           |                +---LABEL - element description
           |
           +---TAG - Data element name (4-8 characters).
```

STATEMENT OPTIONS DEFINITION

```
  NAME    - Defines the data element.

            TAG     - Data element name, 4-8 characters (the
                      data element is a Common data element,
                      so there is no restriction on the
                      contents of positions 1-3).

            LABEL   - A 40-character element description.
```

Example:

```
NAME ENDTS End Time Stamp
```

4.2.5 Guidelines for Coding MCG Statements

In defining the statements for a file, we recommend that these rules be followed:

- Use the common data element cluster code specification of 99 in the NAME statements that define common elements to a file.

  While it is possible to define a common data element with different characteristics in different files, doing so defeats several major CA MICS design objectives that relate to the integration of data from various sources. Using the cluster code of 99 reduces the effort needed to code a user component, uses debugged element definitions, and helps maintain a high level of product integrity.

- Define ENDTS (End Time Stamp) in all timespans as a common, maximum field (XM). This will ensure that as data is summarized, the ENDTS will contain the highest ENDTS of the observations included. ENDTS is defined as a common data element in the $DEGENIN member of sharedprefix.MICS.GENLIB.

- Define STARTTS (Start Time Stamp) in all timespans as a common, minimum field (XN). This will ensure that as data is summarized, the STARTTS will contain the lowest STARTTS of the observations included. STARTTS is defined as a common data element in the $DEGENIN member of sharedprefix.MICS.GENLIB.

- All generation input members support a comment facility. Lines whose first non-blank character is an asterisk (*) are treated as comments and ignored by the MCG. If you change any distributed CA MICS generation input statements, comment the fact directly in the member. Note in the comment the date of the change, your name, the reason for the change, and the intended results.

- The CA MICS Component Generator translates lowercase characters to uppercase for those statement definition fields that must be uppercase. Lowercase characters are best used in labels on NAME statements.

4.2.6 MCG Illustrations

```
The statements listed below define the Network Performance
Analyzer (NPA) with one file, the NCP Activity File (NCP).

The examples in this chapter are based on a network
performance analysis (NPA) component that was superseded
entirely by the CA MICS SNA Network Analyzer. The references
to the NPA Component in this chapter provide an illustration
of how such tailoring operations would apply to any
hypothetical CA MICS component.

*
* The following GEN statement causes the MCG to generate file
* description members only.
*
GEN GENFILES
*
* The COMP statement identifies the NPA component.  This
* component is step 050 of the CA MICS operational jobs, so
* the NPA update step of the CA MICS DAILY job, for example,
* would step DAY050.  The input source for reading NPA data
* is a VBS data set of default blocksize.  The accounting
* component will not be active for NPA (because there is no
* user-related identifier on any NPA data).
*
COMP NPA 050 32000 VBS NOACCT NPA Component
*
* If COMPOPT, COMPJOB, INPUTSOURCE, or DBMDATA statements
* existed, they would be placed here.
*
* If an OPTION statement existed, it would be placed here.
*
* The following elements are common to NPA files.
*
TYPE XR  3 . 3 . 3 .
NAME NPANRRLS  Receiving Line Speed (line/link)
NAME NPANRSLS  Sending Line Speed (line/link)
TYPE XA  5  TIME11.2  6  TIME15.2  6  TIME15.2
NAME NPATMAOT  Actual Observation Time
TYPE XR  4  TIME8.  4  TIME8.  4  TIME8.
NAME NPATSEIS  Expected Interval Time
NAME NPATSFRU  Foreward RU Send Time
NAME NPATSLOG  Time Record Logged To SMF
*
* NPA's information area is also called NPA.
*
AREA NPA NPA Activity Information Area
*
```

```
* The following FILE statement describes the NCP Activity
* File.  It is a standard CA MICS file (the 00 after NCP),
* is not considered for introduction to the YEARS timespan,
* and has no archive audit application.
*
FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
*
* If FOPT, CYCLES, or COMPRESS statements were needed, they
* would be placed here.
*
* The next four data elements are retained numeric and are
* saved on the CA MICS database in the NPANCP files with a
* length of 3 bytes.
*
TYPE R 3 . 3 . 3 .
NAME NCPNRFBQ     00  0 0 0 0 0 Free Buffer Queue Length
NAME NCPNRHQL     00  0 0 0 0 0 NCP Channel Hold Queue Length
NAME NCPNRIQL     00  0 0 0 0 0 NCP Channel Intrmd Queue Len.
NAME NCPNRSFB     00  0 0 0 0 0 Free Buffer Count At Slowdown
*
* The next data elements are accumulated numeric elements.
* They will be summed in the generated summarization process.
* The saved data element length in timespans higher than
* DETAIL should be slightly larger to ensure that precision
* is not lost.  The format length also changes with the
* increased length.
*
TYPE A 5 TIME11.2 6 TIME15.2 6 TIME15.2
NAME NCPTMFCT     00  0 0 0 0 0 Free Cycle Time
NAME NCPTMTIS     00  0 0 0 0 0 Time In Slowdown
TYPE M 3 . 3 . 3 .
NAME NCPMXANP     00  0 0 0 0 0 Max Available NCP Buffers
NAME NCPMXFBH     00  0 0 0 0 0 Free Buffer High Water Mark
TYPE N 3 . 3 . 3 .
NAME NCPMNFBL     00  0 0 0 0 0 Free Buffer Low Water Mark
*
* The following data elements are common to files in more
* than one CA MICS product. The type and label descriptions
* for those data elements are contained on
* sharedprefix.MICS.GENLIB($DEGENIN).  Common data elements
* are generically named, according to their purpose or
* content, such as DURATION.  It is not necessary to code
* labels for common data elements.  A period (.) is coded to
* note the absence of a label.
*
NAME RESNAME      99  0 0 0 0 0 .
*
* DAY, below, appears in the DETAIL and DAYS timespans, but
* is dropped from the file in WEEKS and MONTHS.  If this file
```

```
* was active in the YEARS timespan (which it is not), DAY
* would have been dropped there too.  Furthermore, DAY is a
* part of the key structure for the NPANCP file.  In the
* DETAIL and DAYS timespans, DAY is the 5th data element in
* the key.

NAME DAY          99  5 5 N N N .
NAME HOUR         99  6 6 6 N N .
NAME WEEK         99  0 0 4 N N .
NAME YEAR         99  3 3 3 3 3 .
NAME MONTH        99  4 4 N 4 N .
NAME INTERVLS     99  0 0 0 0 0 .
NAME DURATION     99  0 0 0 0 0 .
NAME STARTTS      99  0 0 0 0 0 .
NAME ENDTS        99  7 0 0 0 0 .
NAME ZONE         99  0 0 5 5 4 .
NAME DAYNAME      99  0 0 N N N .
NAME SYSID        99  1 1 1 1 1 .
NAME NCPNAME      99  2 2 2 2 2 .
NAME NETNAME      99  0 0 0 0 0 .
*
* If any of the elements were documented under an alias name,
* the ALIAS statement would precede the NAME or NAMX
* statements.
*
* If any of the elements were computed, the EXP and DEPEND
* statements would follow the NAME or NAMX statements.
*
End of example
```

4.3 User Exit Facilities

Modifying existing code directly is certainly a way to change any software product. There may be situations in CA MICS that warrant such code modification. However, the large majority of extended CA MICS system tailoring can be done through the CA MICS User Exit Facilities.

There have been a large number of user exit points defined within CA MICS. Each of these points corresponds to a position in the CA MICS logic where user modification:

- is likely to occur, based on our experience.

- presents a low level of possibility to compromise CA MICS database and operational integrity.

- is relatively easy to document and understand.

Using the CA MICS exit facilities to modify CA MICS logic is the safest method of system modification. Following the material in this chapter will allow you to get a substantial amount of optional system customization done and will cause almost no extraordinary considerations for the application of standard vendor maintenance to the product.

4.3.1 Definition and Methodology

```
CA MICS exit points are provided to permit the addition of
 user code at key points in the execution of CA MICS.

 Within CA MICS, you can implement user code which is to be
 used throughout all database units in the CA MICS database
 complex, or you can selectively implement code to be used by
 a single database unit only.  Your careful attention to the
 structure of the CA MICS complex libraries and the exit
 routines assures the successful operation of your code.

 The text in the following sections makes many references
 to SAS language macro processing.  All references to SAS
 macro processing use the same terminology.

   o  The term "SAS Macro Statement" is used to refer to the
      macros in use before SAS version 82, which were coded in
      the form

           MACRO name ...content... %

   o  The term "SAS Macro Language" is used to refer to the
      macro language that is delivered with current levels of
      SAS.  This language includes both %MACRO constructions
      and global variables.  These are constructed in the form

           %MACRO name;
               ...content...
           %MEND name;

      or

           %LET name = ...content... ;

   o  The term "SAS Macro" without the reference to statement
      or language is used to refer to the concept of inserting
      variable code into CA MICS programs.

 These DOs and DON'Ts remind you about SAS coding techniques.
$SCOL
               ****  D O   the following:  ****


      DO:  Use a double percent sign (%%) when attempting to
           code a single percent sign within a SAS Macro
           Statement (but not with SAS Macro Language).  SAS
           senses this inside macro code and converts the
           compiled constant to a single %.
```

```
      DO:  Use the member that is supplied in the distributed
           version of the prefix.MICS.USER.SOURCE library as
           the starting point for your database unit local
           exits.

      DO:  Use the member that is supplied in the distributed
           version of the sharedprefix.MICS.SOURCE library as
           the starting point for your complex-wide exits.

      DO:  Liberally comment all code.

      DO:  Examine input record length to be sure INPUT
           requests can be satisfied from the current record.

      DO:  Reference input field locations with pointer logic
           where possible.

      DO:  Preserve data element integrity by type (numeric,
           time stamp, character variable) and range (having
           the value agree with the FORMAT and LENGTH
           specifications).

      DO:  Define new retained data elements by specifying
           the RETAIN statement after LENGTH and FORMAT
           statements, if applicable.

      DO:  Test all exit logic under the CA MICS-required SAS
           release BEFORE implementation into production.

      DO:  Call the CA MICS Product Support Group if you have
           ANY questions.
$SCOL
              ****   D O N ' T   do the following:  ****


      DON'T:  Define SAS macros in your exit code. SAS will
              not allow a macro to be defined within a SAS
              macro statement definition.

      DON'T:  Use columns 73-80 for SAS statements. CA MICS
              uses only 1-72.

      DON'T:  Code these SAS statements in your exit:  RETURN,
              DELETE, or INPUT without a trailing '@'.

      DON'T:  Code INPUT statements that could cause SAS to
              perform an automatic input statement skip.  Use
              the RECLEN variable to be sure that the record
              is big enough to contain data at any intended
```

```
        input offset.

     DON'T:  Forget the CA MICS naming standards, especially
             for DDnames, file names, and data element file
             prefixes.

     DON'T:  Assume that any data element is RETAINed or
             not RETAINed.

     DON'T:  Redefine any currently existing CA MICS data
             element unless the documentation of the specific
             exit in question says that it is allowed.

     DON'T:  Assume that the SAS automatic variable _N_
             contains the actual input observation number.

     DON'T:  Assume that setting the SAS automatic variable
             _ERROR_ will cause an input observation to be
             ignored or CA MICS processing to terminate.

     DON'T:  Use SAS language features not supported by the
             SAS release specified for use by the CA MICS
             update process.

 CA MICS exit points are described more fully in the sections
 that follow:

     1 - Direct Exit Points
     2 - Indirect Exit Points
```

4.3.1.1 Direct Exit Points

```
Direct exit points apply to both the SAS Macro Statement and
the SAS Macro Language exit structures.  The following text
addresses each in turn.
SAS MACRO STATEMENT EXIT STRUCTURE

Every exit explained in this section is implemented by
inserting SAS code into an existing SAS macro stub.  The stub
routine is of the form:

    MACRO macro-name
    %

where macro-name   indicates the function and location of the
                   macro within the CA MICS code.
```

CA MICS exit SAS macro names all begin with an underscore and are structured as follows:

```
    _USRxxxx     or
    _USRtfff
```

```
where xxxx  is a generic function name,
         t  is a 1-byte function code, and
       fff  is a file identifier.
```

These macros are stored in members called #cccEXIT on the sharedprefix.MICS.SOURCE library, where ccc is the component identifier. The exit routines contained in these members will apply to all database units in the CA MICS complex.

An example of such an exit member is #SMFEXIT, which contains the file manipulation, accounting modification, and operational exits for the Batch and Operations Analyzer. This member includes the definition of the _USRSJOB macro, which enables users to add or change elements on the BATJOB file. sharedprefix.MICS.SOURCE(#SMFEXIT) contains the skeleton for the _USRSJOB macro (among others), which looks like the following:

```
    .
    .
    MACRO _USRSJOB
    %
    .
    .
```

The exit members on the sharedprefix.MICS.SOURCE library are not invoked directly. Rather, the database unit logic references the name of the member on the prefix.MICS.USER.SOURCE library, which in turn %INCLUDEs the exit member from the sharedprefix.MICS.SOURCE library. For our example, the database unit's user source library contains an exit member called #SMFEXIT. The contents of prefix.MICS.USER.SOURCE(#SMFEXIT) are:

```
    /* some comments */
    %INCLUDE SOURCE(#SMFEXIT);
```

```
SAS MACRO LANGUAGE EXIT STRUCTURE

The general form of a direct Macro is:

    %MACRO exit-point-name;
     ...(content)...
    %MEND  exit-point-name;

where exit-point-name  indicates the function and location of
                       the Macro within the CA MICS code.

CA MICS direct exit point names are structured as follows:

    xxxxxxx

where xxxxxxx  is a generic function name.

Some CA MICS exit points are currently implemented through
the use of direct SAS Macro Language exit points.  An example
of a direct exit is the JCLEXIT exit point in the CA MICS JCL
generator.  See Section 4.3.2.3 in this guide for a detailed
description of this exit.

Regardless of the type of macro processing (Macro Statement
or Macro Language), some exits are intended for complex-wide
application.  Such exit logic is added to the exit member on
the source library for the complex, sharedprefix.MICS.SOURCE.
Thus, to add logic in our example, which will be executed at
the designated point in the daily SMF update process for all
database units on which the Batch and Operations Analyzer is
installed, put the code in source library member #SMFEXIT
between the macro name and the percent sign that follows, as:

contents of sharedprefix.MICS.SOURCE(#SMFEXIT):

    .
    .
    .
    MACRO _USRSJOB
     RETAIN MYCONST 5;
     JOBCPUTM = JOBCPUTM * MYCONST;
    %
    .
    .
    .
```

Some exits are intended for use by a specific database unit. This is the case for exits to be used with CA MICS Accounting and Chargeback. For this purpose, exit routine invocation is channeled through the prefix.MICS.USER.SOURCE library in each database unit. The previous definition of an exit macro can be overridden by a definition added after the sharedprefix.MICS.SOURCE library member is included. In the example, a _USRSJOB macro has been defined in sharedprefix.MICS.SOURCE(#SMFEXIT). To override this definition for a particular database unit, redefine the macro in prefix.MICS.USER.SOURCE(#SMFEXIT), as:

contents of prefix.MICS.USER.SOURCE(#SMFEXIT):

```
    /* some comments */
    %INCLUDE SOURCE(#SMFEXIT);
--->MACRO _USRSJOB
---> RETAIN MYCONST 4;
---> JOBCPUTM = JOBCPUTM * MYCONST;
--->%
```

The exit member referenced by the database unit's daily update cycle is in prefix.MICS.USER.SOURCE. In our example, the daily SMF format routine (DYSMFFMT) would include prefix.MICS.USER.SOURCE(#SMFEXIT), which itself includes sharedprefix.MICS.SOURCE(#SMFEXIT). The diagram below illustrates the flow for any given database unit.

```
DYSMFFMT:

   |.
   |.
   |%INCLUDE USOURCE(#SMFEXIT);
   |
             prefix.MICS.USER.SOURCE(#SMFEXIT)
             contains:
             |.
             |.
             |%INCLUDE SOURCE(#SMFEXIT);
             |
                       sharedprefix.MICS.SOURCE(#SMFEXIT)
                       contains:
                       |
                       |.
                       |.
                       |MACRO _USRSJOB
                       |user code;
                       |%
                       |.
                       |.
             |.
             |.
   |.
   |.
```

To code an exit local to the database unit, you need to redefine the macro in the prefix.MICS.USER.SOURCE library member, after the sharedprefix.MICS.SOURCE library copy or stub has been included. In our example, _USRSJOB would be redefined as:

```
DYSMFFMT:

   |.
   |.
   |%INCLUDE USOURCE(#SMFEXIT);
   |
             prefix.MICS.USER.SOURCE(#SMFEXIT)
             contains:
             |.
             |.
             |%INCLUDE SOURCE(#SMFEXIT);
             |
                       sharedprefix.MICS.SOURCE(#SMFEXIT)
                       contains:
                       |.
                       |.
                       |MACRO _USRSJOB
                       |shared user code;
                       |%
                       |.
                       |.
             |.
         --->|MACRO _USRSJOB
         --->|shared user code;
         --->|unit user code;
         --->|%
             |.
             |.
   |.
   |.
```

The definition of _USRSJOB in the user source library member comes after the definition of the macro in the source library as seen by the SAS compilation of the whole SMF update step. The second macro definition completely overrides the first, and thus the second definition must contain any code from the first definition that is to be preserved.

```
This method has an inherent problem.  The shared exit code is
supposed to apply to all database units in the complex.  If
a change to the shared exit code is needed, the code must be
updated in the exit member both in sharedprefix.MICS.SOURCE
and in prefix.MICS.USER.SOURCE.  This creates a problem
because of the probability that the user source library
version of the logic update might be forgotten.
```

This situation can be avoided by a simple construction. Define an extension macro for the macros that need to be extended in this way. The extension macro name would be of the form _USRUfff, where fff is the file name. The macro reference must be in the original macro definition (_USRSfff), and the stub for the _USRUfff macro must be defined in sharedprefix.MICS.SOURCE(#cccEXIT). Add the unit database local code to prefix.MICS.USER.SOURCE(#cccEXIT), under the macro name _USRUfff, and the entire code will resolve correctly at the reference to _USRSfff. In the SMF example, modify the members as indicated by "--->" in the following diagram:

```
   |.
   |.
   |%INCLUDE USOURCE(#SMFEXIT);
   |
             prefix.MICS.USER.SOURCE(#SMFEXIT)
             contains:
             |.
             |.
             |%INCLUDE SOURCE(#SMFEXIT);
             |
                       sharedprefix.MICS.SOURCE(#SMFEXIT)
                       contains:
                       |.
                       |.
                       |MACRO _USRSJOB
                       |shared user code;
                  --->| _USRUJOB
                       |%
                       |.
                       |.
                  --->|MACRO _USRUJOB
                  --->|%
                       |.
                       |.
             |.
        --->|MACRO _USRUJOB
        --->|unit user code;
        --->|%
             |.
             |.
   |.
   |.
```

This construction, using the _USRUfff macros, is applicable to the various CA MICS components. Not all the _USRUfff macro stubs that are possible are precoded for you by CA MICS, and you must add the _USRUfff macro reference and stub to the #cccEXIT member yourself if you wish to take advantage of this method. However, there are some exit macros that are precoded with _USRUfff stubs. Some of these precoded macros are listed below:

File/ Accounting Macro	User Extension Macro	Exit Code Location
_USRSJOB	_USRUJOB	sharedprefix.MICS.SOURCE(#SMFEXIT)
_USRSPGM	_USRUPGM	sharedprefix.MICS.SOURCE(#SMFEXIT)
_USRSSPL	_USRUSPL	sharedprefix.MICS.SOURCE(#SMFEXIT)
_USRSCAC	_USRUCAC	sharedprefix.MICS.SOURCE(#CICEXIT)
_USRSCSU	_USRUCSU	sharedprefix.MICS.SOURCE(#CICEXIT)
_USRSIAC	_USRUIAC	sharedprefix.MICS.SOURCE(#IMSEXIT)
_USRSISU	_USRUISU	sharedprefix.MICS.SOURCE(#IMSEXIT)
_USRS_IS	_USRU_IS	sharedprefix.MICS.SOURCE(#IMSEXIT)
_USRSSUA	_USRUSUA	sharedprefix.MICS.SOURCE(#IDMEXIT)
_USRSSAC	_USRUSAC	sharedprefix.MICS.SOURCE(#IDMEXIT)
_USRSTSI	_USRUTSI	sharedprefix.MICS.SOURCE(#TSOEXIT)
_USRSTSC	_USRUTSC	sharedprefix.MICS.SOURCE(#TSOEXIT)
_USRSTSU	_USRUTSU	sharedprefix.MICS.SOURCE(#TSOEXIT)
_USRUJPJ	_PWRJPJB	sharedprefix.MICS.SOURCE(#PWREXIT)
_USRUJPO	_PWRJPOA	sharedprefix.MICS.SOURCE(#PWREXIT)
_USRUJPP	_PWRJPPG	sharedprefix.MICS.SOURCE(#PWREXIT)

This method of tailoring affords you great flexibility in tailoring CA MICS and, because of the power of the SAS language, is easy to do. You should, however, always follow these guidelines:

- Validate the fields your exits use to make their processing decisions. While CA MICS does some basic validation, it cannot catch the kinds of installation specific errors you can (e.g., an invalid account number on a job card). When one of your exits detects invalid input data, it should NEVER pass it on into the database (i.e., pass through an invalid account number so it becomes part of the key of a record). Doing so will increase the size of the database and decrease its usefulness for analysis and reporting. Rather, your exits should take some sort of standard default action when they detect invalid input data (e.g., assign a job with an invalid account code to an overhead category).

- Exit code is inserted directly into CA MICS processing logic. It thus becomes the responsibility of the user to make certain that the code inserted will not do anything that would cause subsequent CA MICS logic to operate abnormally.

4.3.1.2 Indirect Exit Points

Indirect exit points are implemented by coding a SAS Macro variable to indicate the name of a user source library concatenated member to insert.

The general form of a indirect exit is:

```
%LET exit-point-name = user-source-member-name ;
```

where exit-point-name indicates the function and location of the Macro within the CA MICS code.

CA MICS indirect exit point names are structured as follows:

```
    USRtfff

where    t  is a 1-byte function code, and
       fff  is a file identifier.
```

These exit points are stored in members called #cccEXIT on the sharedprefix.MICS.SOURCE library, where ccc is the component identifier. The exit routines contained in these members will apply to all database units in the CA MICS complex.

For purposes of the description of indirect exits below, let us assume a hypothetical component identifier of ccc. An example of such an exit member is #cccEXIT, which contains the file manipulation, accounting modification, and operational exits for the CCC Component. This member includes the definition of an indirect exit point whose purpose is file and accounting modification of the CCCFFF file. The Macro is called USRSFFF. If sharedprefix.MICS.SOURCE(#cccEXIT) contains a %LET statement for the USRSFFF exit point, it must look like the following:

```
    .
    .
    %LET USRSFFF = EXITMEM1:
    .
    .
```

Coding the statement above will cause SAS logic residing in prefix.MICS.USER.SOURCE(EXITMEM1) to be %INCLUDEd and executed at the point at which the USRSFFF exit point occurs in distributed CA MICS code.

Indirect exit points are defined by the occurrence of a reference to the CA MICS %FXIT Macro in CA MICS SAS code. The %FXIT Macro is the CA MICS conditional include of user-coded exit logic. If the exit point for which the %FXIT Macro is referenced has been defined, a

```
   %INCLUDE USOURCE(member-name-list);
```

will be generated at that point in the CA MICS SAS compilation. If no exit point definition has occurred prior to the %FXIT Macro invocation, no %INCLUDE is generated.

Here is an illustration of the method, again using the example of file FFF in component CCC:

contents of sharedprefix.MICS.SOURCE(#cccEXIT):

```
    .
    .
    .
    %LET USRSFFF = EXITMEM1;
    .
    .
    .
```

contents of prefix.MICS.USER.SOURCE(EXITMEM1):

```
    RETAIN MYCONST 4;
    FFFCPUTM = FFFCPUTM * MYCONST;
```

Some exits are intended for use by a specific database unit. This is the case, for example, with exits to be used with CA MICS Accounting and Chargeback. For this purpose, exit routine invocation is channeled through the prefix.MICS.USER.SOURCE library in each database unit. The previous definition of an exit macro can be overridden by a definition added after the sharedprefix.MICS.SOURCE library member is included. In the example, a USRSFFF exit point has been defined in sharedprefix.MICS.SOURCE(#cccEXIT). To override this definition for a particular database unit, redefine the variable in prefix.MICS.USER.SOURCE(#cccEXIT), as:

contents of prefix.MICS.USER.SOURCE(#cccEXIT):

```
    /* some comments */
    %INCLUDE SOURCE(#cccEXIT);
--->%LET USRSFFF = EXITMEM2;
```

contents of prefix.MICS.USER.SOURCE(EXITMEM2):

```
    RETAIN MYTWO 2;
    FFFEXCPS = FFFEXCPS / MYTWO;
```

The exit member referenced by the database unit's daily update cycle is in prefix.MICS.USER.SOURCE. In our example, the daily CCC format routine (DYCCCFMT) would include prefix.MICS.USER.SOURCE(#cccEXIT), which itself includes sharedprefix.MICS.SOURCE(#cccEXIT). The diagram below illustrates the flow for any given database unit.

```
DYCCCFMT:

   |.
   |.
   |%INCLUDE USOURCE(#cccEXIT);
   |
             prefix.MICS.USER.SOURCE(#cccEXIT)
             contains:
             |.
             |.
             |%INCLUDE SOURCE(#cccEXIT);
             |
                       sharedprefix.MICS.SOURCE(#cccEXIT)
                       contains:
                       |
                       |.
                       |.
                       |%LET USRSFFF = EXITMEM1;
                       |.
                       |.
             |.
             |.
   |.
   |.
   |%FXIT(NAME=USRSFFF);
     (which resolves to null)
   |.
   |.
```

To code an exit local to the database unit requires you to redefine the variable in the prefix.MICS.USER.SOURCE library member after the sharedprefix.MICS.SOURCE library copy or stub has been included. In our example, USRSFFF would be redefined as:

DYCCCFMT:

```
   |.
   |.
   |%INCLUDE USOURCE(#cccEXIT);
   |
             prefix.MICS.USER.SOURCE(#cccEXIT)
             contains:
             |.
             |.
             |%INCLUDE SOURCE(#cccEXIT);
             |
                       sharedprefix.MICS.SOURCE(#cccEXIT)
                       contains:

                       |.
                       |%LET USRSFFF = EXITMEM1;
                       |.
             |.
          --->|%LET USRSFFF = EXITMEM1 EXITMEM2;
             |.
   |.
   | %FXIT(NAME=USRSFFF);
      (which resolves to)
             | %INCLUDE USOURCE(EXITMEM1 EXITMEM2);
               (which includes EXITMEM1)
                       | RETAIN MYCONST 4;
                       | FFFCPUTM = FFFCPUTM * MYCONST;
               (and includes EXITMEM2)
                       | RETAIN MYTWO 2;
                       | FFFEXCPS = FFFEXCPS / MYTWO;
             |.
   |.
   |.
```

4.3.2 Types of CA MICS Exits

The previous section explained where the CA MICS exit facility SAS macros reside. The members of the source library that contain these macros are included in CA MICS daily processing in several places; those places are determined by general groups of functions that the exit macros contain.

The three function groups are:

- General operational exits that apply to all CA MICS components.
- File building and accounting exits that apply to only one component.
- JCL generation output exit that applies to both complex- and unit-level generations.

Individual CA MICS product guides contain information about the #cccEXIT members.

4.3.2.1 General Operational Exits

```
Exits that apply to any or all installed components within
 CA MICS can be used to add logic for selecting records,
 extend checkpoint manipulation, rearrange operational jobs,
 expand ZONE capabilities definition, and so on.

 These exits are part of the CA MICS Platform and are located
 in sharedprefix.MICS.SOURCE(#BASEXIT).  The exits are:

 Exit Name: _USRSEL

 Purpose:   Perform input record selection in DAILY job.

 Method:    The macro is referenced in code that processes
            input data.  The routines are daily component
            format routines and the CA MICS input reader.
            Before an invocation, the program variable
            SKIP_REC is set to a value of zero.  If the exit
            sets this variable to a value of one, the calling
            routine bypasses the current input record and
            reads the next record.

            Because this exit is used in many places, it is
            necessary to qualify all coding in this exit by
            examining the name of the routine in which the
            exit was invoked.  The program variable ROUTINE
            exists for this purpose.  ROUTINE is an 8-byte SAS
            character variable containing the name of the
            invoking module in the DAILY update.

            A typical implementation of this method follows.

              MACRO _USRSEL
                IF ROUTINE = 'INPUTRDR' THEN DO;
                  (exit code here)
                END;
                ELSE IF ROUTINE = 'DYIMSFMT' THEN DO;
                  (exit code here)
                END;
              %

 Typical    This macro can be used to remove unwanted data
 Use:       from CA MICS processing. For example, there is a
            site with three CPUs logging to SMF.  One of the
            CPUs is usually dedicated to IMS; the SMF data is
            not to be processed by CA MICS, and that CPU's
            original SMF SYSID is not defined to CA MICS. If
            SMF data from that system is ever input to CA
```

```
MICS, the CA MICS daily job will ABEND with a user
310 SAS ABORT ABEND when the UNDEFINEDSYSID
control parameter is set to ABORT, indicating that
CA MICS does not know what to do with the data.
If it is not practical to remove the data from the
input file that is defined in the INPUTSMF DD
concatenation or to use the UNDEFINEDSYSID control
parameter, you can code a _USRSEL exit to delete
the records.  The following coding sample
illustrates this approach:

  MACRO _USRSEL
    IF ROUTINE = 'INPUTRDR' THEN DO;
      IF ORGSYSID = 'CPU1' THEN DO;
        SKIP_REC = 1;    /* SKIP CPU1 RECORDS  */
      END;
    END;
  %

This macro can also be used to output SMF records
to the prefix.MICS.USER.DATA in the DAYSMF step.
Input records are stored in this file when the
element USERFILE is set to a value of "1" in the
_USRSEL exit.  This file is allocated in the
DAYSMF step with DISP=OLD for output processing
and it is again allocated in the DAY500 step for
any user-defined processing, such as report
generation or user data collection.

The following coding sample illustrates how you
could output user SMF record type 299 to the USER
DATA file for processing by your own code in the
DAY500 step.

  MACRO _USRSEL
    IF ROUTINE = 'INPUTRDR' THEN DO;
       IF SMFRTYPE EQ 299 THEN USERFILE = 1;
    END;
  %

Invoked Locations:  This macro is referenced directly in the
          following locations:

          ROUTINE='INPUTRDR', DAILY job step DAYSMF
            Reference sharedprefix.MICS.SOURCE(INPUTRDR)

          ROUTINE='DYcccFMT', DAILY job step DAYsss
            (where ccc is the component identifier of a
            CA MICS product or a user component, and sss is
```

```
                    the step of the DAILY job coded on the COMP
                    statement in sharedprefix.MICS.GENLIB(cccGENIN))

                    NOTE:  DYcccFMT processing is defined in one
                           or more similarly-named members of
                           sharedprefix.MICS.SOURCE.


Exit Name: _USRIHL

Purpose:   Examine input history log

Method:    The macro is referenced in code that passes the
           DETAIL timespan checkpoint SAS file.  This code
```

```
           that file, and in the process allows you to
           inspect the file's contents.

Typical    This macro can be used to abort daily update
Use:       cycles based on an extended checkpoint examination
           algorithm.  For example, the daily CICS format
           routine would end normally if data from at least
           one CICS system were successfully processed.
           However, you might not want the database to be
           updated if CA MICS has not processed data from the
           production CICS system.  This exit could be used
           to examine all the entries in &ADMX..CKPTDATA to
           look for the CICS production system.  If no data
           has been found, the user exit code could execute a
           SAS ABORT ABEND statement, stopping the CICS daily
           update until the correct input data is available.

           Note:  If you code a user exit ABEND, it is your
                  responsibility to document the ABEND
                  condition and its correction for your
                  operations or quality control group.

Invoked Locations:  This exit macro is referenced by the
           checkpoint macro CKRTN4.  CKRTN4 is a Macro
           statement called _CKRTN4 located in:

             sharedprefix.MICS.SOURCE(#CHECKPT)

           and a Macro language macro in:

             sharedprefix.MICS.MACAUTOS(CKRTN4)

           The CKRTN4 macro is referenced by the DAILY job
           steps for each component.

           The following coding example illustrates how you
           could ensure that at least 10 records from both
           the production and test CICS regions were kept.
           The element names to use with this exit can be
           identified by doing a SAS PROC CONTENTS on the
           DETAIL file CKPTDATA.

             MACRO _USRIHL
             IF ROUTINE EQ 'DYCICFMT' THEN DO;
               RETAIN CICP CICT;
               IF COMPT EQ 'CIC CICP' AND
                  KEPT GE 10 THEN CICP = 1;
               IF COMPT EQ 'CIC CICT' AND
                  KEPT GE 10 THEN CICT = 1;
```

```
    IF EOF THEN DO;
      FILE PRINT;
      IF (NOT CICT) THEN
        PUT "INSUFFICIENT DATA FOR TEST REGION" /
            "DAILY UPDATE RUN ABORTED";
      IF (NOT CICP) THEN
        PUT "INSUFFICIENT DATA FOR PROD REGION" /
            "DAILY UPDATE RUN ABORTED";
      IF NOT (CICP AND CICT) THEN ABORT ABEND;
    END;
  END;
  %
```

Exit Name: _USRSTEP

Purpose: Provide user step name for restart messages

Method: The macro is referenced in code that prints the restart message on the CA MICS run status report. See the CA MICS PIOM, Sections 4.3 and 4.3.1, for information about the content and use of the report.

Typical Use: This macro can be used to ammend the restart instructions given by the CA MICS Run Status Report. Refer to the OS Step Names information in Chapter 2 of the Planning, Installation, Operation, and Maintenance Guide for information about using Generation Control Language to change CA MICS operational job step names, including instructions for coding this exit.

Invoked Locations: This exit macro is referenced by the CA MICS run status report program, located in:

sharedprefix.MICS.SOURCE(RSTATUS)

Exit Name: USRSPC

Purpose: Perform data selection and element manipulation to the ADMSPC file.

Method: The macro is referenced before the output of observations to the ADMSPC file. This exit starts with a "%MACRO" statement and ends with a "%MEND" statement. The exit is invoked as %USRSPC.

A typical implementation of this method follows.

```
%MACRO USRSPC;
  IF SPCSTEP = 'DAY030' THEN DO;
    (exit code here)
  END;
%MEND  USRSPC;
```

Typical Use: This macro can be used to remove unwanted data from the ADMSPC file or alter data element content. If the exit sets the program variable SKIP_REC to a value of one, the current observation is not written to the ADMSPC file. While the ADMSPC file is a spin file, only new observations are presented to the USRSPC exit. Existing observations in the ADMSPC file are not affected.

To confine your exit code to a particular step, test the SPCSTEP data element which contains the step name of the CA MICS Operational job step that is executing. The ADMSPC file is updated in the DAILY and Incremental Update job steps.

Exit Name: USRZONE

Purpose: Override or expand ZONE definition options

Method: The macro is referenced in code that sets ZONE values. This exit starts with a "%MACRO" statement and ends with a "%MEND" statement. The exit is invoked as %USRZONE.

Typical Use: This macro can be used to alter or augment ZONE values. ENDTS (End Time Stamp), ZONE, HOUR and SYSID are available at the point of invocation.

Great care must be exercised in using this facility as misuse can corrupt the database, increase DASD space requirements, and/or impact chargeback or capacity planning activities as noted here:

- Standard CA MICS ZONE definitions are based on hour boundaries, for example 8 AM to 5 PM Monday through Friday. Some organizations might want to define CA MICS ZONEs according to a work schedule based on 30 minute or 15-minute

boundaries. The %USRZONE exit can be used to redefine the CA MICS ZONE on sub-hour boundaries per the value of TIMEPART(ENDTS). However, be aware of the following issues and considerations:

- Defining ZONE on a sub-hour boundary requires that ZONE be added as a sequence/summary control element for the DAYS timespan in each CA MICS file. ZONE is not typically a sequence/summary element at the DAYS timespan because the standard ZONE definition ensures that all records for a YEAR, MONTH, DAY, and HOUR are assigned to the same ZONE. But, if you redefine ZONE on a sub-hour basis (without adding ZONE to the summarization key), data in the DAYS timespan will contain corrupted ZONE values because some DAYS time span records will contain composite records for an HOUR which at the DETAIL timespan reflected more than one ZONE. Data in the WEEKS and MONTHS timespans (which is summarized by ZONE), will also be corrupt because the MONTHS timespan is derived from the DAYS timespan data (when DAYS is active).

- Defining ZONE on a sub-hour boundary (along with appropriate DAYS timespan summary key updates) enlarges the CA MICS database. For example, the standard DAYS timespan cycle contains 24 observations for each SYSID - one for each hour. However, with a sub-hour ZONE definition, there are two observations in the database for those hours that cross the ZONE definition boundary. Likewise, because the WEEKS timespan is summarized by ZONE and HOUR, more observations appear in WEEKS timespan files.

o Some organizations might have structured their workloads such that the "prime shift" period varies by SYSID. For example, if one LPAR processes internal production activities such as accounts receivable, shipping, and so on, with a normal 8 to 5 prime shift, while another LPAR is dedicated primarily to the company's WEB server for direct customer order and inquiry where peak workloads arrive after 6 PM,

then different ZONE structures might be appropriate. Address this requirement by splitting data for these different functional systems into individual CA MICS unit databases, where each can have a unique ZONE definition. Or, if data for these unique systems is maintained in a single unit database, the ZONE definition could be structured with unique ZONE values for each of the "prime shift" periods (e.g., ZONE 1 is 8 to 5, ZONE 2 is 6 PM to midnight, ZONE 3 is the 5:00 hour, and so on. An alternative is to use the %USRZONE exit facility to define ZONE 1 uniquely by SYSID. However, this can lead to problems or confusion, or both, in reporting and analysis. For example, attempting to report grand total workload (say for financial planning) for the day time period by selecting data for ZONE 1 actually combines data from two separate time periods. Likewise, activity graphs by HOUR can report unexpected workload gaps if ZONE is included in data selection criteria.

Invoked Locations: This exit macro is referenced by the code that sets the ZONE values and is located in:

sharedprefix.MICS.MACAUTOS(ZONE)

The %ZONE macro is referenced by the DAILY job steps for each component. It can also be invoked at other locations where processing routines compute or re-compute the ZONE value.

4.3.2.2 Standard File and Accounting Exits

Exits that apply to only one file in a single CA MICS product can be used to modify the contents of records on that file immediately before the current observation is output.

Standard file exits can serve any purpose that could be fulfilled at the individual file level. Some of the possibilities are:

- Calculating values for new data elements that are to be added to the file.

- Scaling data elements that are currently on the CA MICS file in question according to some user algorithm.

- Performing the calculations necessary to implement an installation algorithm.

These exits can be used to install a user-written accounting subsystem into CA MICS.

Installing user code to apply to all database units in the CA MICS database complex is a matter of adding the code directly into the exit macro in the #cccEXIT member on sharedprefix.MICS.SOURCE.

Installing user code to apply to a single database unit is a matter of redefining the macro and adding the code directly into the exit macro in the #cccEXIT member on prefix.MICS.USER.SOURCE.

The exit described here resides on the CA MICS source code library sharedprefix.MICS.SOURCE in several different members, which are denoted by fff, a three-character file identifier. A general description of the USRSfff exit and its functions follows.

Exit Name: _USRSfff, where fff is a three-character file identifier.

For example, the CA MICS Hardware and SCP Analyzer (RMF) has an information area called the Hardware Information Area (HAR). The CPU Information File in the Hardware Information Area is called HARCPU and exists in the CA MICS database in timespans and cycles such as DETAIL.HARCPU01. The standard file exit for this file is _USRSCPU, which is located in the #RMFEXIT member of sharedprefix.MICS.SOURCE.

Purpose: Perform calculations immediately prior to file output.

Method: The macro is referenced in code that processes input data immediately before the SAS OUTPUT statement for the associated file. If the file exists on the CA MICS database in the DAYS timespan but not in the DETAIL timespan (such as the CICCSU file), this exit is taken immediately before what would have been the DETAIL observation is summarized to the DAYS level. The routines that reference these exit macros are daily component format routines (DYcccFMT).

Before invocation, all data elements on the file have been valued by CA MICS. That is, any data element generated to be on the DETAIL timespan or that is summarized to the DAYS timespan of the file can be referenced by the exit.

These exits are invoked in one place each. There may be more than one such exit invoked in any single SAS DATA statement, because some data statements build more than one CA MICS file.

Usage Notes: These macros give you great flexibility in changing the contents of CA MICS files. Such changes may be very desirable at your installation. However, modifications also increase the degree of maintenance work you must perform for CA MICS.

Coding these exits assumes more than just a brief introduction to CA MICS. It would be a good idea to study the code in the format routine from which the exit will be invoked. Such items as the choice of names for "temporary" program variables can be absolutely critical to the normal operation of the rest of the component's format routine. Inappropriate choices of variable names can destroy data elements in other files, disturb the CA MICS checkpoint mechanism, and generally delay CA MICS data availability. Thorough testing is essential.

Invoked Locations: These macros are referenced in a single location by a single component. See the individual product guides for more information on _USRSfff exits.

4.3.2.3 JCL Generation Output Exit (JCLEXIT)

The JCLEXIT facility enables you to tailor items produced by the CA MICS JCLGEN process. The exit is used to make changes that are not possible with the basic or extended JCLGEN facilities. Before writing a JCLGEN exit, you should:

- Ensure that there is not an existing JCLGEN facility that can be used to accomplish the tailoring.
- Discuss the design with the CA MICS Product Support Group before undertaking the actual work.

The JCLGEN process creates IEBUPDTE streams in temporary work files that are used to update the CA MICS libraries. If the generation has been successful, macro JCLUPDT is invoked to process the IEBUPDTE stream and update the appropriate CA MICS library. The user exit macro JCLEXIT is invoked by the JCLUPDT macro.

To use the JCLEXIT facility during a complex-level generation and all unit-level generations, code the JCLEXIT macro in sharedprefix.MICS.SOURCE(#USRMAC). If you want to invoke the JCLEXIT macro for one specific unit database, code it in prefix.MICS.USER.SOURCE(#USRMAC).

The macro to be coded is a SAS Macro Language direct exit of the form:

```
%MACRO JCLEXIT;

   /* user statements go here */

%MEND JCLEXIT;
```

If present, the macro executes in-stream within the JCLUPDT function. It is placed so that it has an opportunity to examine each record before it is added to the CA MICS data set. The following SAS variables are defined at the time the exit receives control:

- &COMPLEX - A SAS macro variable that will have a value of COMPLEX if the JCL generation is being performed at the COMPLEX level and a null value if the JCL generation is being performed at the unit level.
- &GENLIST - A SAS macro variable that contains the name of the MICS.PARMS GENLIST member.

```
LCLLIBMM - A 16-byte variable.  The first 8 bytes are the
           DDNAME of the library to be updated, the last
           8 bytes are the member being created.

LCLLINE  - The 80-byte record about to be put into the
           member in the output library.

WRKDDN   - The DDNAME of the library to be updated.

             TCLIST   - sharedprefix.MICS.CLIST
             TGENLIB  - sharedprefix.MICS.GENLIB
             TMACAUTO - sharedprefix.MICS.MACAUTOS
             TSOURCE  - sharedprefix.MICS.SOURCE
             OUTUSOR  - prefix.MICS.USER.SOURCE
             TCNTL    - sharedprefix/prefix.MICS.CNTL
             TPARMS   - sharedprefix/prefix.MICS.PARMS
```

IMPORTANT NOTE:

The JCLUPDT function, which invokes the JCLEXIT macro, is used by the JCLGEN process to update all libraries. It is IMPORTANT that the value of the variable WRKDDN be checked to ensure that user updates only occur for the CNTL and CLIST libraries.

The following sample JCLEXIT macro illustrates how you can ensure that each generated JCL statement contains only uppercase characters:

```
%MACRO JCLEXIT;
  IF WRKDDN = 'TCNTL' THEN LCLLINE = UPCASE(LCLLINE);
%MEND JCLEXIT;
```

4.4 SAS Functions and Call Routines

There are SAS Functions and Call Routines that are part of CA MICS and are supported by CA and not SAS Institute. CA-supported Functions and Call Routines can be identified by browsing the sharedprefix.MICS.LOAD library.

The module names are prefixed with 'UWU' and suffixed with up to five characters of the Function or Call Routine name as specified on a SAS language statement. For instance, the following SAS language statement references the UWUGBLFC Call Routine in sharedprefix.MICS.LOAD:

```
CALL GBLFCN('GET', 'WIDGET', WIDGETVL);
```

4.4.1 SAS Call Routine GBLFCN

```
Name:            GBLFCN

Abstract:        Provides for the storage and retrieval of
                 CA MICS global variables. These variables
                 persist beyond the SAS Data step in which
                 they were created and are destroyed at
                 the end of the Job step.

External
Specifications:  CALL GBLFCN(action, name, value);

    The variables action, name, and value are character
    variables defined as follows:

    action - action to perform on CA MICS global variables.
             Valid values are:

             "SET"   To create or replace the value
                     associated with a variable name.

             "GET"  To retrieve the value associated with a
                     variable name.  If no match is found,
                     blank is returned.

             "PRINT" To print all variable names and their
                     associated values on the MICSLOG.

    name   - literal or variable containing a 1- to
             16-character name to be associated with a value.

    value  - a 1- to 200-character variable or literal.
             If action="SET", then value=a tag of "name".
             If action="GET", then value=the value
             associated with "name".
```

```
Sample usage:

            data _null_;
              ARG3='RMF SMF TSO CICS';
              call GBLFCN('SET', 'COMPTS', ARG3);
            run;

            data _null_;
              length VALUE $200;
              retain VALUE ' ';
              call GBLFCN('GET', 'COMPTS', VALUE);
              put VALUE=;
            run;

       The output will be:

            VALUE='RMF SMF TSO CICS'

Usage notes:

The most common coding error associated with GBLFCN is having
name or value truncated because the variable for name or
value is too short.  To avoid this problem, make the name
variable's length 16 and value variable's length 200.

SAS will give a "VARIABLE NOT INITIALIZED" message for
variables that are only set with CALL functions.  To avoid
these messages, RETAIN the value variables with a value.
```

4.4.2 SAS Function and Call Routine MALABEL

```
Name:           MALABEL

Abstract:       Provides dynamic access to the value of a
                label assigned to a SAS variable.

External
Specifications: CALL MALABEL(value, name); /*Call Routine */
                   -or-
                value=MALABEL(name);       /*Function     */
```

```
    The variables value and name are character variables
    defined as follows:

    value - a 1- to 200-character variable that contains the
            value of the label associated with SAS variable
            whose name is in "name".

    name  - variable containing the 8-character name of a SAS
            variable.

Sample usage:

              data _null_;
                ...
                length VARLABEL $40 VARNAME $8;
                retain VARLABEL VARNAME ' ';
                label ASID = 'Address Space Identification';
                VARNAME='ASID';
                call MALABEL(VARLABEL, VARNAME);
                put VARLABEL=;
              run;

    The following would be printed on the SAS log:

       VARLABEL=Address Space Identification
```

4.5 User-defined Informats and Formats

To validate the contents of a SAS variable by looking for specific values, you can create a SAS informat or format to search a table in a SAS DATA step rather than coding numerous lines of SAS code. If the permissible values used to build the format table are contained in a sequential or partitioned data set, SAS code can be written to input the file that contains the information and create the necessary PROC FORMAT SAS code to build the SAS format.

The SAS DATA step that performs the table search process would contain only a few lines to use the PUT function to perform the table search and, based on the search results, execute specific SAS statements like creating an observation for all non-matching conditions in a CA MICS file. A SAS format can also be used to translate the value in a SAS variable into another meaningful value, such as deriving an account code value based on the job name or user identification value.

SAS Format Catalogs

SAS stores the output of PROC FORMAT in a special member type of a SAS data library called a catalog. SAS data libraries that contain catalogs are commonly called object libraries.

A SAS object library can contain any number of format catalogs. These formats can be either temporary or permanent. Temporary formats are stored in a catalog on the WORK data set and are available only for the life of the SAS job step. Permanent formats are stored in a permanent SAS object library and are available to multiple job step executions.

CA MICS supports three specific format catalogs to store permanent informats and formats at both the complex and unit levels. The catalog names and uses are defined as follows:

USERFMT1 - To store user-defined informats and formats that override standard CA MICS formats. Entries should be placed in this catalog with extreme caution. Formats placed in this catalog with the same name as standard CA MICS informats and formats will be found first and used in place of the standard ones. This may cause unpredictable results and should only be done with the assistance of Technical Support.

USERFMT2 - To store user-defined informats and formats that augment standard CA MICS formats. We recommend that you store all user-defined informats and formats here unless specific circumstances require that you override a standard CA MICS format.

MICSFMTS - To store informats and formats distributed with CA MICS or defined by standard CA MICS facilities during the installation and maintenance processes. User-defined informats and formats should NOT be placed in this catalog.

Complex-level format catalogs are stored in sharedprefix.MICS.MCOLIB and are available in all standard CA MICS cataloged procedures. Unit-level format catalogs, stored in prefix.MICS.MUOLIB, are available in all standard unit-level CA MICS cataloged procedures. In addition, the CA MICS Workstation Facility will provide access to the appropriate object libraries.

CA MICS processes define the search order for these format catalogs through the use of the SAS OPTIONS statement. The WORK.FORMATS format catalog is always defined first in the search order and is typically followed by the MCOLIB format catalogs, which are succeeded by the MUOLIB format catalogs, and finally the SAS format catalog. Utilizing the USEROBJLIB JCLDEF/JCLDEFC option would add your user-defined formats catalog to the end of the search order. This search order may vary by the process or job being executed.

```
For example,
OPTIONS FMTSEARCH=(WORK.FORMATS MCOLIB.USERFMT1
MCOLIB.MICSFMTS MCOLIB.USERFMT2 MUOLIB.USERFMT1
MUOLIB.MICSFMTS MUOLIB.USERFMT2 LIBRARY.FORMATS);

Some processes such as MICF inquiries define the unit level
(MUOLIB) format catalogs ahead of the complex level (MCOLIB)
format catalogs whenever a unit database has been selected.
The order of the MUOLIB definition on the FMTSEARCH option
will coincide with the order that units have been selected.
In other words, the first unit selected in a File Selection
Step for example, would be the first format catalog defined
on the FMTSEARCH option.


Creating User-defined Informat/Format Catalog Entries
-----------------------------------------------------

Since both complex- and unit-level format catalogs are stored
in SAS object libraries, updating these catalogs requires
exclusive use of the libraries (DISP=OLD).  To eliminate
contention problems between batch and TSO users accessing the
CA MICS system, all users acquire a temporary copy of the
CA MICS object libraries (MCOLIB and MUOLIB) automatically
during initial processing.  This means that updates to the
temporary copy of your user format catalogs must also be made
to the permanent SAS object library.  If you use the standard
CA MICS jobs, USERFMTC and USERFMTU, to update your temporary
user format catalogs, then the permanent SAS object library
will be updated automatically.

Note: Remember to monitor the size of the CA MICS object
libraries over time.  The number of user formats created by
your users dictates the amount of space that is optimal for
your site.
```

4.5.1 Creating Complex-level Catalog Entries

The USERFMTC job is available to update the USERFMT1 and USERFMT2 catalogs in sharedprefix.MICS.MCOLIB. The USERFMTC job is automatically generated in sharedprefix.MICS.CNTL. The input to this catalog update job is the following members of sharedprefix.MICS.USER.SOURCE:

```
USERFMT1 - Adds and updates entries in catalog USERFMT1
USERFMT2 - Adds and updates entries in catalog USERFMT2
```

The only code that needs to be entered in these members is the INVALUE, VALUE, and/or PICTURE statements necessary to generate your formats. Please refer to the SAS Procedures Guide for complete syntax rules on coding these input statements to PROC FORMAT. Any number of these statements can be coded in each member. All formats defined in this member will be updated each time the USERFMTC job is executed. An invocation of PROC FORMAT with output to the appropriate catalog is automatically invoked and need not be coded in the input member.

If you have need to perform more complicated processing, such as a DATA step that reads an external file and then generates a format, that code can either be coded in these members or %INCLUDEd from standard CA MICS libraries or private data sets. The only requirement is that you code your PROC FORMAT invocation statements in the following format:

```
PROC FORMAT LIBRARY=MCOLIB.USERFMTn;
    ...INVALUE, VALUE, and/or PICTURE statements...
RUN;
```

Examples of various input methods that can be used in these members follow.

```
Example 1--Coding INVALUE, VALUE, and PICTURE statements to
           be processed by PROC FORMAT.

    INVALUE CHECK
        '1'-'4'=_SAME_
        '99'   =.
        OTHER  =_ERROR_;

    PICTURE DAYS
        01-31='99'
        OTHER='99 - Illegal day value';

    VALUE ABC
        1='A'
        2='B'
        3='C';

    VALUE $TAPES
        '0300'-'032F'='TAPE'
        '0400'-'043F'='CART'
        '0800'-'08FF'='SILO'
        OTHER='????';


Example 2--Reading a sequential file's values for use in a
           table search.

The following code can be used to input a sequential file
containing a list of valid codes for use in a table search
for permissible values of a SAS variable.  The format will be
saved in the USERFMT2 catalog in sharedprefix.MICS.MCOLIB.
This format is referenced later in this section to select
only observations that do not have an ACCTNO1 value in this
format.

In this example, the SAS code inputs the first non-blank
field in each record as the table look-up entry.  If the
first non-blank field in each record is not used as the
look-up value, the INPUT statement must be changed to
properly input the desired values.  The SAS code shown below
creates a format called $USRIDS in the USERFMT2 catalog in
sharedprefix.MICS.MCOLIB.  You may want to change the name of
the format that is created.  The format name must begin with
a "$" if you use a SAS character variable to do the table
look-up in the DATA step PUT function.  If a numeric variable
is used in the PUT function, the FORMAT name must begin with
a letter.
```

```
/* ALLOCATE THE INPUT DATA SET */
FILENAME INPUT 'your.input.dataset.name' DISP=SHR;
DATA _NULL_;
LENGTH TEMP1 $16;     /* MAX LENGTH OF LOOK-UP VALUE */
INFILE INPUT END=EOF; /* INPUT SEQUENTIAL FILE */
FILE FT15F001;        /* TEMP WORK FILE IN MICSDM PROC */
PUT "PROC FORMAT LIBRARY=MCOLIB.USERFMT2;" /
    "VALUE $USRIDS ";
DO UNTIL(EOF);
  /* LOOK-UP VALUES ARE IN THE FIRST NON-BLANK COLUMN */
  INPUT TEMP1 $ ;
  PUT "'" TEMP1 "' = '1'";
END;
PUT "OTHER = '0';";
STOP;
RUN;
%INCLUDE FT15F001;  /* INC. THE GENERATED PROC FORMAT */
RUN;
FILENAME INPUT CLEAR;  /* FREE THE INPUT DATA SET */
```

An alternate method to perform the same process is to use the CNTLIN option of PROC FORMAT. Notice that the format name variable FMTNAME does not have the $ in the retained value. The fact that this is a character format is defined by setting the format type variable, TYPE, to C. The format is still referenced as $USRIDS in a PUT statement.

```
/* ALLOCATE THE INPUT DATA SET */
FILENAME INPUT 'your.input.dataset.name' DISP=SHR;
DATA WORK.CNTLIN;
LENGTH START END $16;   /* MAX LENGTH OF LOOK-UP VALUE */
RETAIN FMTNAME 'USRIDS' /* NAME OF FORMAT TO BE GEN'D */
       TYPE 'C'         /* THIS IS A CHARACTER FORMAT */
       HLO ' '          /* FOR USE WITH OTHER= OPTIONS */
       LABEL '1';
INFILE INPUT END=EOF;   /* INPUT SEQUENTIAL FILE */
DO UNTIL(EOF);
  /* LOOK-UP VALUES ARE IN THE FIRST NON-BLANK COLUMN */
  INPUT START $ ;
  END=START;            /* START/END OF RANGE ARE SAME */
  OUTPUT;
END;
START='**OTHER**';      /* GENERATE AN OTHER= VALUE */
END=START;
HLO='O';
LABEL='0';
OUTPUT;
STOP;
RUN;
/* GENERATE THE FORMAT */
PROC FORMAT CNTLIN=WORK.CNTLIN
            LIBRARY=MCOLIB.USERFMT2 PRINT;
RUN;
FILENAME INPUT CLEAR;  /* FREE THE INPUT DATA SET */
```

If the code in the above example were stored in a private partitioned data set, it could be added to the USERFMTC update process as follows:

```
%INCLUDE 'your.private.pds(member)';
```

SAS will dynamically allocate, read, and free the data set for you.

If the code in the above example was stored in another member of sharedprefix.MICS.USER.SOURCE, it could be added to the USERFMTC update process as follows:

```
%INCLUDE USOURCE(member);
```

A sample SAS DATA step that uses the format $USRIDS to subset a SAS file by selecting observations that have certain ACCTNO1 values is illustrated in the sample SAS code that follows:

```
//MICS  EXEC MICSSHRP
//SYSIN  DD   DATA,DLM=ZZ
/* SAVE ALL OBS. THAT CONTAIN INVALID ACCTNO1 VALUES */
DATA REJECTS (KEEP=ACCTNO1 ACCTNO2 JOB RDRTS);
SET &BATX..BATJOB01;
IF PUT(ACCTNO1,$USRIDS.) = 0  THEN OUTPUT;
RUN;
/* PRINT REJECT OUTPUT REPORT */
PROC PRINT DATA=REJECTS; BY ACCTNO1;
ID JOB; VAR ACCTNO1 ACCTNO2 RDRTS;
TITLE " REJECT LISTING FROM BATJOB FILE ";
RUN;
ZZ
```

The logic in the sample DATA step above will output any observation that does not have one of the user identification (ACCTNO1) values that is contained in the SAS format $USRIDS. This technique can be used to identify and print exceptions or rejects.

4.5.2 Creating Unit-level Catalog Entries

A standard job is available to update the USERFMT1 and USERFMT2 catalogs in prefix.MICS.MUOLIB. This job, USERFMTU, is automatically generated in prefix.MICS.CNTL. The input to this catalog update job is the following members of prefix.MICS.USER.SOURCE:

```
USERFMT1 - Adds and updates entries in catalog USERFMT1
USERFMT2 - Adds and updates entries in catalog USERFMT2
```

The only code that needs to be entered in these members is the INVALUE, VALUE, and/or PICTURE statements necessary to generate your formats. Please refer to the SAS Procedures Guide for complete syntax rules on coding these input statements to PROC FORMAT. Any number of these statements can be coded in each member. All formats defined in this member will be updated each time the USERFMTU job is executed. An invocation of PROC FORMAT with output to the appropriate catalog is automatically invoked and need not be coded in the input member.

If you need to perform more complicated processing, such as a DATA step that reads an external file and then generates a format, that code can either be coded in these members or %INCLUDEd from standard CA MICS libraries or private data sets. The only requirement is that you code your PROC FORMAT invocation statements in the following format:

```
PROC FORMAT LIBRARY=MUOLIB.USERFMTn;
    ...INVALUE, VALUE, and/or PICTURE statements...
RUN;
```

The following examples show input methods that can be used in these members.

Example 1--Coding INVALUE, VALUE, and PICTURE statements to be processed by PROC FORMAT.

```
INVALUE CHECK
    '1'-'4'=_SAME_
    '99'   =.
    OTHER  =_ERROR_;

PICTURE DAYS
    01-31='99'
    OTHER='99 - Illegal day value';

VALUE ABC
    1='A'
    2='B'
    3='C';

VALUE $TAPES
    '0300'-'032F'='TAPE'
    '0400'-'043F'='CART'
    '0800'-'08FF'='SILO'
    OTHER='????';
```

Example 2--Determining the value of one SAS variable based on the value of another SAS variable.

A SAS format can also be used to determine the value of one SAS variable based on the value of another SAS variable. In most simple cases, a SAS DATA step can be coded with a series of IF/THEN/ELSE statements to achieve this result, as shown in the following example.

```
IF USER =: 'PR' THEN TSOACT1 = 'PR25';
  ELSE IF USER =: 'TS' THEN TSOACT1 = 'TS01';
  ELSE IF USER =: 'PL' THEN TSOACT1 = 'PL15';
  ELSE IF USER =: 'RX' THEN TSOACT1 = 'RX25';
  ELSE TSOACT1 = '*UNKNOWN*';
```

If the list of possible values for USER were large or if the possible values changed frequently, it would be more effective and efficient to use a SAS format to assign the value of TSOACT1 based on the first two characters found in the USER variable. When using a SAS format to perform a table search, you do not have to modify your exit routine. You need only replace the SAS format, minimizing coding or syntax errors.

For example, a SAS format could be written to translate a user's logon identifier to a group name for assigning TSOACT1 values. The following code can be used to input a sequential file that contains valid logon identifier prefix values to use for a table search of expected values. If the value present in the USER variable is one that is unknown to the table, a value of "*UNKNOWN*" will be returned from the PUT function when performing the table search in the TSO Account Code Exit Routine (TSOACRT).

The following example shows the SAS format $UIDGRPS being saved in the USERFMT2 catalog in prefix.MICS.MUOLIB each time the job in prefix.MICS.CNTL(USERFMTU) was executed. This code could be added to prefix.MICS.USER.SOURCE(USERFMT2). The job in prefix.MICS.CNTL(USERFMTU) would need to be rerun whenever the list of ID prefix values and group name values changes.

```
/* ALLOCATE THE INPUT DATA SET */
FILENAME INPUT 'user.input.ids' DISP=SHR;
DATA _NULL_;
LENGTH ID $2 GROUP $10 ;
INFILE INPUT END=EOF;         /* INPUT SEQUENTIAL FILE */
FILE FT15F001;                /* USE TEMP WORK FILE */
PUT "PROC FORMAT LIBRARY=MUOLIB.USERFMT2;" /
  "VALUE $UIDGRPS ";
DO UNTIL(EOF);
  /* TABLE LOOK-UP VALUES ARE IN COLUMNS 1-2 AND 10-19 */
  INPUT @01 ID $2. @10 GROUP $10. ;
  PUT "'" ID "' =  '" GROUP "'";
END;
PUT "OTHER = '*UNKNOWN*';";
STOP;
RUN;
%INCLUDE FT15F001; /*INCLUDE THE GENERATED PROC FORMAT */
RUN;
```

An alternate method of performing the same process is to use the CNTLIN option of PROC FORMAT. Notice that the format name variable FMTNAME does not have the $ in the retained value. The fact that this is a character format is defined by setting the format type variable, TYPE, to C. The format is still referenced as $UIDGRPS in a PUT statement.

```
/* ALLOCATE THE INPUT DATA SET */
FILENAME INPUT 'user.input.ids' DISP=SHR;
DATA WORK.CNTLIN;
RETAIN FMTNAME 'UIDGRPS' /* NAME OF FORMAT TO BE GEN'D */
       TYPE 'C'          /* THIS IS A CHARACTER FORMAT */
       HLO ' '           /* FOR USE WITH OTHER= OPTIONS*/
       ;
INFILE INPUT END=EOF;   /* INPUT SEQUENTIAL FILE */
DO UNTIL(EOF);
  /* LOOK-UP VALUES ARE IN THE FIRST NON-BLANK COLUMN */
  INPUT @01 START $2. @10 LABEL $10. ;
  END=START;            /* START/END OF RANGE ARE SAME */
  OUTPUT;
END;
START='**OTHER**';      /* GENERATE AN OTHER= VALUE */
END=START;
HLO='O';
LABEL='*UNKNOWN*';
OUTPUT;
STOP;
RUN;
/* GENERATE THE FORMAT */
PROC FORMAT CNTLIN=WORK.CNTLIN
            LIBRARY=MCOLIB.USERFMT2 PRINT;
RUN;
FILENAME INPUT CLEAR;  /* FREE THE INPUT DATA SET */
```

If the code in the above example was stored in a private partitioned data set, it could be added to the USERFMTU update process as follows:

```
%INCLUDE 'your.private.pds(member)';
```

SAS will dynamically allocate, read, and free the data set for you.

If the code in the above example were stored in another member of prefix.MICS.USER.SOURCE, it could be added to the USERFMTU update process as follows:

```
%INCLUDE USOURCE(member);
```

The SAS code above assumes that the sequential file contains the table look-up entries in columns 1 and 2 and the associated Group Names in columns 10 through 19. You should change the INPUT statement to identify the columns that contain the desired values.

A sample CA MICS TSOACRT routine that uses the format $UIDGRPS is illustrated in the sample SAS code below.

```
IF ROUTINE='DYTSOFMT' THEN DO;
  TSOACT1 = PUT(SUBSTR(USER,1,2),$UIDGRPS.);
  ... other SAS statements ...
END;
```

The table $UIDGRPS is searched for the value contained in the first two positions of the USER variable and the search result is placed in the TSOACT1 variable.

4.5.3 Creating Catalog Entries During Operational Jobs

CA MICS generation processes like JCLGEN and cccCGEN use the LOCKIT data set to ensure that jobs that update Format catalogs do not backlevel each other. The CA MICS operational jobs cannot use this same serialization technique because it would prevent multiple units from being updated at the same time.

If you have a requirement to create a permanent SAS Format during execution of one of the CA MICS operational jobs, like DAILY, the following technique can be used:

- Create a special SAS data library in which to store your Formats.

- Add a LIBRARY DD with DISP=OLD to the operational steps that require update access to the Format catalog.

  Example:

  ```
  //DAY030 EXEC MICSDBP
  //LIBRARY  DD DISP=OLD,DSN=MYLIB.USER.FORMATS
  //INPUTSMF DD .....
  ```

- Update your user code that generates the permanent Format to direct PROC FORMAT to output Formats to the LIBRARY DD by adding the LIBRARY=LIBRARY.FORMATS option.

  Example:

  ```
  PROC FORMAT LIBRARY=LIBRARY.FORMATS;
  ```

- Add a LIBRARY DD with DISP=SHR to any other job steps that require read access to the Format catalog.

```
Example:
  //DAILYRPT EXEC MICSSHRP
  //LIBRARY  DD DISP=SHR,DSN=MYLIB.USER.FORMATS
  //SYSIN    DD .....
```

```
CA MICS will automatically add the LIBRARY.FORMATS catalog to
the SAS FMTSEARCH option for you.  Please note that
LIBRARY.FORMATS is searched after MCOLIB and MUOLIB, so the
names of the Formats placed there must not exist in any of
the other standard Format catalogs.  Also, the catalog name
must be FORMATS or it will not be searched.
```

4.5.4 Input/Output Formats

The following table describes CA-written input and output formats.

INFORMAT NAME	ABSTRACT
MAKB	This informat reads positive integer binary data representing bytes and converts the value into kilobytes through division by 1024. The input length must be four.
MAKBRB	This informat reads real binary (floating point) data representing bytes and converts the value into kilobytes through division by 1024. The input length must be four.
MAMSEC	This informat is used to convert z/OS clock values to seconds. The entire 8 byte clock value is divided by 4,096,000,000. The low order byte is dropped so that the value can be moved into a doubleword for normalization. This effects a division by 256. The normalized value is then divided by 16,000,000. (256*16,000,000=4,096,000,000). After conversion to seconds, scaling is performed if required.
MAPCT	The MAPCT format takes a number and formats it with two decimal places and adds a percent sign on the end. For example if you have two numbers 99 and .5, the result after using the MAPCT. format would be 99.00 % and 0.50 % respectively.

See the individual component guides for information on INFORMATs and FORMATs delivered by a specific component.

4.6 Documentation Programs

This section describes the system used for production of a printed copy of a CA MICS document. Parts of the same system are also used by the online Document Access facility. Both applications must be considered if any changes are being made to the system.

A CA MICS document is built from a set of members found in one or more of the CA MICS TEXT libraries. These TEXT libraries contain the text, figure, and title members that are used to compose the CA MICS guides. The libraries contain variable length records, with a maximum text width of 133 characters (LRECL=137,RECFM=VB). The CA MICS TEXT libraries are:

- sharedprefix.MICS.DOC.TEXT, which contains document text sections, figures, picklists, and title pages.

- sharedprefix.MICS.INC.TEXT, which contains generated text and other members copied by several documents.

- sharedprefix.MICS.DIC.TEXT, which contains the Data Dictionary data element descriptions.

Text that is printed in two column fashion must not exceed a width of 61 characters. This text is referred to as "short text." Figures, titles, and other text using the entire page width as a single column may be as wide as will fit on the printed page (usually 130 characters). This text is referred to as "long text."

Data Dictionary elements reside in a separate library so that they can follow their own separate naming convention,as determined by other CA MICS conventions. These members are always developed as short text.

PICKLISTS

The files that control the assembly of text members into documents are called picklists.

The structure for a guide relies on a book-level picklist that calls individual chapter picklists. The chapter picklists identify the individual files that constitute the chapter. This structure allows you to print an entire guide or just one chapter.

The picklist is also the basis of the display used by online Document Browse. The titles displayed by Document Browse are extracted directly from the picklist title entries. The order of title presentation is also determined by the picklist.

The set of guides presented to the Document Access user is determined by an ISPF table member, DOCLSTP4, that lists the available guide titles and document identifiers in a standard order. These entries identify the book level picklists. All picklists should reside in the DOC.TEXT library by convention. DOCLSTP4 is constructed from a special picklist named PICVOL.

DOCUMENT PRODUCTION PROCESS FLOW

Producing a printed copy of a document is a two-step process. The first step selects the members from the TEXT libraries and produces a single text file containing a rough form of the document. The second step formats the rough copy, adds tables of contents if requested, and produces the final copy with either one or two columns of text to a page.

The entire process is controlled by command records identified by a "$" in column one of the record. These records, called "dollar commands" in this description, are used in both steps of the process. In the first step, dollar commands are read from a picklist. The picklist is a DOC.TEXT library member which usually consists entirely of dollar commands. It lists the text members that make up the document in the order in which they are to appear. The first step program (DOCPICK) reads the list and copies each member named to the rough text file.

Dollar commands are also used to control the formatting process in the second step. These records may be contained in either the picklist or in the section and figure text members. These formatting dollar commands are detected by DOCPICK, but passed to the second step through the rough intermediate file. The formatting controls are used by the second step program, sharedprefix.MICS.SOURCE(DOCUMENT). DOCUMENT produces the final document with two columns of text to a page.

4.6.1 Document Utility Control Commands

The document facilities support a document formatting language (called dollar commands) that allows you to assemble a printed document from one or more text files with control over pagination, page numbering, page size, and layout. The content and format of the printed document is controlled by dollar command lines within the text files. CA MICS documents are built from outline files, called picklists, that contain dollar commands for assembling a group of text files and figures in sequence.
DOLLAR COMMANDS

The processing of the document production programs DOCPICK and DOCUMENT is controlled by records in the text which contain a "$" in column 1. This section describes the use of each dollar command.

The syntax for the dollar commands is:

```
$xxxx nnn nnn
```

where:

- the $ must be in column 1.
- xxxx is the command name, and must begin in column 2.
- nnn is an operand. Many commands have no operands, but if present they are column dependent. Most operands begin in column 7.

Commands that are not recognized are treated as comments and are not printed.

4.6.1.1 Document Assembly Commands

The commands $COPY, $SMEM, $LMEM, and $DMEM select data set members. $COPY is used in book picklists. $SMEM and $LMEM can appear in a picklist or a text module. $DMEM normally is used only from a picklist, but can also be imbedded within a text member. The description of each command follows. Note that the format of these commands is fixed. The document assembly dollar commands are:

$COPY mmm...

When $COPY appears in a picklist, it copies in another picklist. In that case, mmm... has the value PIC... $COPY statements can be nested up to 10 levels deep.

$COPY is used only for picklists. The commands $SMEM and $LMEM replace the old $COPY #INS.... and #INL.... and thus allow any member name to be referenced in a copy operation, as described below.

$DMEM mmm... ttt...

Names a DIC.TEXT (data dictionary) member (61 characters wide) to be included in the output file. "mmm..." is the member name (data element name) and "ttt..." is the long data element name; ttt... can contain up to 40 characters.

$LMEM mmm... ttt...

Names a long text member (up to 130 characters wide) to be included in the output file. This command should normally be imbedded within a text member, but is also used in a picklist to copy title pages. "mmm..." is the member name, and "ttt..." is the figure title that is maintained for easy identification of the member contents.

$SMEM mmm... ttt...

Names a short text member (normally up to 61 characters wide) to be copied to the output document file. This command is used in picklists to identify text sections that make up a larger document. This command can also be imbedded within a text member to copy additional text into the output document. "mmm..." is the member name, and "ttt..." is the descriptive section title that appears in the online Document Browse menu.

Note: The document assembly commands are fixed format. These commands expect the member name field to occupy 8 character positions. Member names should begin in column 7 and titles should begin in column 16. Up to 57 characters of title information will be displayed by online Document Browse or printed in the hardcopy tables of contents.

4.6.1.2 Text Formatting Commands

The text formatting dollar commands are:

$CENT

Signals the start of 61-character lines which are to be centered. Centering means that the line starts in column 35; it is used only within figures. $REST turns off this command.

$CENT nnn

Center text assuming it is nnn columns wide, within 132 character print positions.

$LONG

Signals the start of 130 character lines of text. $REST turns off this command.

$NORM

Reset from $LONG but allow for two up printing under a figure (on the same page). Use a $REST at the end of a section that issues a $NORM after a figure to ensure that a $HEDR starts on a new page, not just a new column. This prevents a succeeding text section from beginning under a figure.

```
$COL

   Used to insert 6670 commands into the text output.

$PAGE lll nnn

   Sets a footer page label and the initial page number
   value.  "lll" is an optional three-character label or
   chapter number.  Option nnn is a three-digit page number;
   when you use it without lll, it must begin in column 11.
   Leaving lll and nnn blank turns off page numbering.

$PARM xxx...  xxx...  ...

   Sets parameters that affect document format.  These are
   discussed later under the heading, "$PARM Command
   Parameters."

$REST

   Reset.  Marks the end of either a $LONG or a $CENT
   section.

$SCOL

   Skip to the top of the next column, unless that is the
   current location.  If the current column is within column
   2, or a single column document is being produced, this is
   equivalent to $SPAG.

$SPAC nnn

   Skip "nnn" lines.  If the skipping runs over the column
   boundary, only the number of lines left in the column are
   skipped.

$SPAG

   Skip to the top of a new page, unless that is the current
   location.

$* comment

   Comment line; it is not printed.  Any command that is not
   recognized is also treated as a comment.
```

4.6.1.3 Heading and Table of Contents Commands

The heading and table of contents commands trigger section heading titles or table of contents entries.

$BOOK 28-character-bookname

Signals MICSDOC to print the abbreviated name of the document, as provided in the 28-character book name. This name will appear on the bottom of two up pages to the left of the chapter-page number. For example:

```
$BOOK CA MICS Intro Guide
```

causes "CA MICS Intro Guide" to be printed at the bottom of each guide page. By convention, the preferred place for these commands is in the ii$TTL member for each guide.

$CPYR yyyy

Supplies the document copyright date in the form of yyyy. This is the date used in the copyright notice printed at the lower left corner of the documentation box when printing two up documents such as CA MICS guides.

$DMEM mmmmmmmm ttt...

This dollar command assembles text members and also creates section heading information. It identifies a data dictionary entry and is used to generate the detailed table of contents entry. "mmmmmmmm" is the member name of the element, and "ttt..." is the descriptive name of the element. "ttt" may contain up to 40 characters.

$FIG Figure n-nn. Title

This command prints a title underneath a figure and also provides an entry for the figure table of contents. The number to the left of the hyphen is the chapter number; the number to the right of the hyphen is a sequence number. The sequence number may have periods within it and must have a period at the end.

```
$FIGC Figure n-nn.  Title

   The same as $FIG, except that the Figure n-nn.  Title line
   is centered based on the current line size.  This command
   makes is easier to put a title on a figure, especially
   when the body of the figure is copied using $LMEM.

$HEDR n.n.n - section title
$HEDR Chapter nn.  CHAPTER TITLE

   Signals the start of a new section or chapter, and
   provides the tables of contents entry.  "nn" is the
   chapter or section number.  The number of periods in this
   number defines the level of indentation in the table of
   contents.  The chapter number must end with a period; the
   section number must not.  The section number and title
   should be separated by a hyphen; the chapter number and
   title should not be.  The number, title, and "Chapter" (if
   present) can contain a total of 61 characters.

$HEDE nnn...  ttt...

   Identifies a CA MICS exception type for inclusion in the
   detailed table of contents.  "nnn..." is the number of the
   exception type, and "ttt..." is a one-line description of
   the exception (the long name of the exception).

$PDAT 18-character-date

   Signals MICSDOC to override the value in the print date
   field using the text provided in the 18-character-date
   field.  This string will appear on the bottom of 2UP pages
   to the right of the chapter-section number.  Example:

              $PDAT Oct 1989 Level

   causes "Oct 1989 Level" to be printed at the bottom of
   each Guide page instead of using the actual print date.
   By convention, the preferred place for these commands is
   in the @ii$TTL member for each guide.

$$TOC

   Marks the point in the text at which the tables of
   contents are inserted.  If more than one $$TOC is
   encountered, the tables of contents are inserted only
   where the first $$TOC occurs.
```

4.6.1.4 $PARM Command Parameters

```
$PARM nnnn nnnn ...

This control record sets parameters that affect document
format.  These parameters can also be set using the SYSPARM
option of the MICSDOC PROC, but only if the parameter affects
the entire document.  Parameters set with the $PARM card take
effect when they are encountered in the input text.  The
values of the parameters can be changed as often as
necessary.  Several parm values may be specified on the same
$PARM card.  Each must be separated by a blank or a comma.

   The parameters currently supported include:

    o  1UP/2UP

       Allows selection of one or two columns of text per
       page.  If encountered within the text, this parm will
       take effect at the end of the current column of text.
       The default value is 2UP.

    o  UCASE

       Requests translation of "|" to "I" for the page border
       box.  This parm is intended to be used in conjunction
       with the UCASE option of the MICSDOC PROC, and is set
       automatically when this option is specified on the
       EXEC statement.  The UCASE option on the EXEC
       statement causes all of the input text to be
       translated to upper case.

    o  LSIZE=nnnn

       Changes the line size of the output text.  This
       parameter is intended primarily for use in one-column
       text.  It does not dynamically flow text across line
       boundaries.  The default value is 61.  When set to 80,
       column 1 is treated as part of the text.
```

o NLINES=nnnn

Changes the number of lines of output per page. This parameter is intended primarily for use in sites that use a paper size that will not hold 60 lines. The minimum value allowed is 20. MICSDOC reserves lines at the top and bottom of each page which are included in the NLINES count. Three lines are reserved at the top of the page. The number of reserved lines at the bottom of a page varies with the type of text on each particular page. MICSDOC reserves 5 lines for text, 2 lines for figures, and 10 lines for dictionary entries. Thus, the default NLINES value of 60 allows for 52 text lines with text type output.

o SNUM

Signals the DOCUMENT program to suppress printing of the section number (from $HEDR entries) as part of all GENERAL and DETAIL Table of Contents lines. Without this option, the format of table of contents lines is:

```
x.x.x.x  title information . . . . . page#
```

where x.x.x.x is the document section number. Using this option, the format of table of contents lines is:

```
title information  . . . . . page#
```

o NONUM

Signals the DOCUMENT program to suppress printing of the section number (from $HEDR entries) as part of all text page headers and footers. This option includes SNUM features and additionally strips section numbers from the beginning of each text section. It also blanks out the section number that would normally print at the bottom of the page box on a two-up print. The NONUM option was added to allow for text to be printed without any numbering except chapter and figure numbers.

o CNTL

This option requests that ./ control record lines be ignored if present in the text.

4.6.2 DOCPICK Program Documentation

DOCPICK, an assembly language program, is the first step in the process of printing one of the CA MICS guides. It combines text members which contain the document sections, figures, and title pages to produce a rough version of the final text. This combination is done under the control of a picklist which names, in sequence, the members that make up the text. The output from DOCPICK is input to the formatting program DOCUMENT. This program produces the final version of the text. DOCUMENT is discussed in Section 4.6.3.

The files used by DOCPICK are:

- DOCSTRM

  This input file contains the picklist members.

- DOCUMENT

  This input file contains the document text members selected with the $SMEM and $LMEM commands.

- DATADICT

  This input file contains data dictionary members selected with the $DMEM command.

- DOCOUT

  This is the principal output file. It contains the members selected from DOCUMENT and DATADICT under the control of the picklists in DOCSTRM.

- SYSTERM

  This output file is used for error messages. It is optional, and if it is not present, messages are discarded.

- DOCIN

  This is an optional sequential input file. If present, it is treated as the first level of input, equivalent to the DOCSTRM file. This file is used mainly in batch as an instream file for ad hoc print requests. This file is, in effect, a sequential picklist, one level above the DOCSTRM file. If present, it overrides the M= exec parm that normally specifies the starting level picklist.

```
DOCPICK RETURN CODES

    0 - Everything OK
    1 - at least one member was not found
    2 - no output records or table rows produced
    4 - Error opening files
    8 - EXEC parm error, see systerm message
   12 - VCOPY string field too long
   16 - File blksize problem
   20 - SYNAD routine entered, I/O problem
   24 - Bad problem, program issued quit without diagnosis
   28 - VDELETE failed
   32 - WILD Card Specification Error
   36 - Unsuccessful load of ISPLINK
   40 - VCOPY or VDEFINE error

DOCPICK PROCESSING

The operation of DOCPICK can be described as follows:

DOCPICK reads the picklist looking for four types of control
cards.  They are:

  - $COPY (copy a picklist member)
  - $SMEM (short text member)
  - $LMEM (long text member)
  - $DMEM (DIC.TEXT member)

DOCPICK processes each card as follows:

 - $SMEM

   The member is copied from the DOCUMENT file.  The control
   card is discarded.

 - $LMEM

   $SPAG and $LONG control records are written to DOCOUT, the
   member is copied from the DOCUMENT file, and then a $REST
   control record is written to DOCOUT.  Carriage control
   characters are removed and spacing lines added instead.
   The control card is discarded.
```

```
 - $DMEM

   The same action is taken as for $SMEM cards, except the
   control card is first copied to the DOCOUT file, as well.

 - $COPY

   Names a picklist member that is inserted in the text where
   the $COPY card appears.  When it appears in a picklist,
   the copied member is treated as an insertion to the
   picklist rather than to the text being built.

   $COPY, $SMEM, $LMEM, and $DMEM statements can be nested up
   to 10 levels deep.

All other $xxxx control cards are simply copied to the DOCOUT
file.

DOCPICK is normally executed in conjunction with DOCUMENT.
The JCL needed to execute both of these programs is the
MICSDOC Proc.


How DOCPICK handles short and long members:

    -  From a picklist (assumed a $SMEM itself)

       To an $SMEM or $DMEM - no special processing

       To an $LMEM -  issue a $SPAG, then a $LONG before
                      copying
                  -  issue a $REST after copying

    -  During transition within a text member (nested copy)

       From an $SMEM or $DMEM to an $LMEM
          - issue a $SPAG, then a $LONG before copying

       From an $LMEM to an $SMEM or $DMEM
          - issue a $REST after copying

The logic above applies to the $COPY command as well, based
on the rules outlined for how $COPY short/long type is
determined.
```

Special note on LMEMs:

As a convenience, DOCPICK assumes that carriage control characters in column 1 of a long member should be converted to extra line or page skips before passing the text on to the DOCUMENT program. This makes it easier to plug reports output directly into the documentation. This feature is still supported for the standard processing option, but is suppressed for unload type options, such as CNTL and CTLC.

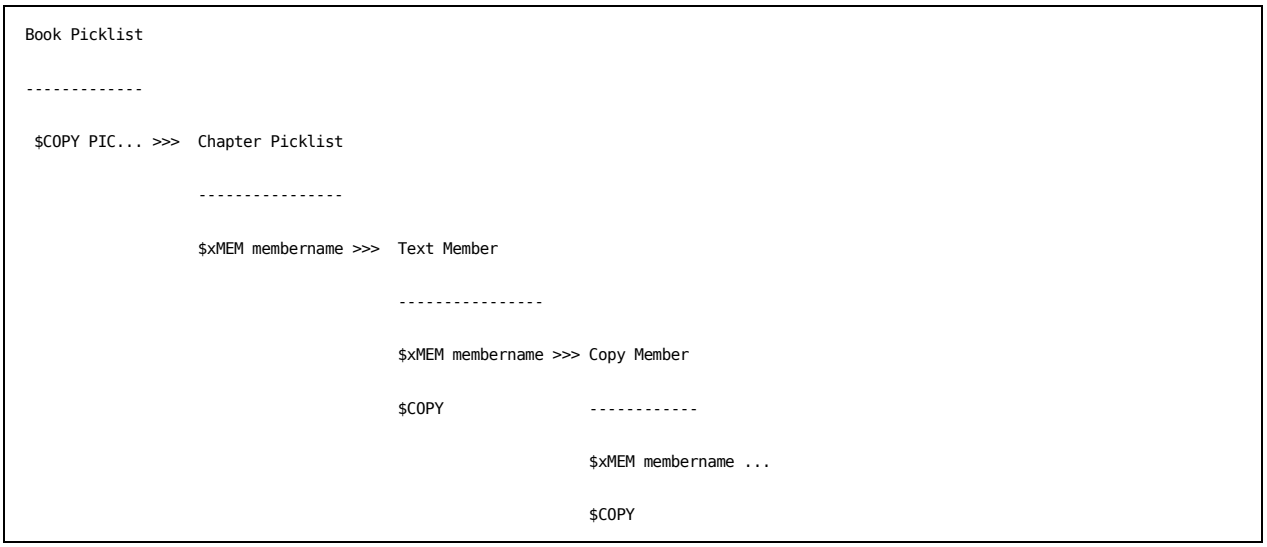


Figure 4-2. DOCPICK Nested Copy Processing

In the figure above, $xMEM refers to SMEM, LMEM, or DMEM. These commands will now allow $COPY-type operations on any member, regardless of its name. LMEMs are treated as 133 characters wide. SMEMs and DMEMs are treated as 80 columns wide, but normally using only 62 columns. For variable length records, the full length of the record is processed, but certain options and formats require short/long text conventions to followed (e.g., 2UP, SN, $CENT, etc.).

$COPY will try to determine the type of member based on #INS and #INL naming conventions. If the fourth character of the name is "S", assume a short member, if the fourth character is "L", assume a long member; otherwise, assume that copy member is the same type as the calling member, meaning don't do anything special about changing page format, just copy the text.

DOCPICK stacks the member type along with the other information stored when a COPY or xMEM command is processed. The types are S, L, or D, which describes the format and also the DD type to use for the copy operation. One parameterized read routine is used. DDnames for each type S,L, or D are determined at program initialization based on what DDs have been provided, as follows:

$SMEM - DOCUMENT - required
$DMEM - DATADICT - optional (use DOCUMENT DD if not present)

Thus, all xMEM commands can run off the single DDname of DOCUMENT, or off separate DDnames, if needed.

Using VB text format, separate DDnames are not needed to support 80 vs. 133 character record lengths. The DATADICT DD allows for a self-contained member naming scheme for dictionary elements that will not come in conflict with text section naming conventions. Example: ID is a data element but would also be Chapter 13 of the Database Structure and Content Guide (if that book had a Chapter 13).

```
DOCPICK OPTIONS  - (passed through an EXEC parm)

 - CMD

   Output $commands only  -  no text.  An unload option for
   analysis of $command structure, copy cross references,
   compare picklist title to hedr title, etc.

 - CNTL

   Unload text in DOCUPDT format (./ control records).
   Embedded copy commands are not resolved in text.

 - CTLC

   Same as CNTL, except all copy commands are resolved.
   Embedded copy commands themselves are left in the text.
   This option is for input to the text cross reference
   program.  Since this program needs to know the type of
   member (S,L,D), and the guide ID to which it belongs,
   these two additional pieces of information are output on
   the ./ control records in columns 32 and 34 respectively.
   I=id should also be specified.

 - FMT

   Format the output for the browse/print user rather than
   for guide production.  Convert $SPAG, $HEDx, $SCOL, $REST
   to carriage control.  Add spaces for browse.  Convert
   $HEDx to a title, etc.

 - I=id

   Supplies the Doc Identifier for T= processing.  Allows
   DOCPICK to determine docID length and string value.  Also
   used for CTLC guide ID output.

 - K=section key prefix

   Used mainly by Document Browse; requests that only those
   members that logically belong to the specified section key
   be picked, and that all other members be ignored.  For
   example, K=0103 would specify that only sections and
   figures belonging to 1.3 be picked.  This means that FD13
   and FD13n would be picked, but FD1 and FD2 would not.
   Note that the key of a figure is the same as the section
   number preceding it.
```

- L=level limit
  Used mainly by Document Browse, requests that only those members that logically belong to "level limit" or less be picked, and that all members that belong at a level greater than the limit be ignored. For example, L=2 would specify that only level 1 or level 2 sections and figures be picked. This means that FD1 and FD11 would be picked, but FD111 would not. Note that the level of a figure is 1 greater than the section number preceding it.

- M=member

  Member name of picklist starting point or individual text member to print. Alternative to supplying member name in the DOCSTRM JCL statement. Allows for concatenation on the DOCSTRM DD.

- N=output file number

  Character string suffix appended to the name DOCOUT to allow the program to write to the DOCOUTn DDNAME. This option is used by Document Browse to allow for different output files for each side of split screen. The value of the Dialog Manager variable ZSCREEN is used as the file name suffix. It is set on the call to DOCPICK as N=&ZSCREEN.

- S=pref

  Apply pref string as prefix selection on member names to be picked. Only members with names beginning with pref are written to output.

- SN

  Strip line numbers on input. Will blank out cols 73-80 on all lines assumed to be from SMEMs.

- T=tname

  Requests that Table of Contents ISPF Table "tname" be built from picklists. No text is picked; the program simply runs through the picklist structure.

- UCASE

  Requests that DOCPICK translate text to upper case. This option is for sites which do not have upper/lower case print capability. The option is activated by specifying "UCASE" in the PARM parameter of the EXEC card (or via the CASE parameter if MICSDOC is being used). When using the option from the PROC, the parameter is also passed to the DOCUMENT program.

- VCOPY

  Requests that Dialog Manager services variable DOCPVM should be VCOPYed for list of members to pick, rather than using a picklist. Used by the Browse (B) line command. DOCPVM contains a list of one or more type-name strings of the form tnnnnnnnn tnnnnnnnn ... etc. T is the Xmem type (S,L,D) or C for $COPY of a picklist. nnnnnnnn is the member name. This string is treated like a compact form of a picklist.

  The $WILD command is also supported through this variable in the following format: Wpppppppp*vvvvvvvv, where "p" represents a 1 to 8 character pattern name and "v" represents a 1 to 8 character wildcard replacement value.

DOCUMENT ACCESS

Document Access uses the DOCPICK program to:

- Build the ISPF tables used for table of contents browse.

- Pick line selected members chosen by the user and format them into a browse/print temporary data set.

- Pick a subset of members named in a picklist by selecting on member name substrings (leading string only).

- Unload entire guides in a form usable by the Subject Cross reference file build process.

THE DOCIN FILE

DOCIN is a SYSIN type sequential input file in DOCPICK, which can be used to pass information about component selection to the DOCPICK program executing in batch. This file is optional, and if present, reads the selection information generated by Document Access during the construction of the print job. Use of this input file removes the constraint that all information must reside in a PDS in one of the concatenations of input files used by DOCPICK.

A similar capability is provided for interactive execution of DOCPICK under ISPF. This is done by passing the component selection information through the DOCPVM dialog manager variable currently used by Document Browse for member selection.

4.6.3 Document Program Documentation

DOCUMENT is a formatting and print program for producing documents with one or two columns of text per page. It is written in SAS and is intended to read the text file produced by DOCPICK. DOCUMENT reads this file, produces tables of contents for the document, and does formatting under the control of dollar command records. Documentation for DOCPICK and for the set of allowable dollar commands is contained in Sections 4.6.1 and 4.6.2.

The files used by DOCUMENT are as follows:

TEXT

This input file contains the text file created by DOCPICK. This file is a sequential stream of text and dollar commands.

TOC

This temporary file is created by DOCUMENT to store the tables of contents. It is later read by DOCUMENT if the tables of contents are requested.

```
  REPORT

    This output file contains the final version of the
    document, complete with tables of contents if requested.


NOTE:  Since DOCUMENT is a SAS program, it also requires all
       the files normally required by SAS.
OVERVIEW OF PROGRAM OPERATION

The program is divided into three phases.  Phase one reads
the text file and builds the three types of tables of
contents (general, detailed, and figures).  They are written
to three SAS files:  GENTOC, DETTOC, and FIGTOC.

Phase two reads the three tables of contents and writes an OS
data set containing all three.  This file has the same format
as any short text member.

Phase three reads the text file again.  It processes the $
control records encountered in the file and takes the action
requested by the record (e.g., skipping lines, starting new
columns, etc.).  In addition, the third phase inserts the
tables of contents if a $$TOC card is encountered and formats
the text.  The formatting may be for one or two columns to a
page for the short text members.
JCL FOR EXECUTION

DOCUMENT is usually executed after DOCPICK.  A sample of the
JCL needed for both programs is contained in the MICSDOC
PROC.
```

4.7 Calendar Derivations

```
CA MICS calendar derivation routines reside in the
sharedprefix.MICS.SOURCE library.  The major calendar types
that can be implemented in CA MICS are the 12 month, 13
month, and 445/454/544 calendar, where the latter
describes the number of weeks per month in each calendar
quarter.
```

Note: The CA MICS Accounting and Chargeback Product includes facilities for defining a unique accounting calendar separate from the global CA MICS calendar definitions. This capability is designed to help you meet your requirements for chargeback and accounting relative to your company's fiscal calendar while continuing to use the standard 12-month calendar for the majority of your CA MICS information. See the CA MICS Accounting and Chargeback Guide for more information before altering the global CA MICS calendar.

Depending on the type of calendar being used, the source member name may be #DWMY12 ($DWMY12), #DWMY13 ($DWMY13), #DWMYWK1 ($DWMYWK1), or any other 8-character name as specified by the 13MONTHYEAR parameter in sharedprefix.MICS.PARMS(CPLXDEF) or prefix.MICS.PARMS(SITE). Determine the member currently being used for your CA MICS system before attempting modification. For information on the 13MONTHYEAR parameter, see the PIOM, Sections 2.3.1.8 and 2.3.2.4.

Regardless of the type of calendar being used, whether it is 12 or 13-month based, the purpose of the source member is to define the derivations for the CA MICS data elements DAY, WEEK, MONTH, and YEAR. The derivations are defined as SAS macros which are called at various points during CA MICS execution.

Currently, the macros are shipped in two styles: the Macro statement and the Macro language. The ones defined in Macro statement reside in the #DWMYnn members, while the corresponding Macro language definitions reside in the $DWMYnn members. It is necessary to maintain both styles of macros for compatibility. Therefore, if you have made or plan to make modifications to the #DWMYnn members, the same modifications must also be applied to the $DWMYnn members.

The operation and logic of the two members are identical. What is different is that the $DWMY12 member, written in SAS Macro language, has the macros defined with a %MACRO macroname--%MEND macroname style rather than the underscore and % style used in the older SAS Macro statements.

The macro invocation of the newer style (as in $DWMY12) SAS Macro language macros is done by %macroname rather than _macroname used in the old style (as in #DWMY12).

If you do not utilize the #DWMYnn members at all, but instead have coded your own calendar derivation member in sharedprefix.MICS.SOURCE, then that member must contain the macro definitions in both styles. For example, the member must contain the MACRO _YEAR .... % and the %MACRO YEAR; ... %MEND YEAR; definitions. Notice that the Macro language style names do not start with an underscore (_).

SPECIAL NOTE: Calendar data element derivations are very critical to the proper operation of the CA MICS database. If you intend making any modifications to standard CA MICS calendar derivations, read this entire section first. Backup your databases and libraries before implementing the change, and test each modification thoroughly.

4.7.1 Calendar Support Macro Descriptions

CA MICS database files typically include calendar related data elements as part of file summarization sequences. The "average" CA MICS file key sequence includes SYSID and other environmental descriptors (such as user ID), but end with some combination of the timestamp-oriented data elements YEAR, MONTH, WEEK, DAY, HOUR, and ZONE.

All of these data elements (YEAR, MONTH, WEEK, DAY, HOUR, and ZONE) are found by calculating some function of the data element ENDTS, End Time Stamp. ENDTS is always set by CA MICS logic to be a SAS "datetime" value, usually printed using the SAS DATETIME19.2 output format. ENDTS contents are a single numeric value that represents the date and time of a certain data source event. SAS represents any such time stamp as a floating point number of seconds after 01JAN1960. For more information about SAS representation of datetime values, see the SAS product basics documentation.

It becomes necessary in CA MICS logic to extract the several parts of ENDTS, separating ENDTS into the YEAR, MONTH, DAY, and so on, in which the data source event occurred.

HOUR is always the integer hour of the day in which ENDTS lies, and should not be modified. ZONE determination is discussed later in this section. The other elements have their own determination macros. Each element is described next. The examples shown assume a standard Gregorian calendar, as implemented in standard CA MICS 12-month year support macros (SOURCE member #DWMY12/$DWMY12).

WKSTART - From CPLXDEF (complex-level default) or SITE (unit-level override) parameter definition, the number from the following table of the day of the week on which a calendar week is said to begin. The numbers in this table are identical to the values returned by the SAS WEEKDAY function.

1 = Sunday
2 = Monday
3 = Tuesday
4 = Wednesday
5 = Thursday
6 = Friday
7 = Saturday

Example: If WKSTART is specified to be a value of 1, a calendar week begins on Sunday.

Program reference: &WKSTART

YRSTART - From ENDTS, extract the date of the first day of the year in which ENDTS occurred, and store it into the numeric data element YRSTART.

Example: ENDTS is 16AUG2001:14:00:00.00, 2pm on the 16th day of August, 2001. YRSTART is 01JAN2001.

Program reference: %YRSTART

```
YEAR    - From ENDTS, subtract 1900 from the gregorian year
          (e.g., 2001), and store it into the numeric data
          element YEAR.

          Example:  ENDTS is 16AUG2001:14:00:00.00, 2pm on
                    the 16th day of August, 2001.  YEAR is
                    101.

          Note:  Prior to year 2000, YEAR is the 2-digit
                 year of the century (e.g., 98 for 1998).

          Program reference:  %YEAR

MONTH   - From ENDTS, extract the number of the month within
          the YEAR, and store it into the numeric data
          element MONTH.

          Example:  ENDTS is 16AUG2001:14:00:00.00, 2pm on
                    the 16th day of August, 2001.  MONTH is
                    8.

          Program reference:  %MONTH
```

```
WEEK    - From ENDTS, extract the number of the week within
          the YEAR, typically 1 to 54, and store that number
          in the numeric data element WEEK.

          A typical logic flow consists of the following:

          1.  Find the date associated with the datetime
              value ENDTS.  The data element name ENDDT is
              reserved within CA MICS for storing this value.

          2.  Find the year start date of the year in which
              ENDDT occurred.  The data element name YRSTART
              is reserved within CA MICS for storing this
              data element.  This element is the output of
              invoking the %YRSTART macro.

          3.  Find the SAS WEEKDAY value of the year start
              date.

          4.  WEEK is calculated as:

              IF WEEKDAY(YRSTART)-&WKSTART >= 0 THEN
                WEEK=INT((ENDDT-YRSTART+WEEKDAY(YRSTART)
                -&WKSTART)/7+1);
              ELSE WEEK=INT((ENDDT-YRSTART+WEEKDAY(YRSTART)
                -&WKSTART+7)/7+1);

          Example:  ENDTS is 16AUG2001:14:00:00.00, 2pm on
                    the 16th day of August, 2001.  If WKSTART
                    has a value of 1, which means weeks start
                    on Sunday, then WEEK is 33.

          Program reference:  %WEEK


DAY     - From ENDTS, extract the number of day of the month,
          and store it into the numeric data element DAY.

          Example:  ENDTS is 16AUG2001:14:00:00.00, 2pm on
                    the 16th day of August, 2001.  DAY is 16.

          Program reference:  %DAY
```

Remember, these macros must be defined as both macro language and macro statements.

Macro Language	must also be defined as	Macro Statement
%YRSTART		_YRSTART
&WKSTART		_WKSTART
%YEAR		_YEAR
%MONTH		_MONTH
%WEEK		_WEEK
%DAY		_DAY

4.7.2 Non-Standard Calendar Modifications

This section explains how calendar systems implemented in CA MICS can be modified to accommodate unusual calendar requirements. Attention has been given to the decision processes required to effectively code CPLXDEF or SITE parameters, make modifications to the default date routines, and make the maintenance considerations involved in supporting a particular calendar.

Note: CA MICS Accounting and Chargeback lets you define a unique accounting calendar, separate from the global CA MICS calendar definitions. You can use it to meet your requirements for chargeback and accounting relative to your company's fiscal calendar, while continuing to use the standard 12-month calendar for the majority of your CA MICS information. See the CA MICS Accounting and Chargeback Guide for more information before altering the global CA MICS calendar.

Each section specifically addresses a particular kind of fiscal calendar strategy. The sections are:

1 - Twelve Month Calendar Modifications
2 - Thirteen Month Calendar Modifications
3 - Calendar Modifications for 445/454/544
4 - Whether to Use DWMY12 or DWMYWK1 Date Macros

Considerations For Capacity Planning

Sites commonly modify the number of years of data to keep or forecast in their CA MICS databases. When deciding how many years of data to keep or forecast for your site, consider how you plan to use the data in the database.

If you are simply running data into CA MICS and are not building any MICF inquiries that would generate dates into the future, as in forecasting, then probably 2 or 3 years is plenty. However, if you have the CA MICS Capacity Planner, you will very likely need to specify more.

How many more? If you are going to forecast data five years beyond the present date, then CA MICS will need to know about those fiscal years. If you forecast less, then CA MICS needs to know less.

The CA MICS Capacity Planner builds files of highly summarized data used by the capacity management specialist and others to perform capacity planning tasks. The default maximum number of months or weeks of data used in building and maintaining these files is 60 months or weeks. If you use the default, then CA MICS will need to know about fiscal years spanning the past five years (60 months). If you use more, once again, you will need to specify more.

The number of years that you need to specify to CA MICS depends on the timespan of data you foresee being used in CA MICS. Consider the following example:

If you wanted to take the default 60 months of data used in the CA MICS Capacity Planner for maintaining MONTHS level files and also wanted to be able to forecast up to five years into the future, you will need five years of specifications to handle the forecasts, one year for the present, and five past years to handle the data management specifications you wanted for Capacity Planner files. A total of 11 fiscal year start dates should be specified in CA MICS.

The dates you specify this year for doing 5 years of forecasting will need to be extended next year (since one year has passed). Keeping this in mind, you should review your specifications yearly to ensure that they will still do the job for you.

4.7.2.1 Twelve Month Calendar Modifications

Two varieties of Gregorian-based 12-month calendars exist:

o The fiscal year starts January 1.

o The fiscal year starts on the first of some other month.

Note: CA MICS Accounting and Chargeback lets you define a unique accounting calendar, separate from the global CA MICS calendar definitions. You can use it to meet your requirements for chargeback and accounting relative to your company's fiscal calendar, while continuing to use the standard 12-month calendar for the majority of your CA MICS information. See the Accounting and Chargeback Guide for more information before altering the global CA MICS calendar.

If your CA MICS data center has a fiscal year that begins January 1st of each year and ends 12 months later in December, you do not need to make any changes to your environment. (But you should review member MNTHFMT of the sharedprefix.MICS.SOURCE library to ensure that it has not been changed from the default that month 1 is called JANUARY.)

Furthermore, there are some considerations surrounding the calculation of the week of the year element (WEEK) when the end of a year is reached. This arises because the week boundary and the year boundary rarely coincide, leaving part of a week in one year and the rest of the week in the next.

Thus, at the beginning of a new year, when reporting on the previous week's observations, the result can appear to be either inaccurate or inconsistent depending on the run date of the report. See the discussion in Section 4.7.2.4.

On the other hand, if your data center uses the second variety, where the year begins on the first day of some month other than January, you should ensure that member MNTHFMT of the sharedprefix.MICS.SOURCE library has been modified to reflect your calendar. For example, if your fiscal year begins on July 1st of each year, the statements in MNTHFMT should be changed so that month 1 is JULY, month 2 is AUGUST, and so forth.

If you need to make modifications, these changes will take place when you next run BASPGEN or ALLPGEN. One result of the running of the BASPGEN or ALLPGEN is the creation of a format, MNTHFMT.

See the PIOM, Section 2.3.2.4, for the discussion under Thirteen Month Fiscal Year Option. While this discussion centers around 13-month calendars, the change to MNTHFMT is indeed necessary for 12-month Gregorian-based fiscal year calendars that begin on the first day of any month other than January.

In addition to changing the MNTHFMT member of the sharedprefix.MICS.SOURCE library, members #DWMY12 and $DWMY12 must be modified to reflect the start date of the fiscal year. There are two alternatives for implementing the modifications required to change the start date of your 12 month calendar.

ALTERNATIVE I : For implementing new date routines one unit at a time.

A. CODE THE COMPLEX-LEVEL CPLXDEF OR UNIT-LEVEL SITE PARAMETER MEMBER

If you want to define a global, complex-level special calendar, then code the 13MONTHYEAR parameter in sharedprefix.MICS.PARMS(CPLXDEF) as described below and run sharedprefix.MICS.CNTL(CPLXGEN). The new complex-level special calendar will take effect when you execute the unit-level BASPGEN job.

Otherwise, specify the 13MONTHYEAR parameter in prefix.MICS.PARMS(SITE) to restrict the special calendar to a specific unit.

Code the parameter as follows:

'13MONTHYEAR NO #DWMY=#memname'

where memname is your modified member. (See the PIOM, Sections 2.3.1.8 and 2.3.2.4 and Section 4.7.2.4 in this guide.) Memname must exist as a #memname version and as a $memname version if memname begins with the letters DWMY. BASPGEN is coded to include #DWMYxxx from the #BASMSTR module it generates and to include $DWMYxxx from the $BASMSTR module it generates. If memname does not begin with DWMY, BASPGEN will generate code to include #memname from the #BASMSTR module and from the $BASMSTR module.

For example, memname might be called DWMYSPC. Therefore a # version would exist called #DWMYSPC, which contains macro definitions written in SAS Macro statements. A $ version, $DWMYSPC, containing corresponding macros written in SAS Macro language would also exist. (You will see below how to code changes for both members).

B. CODE THE CHANGES FOR #memname AND $memname

1. For the #memname (which contains the date macros written with SAS Macro statements), consider the following example: suppose that your site has a fiscal year that begins on May 1 and May is considered the first month in the year. The #memname member would contain SAS code constructed along these lines:

```
%INCLUDE SOURCE(#DWMY12);

MACRO _SETYRST
  ENDDT    = DATEPART(ENDTS);
  TEMPMNTH = MONTH(ENDDT);
  IF TEMPMNTH > 4 THEN
   YRSTART = MDY(05,01,YEAR(ENDDT));
  ELSE YRSTART = MDY(05,01,(YEAR(ENDDT)-1));
%

MACRO _YEAR
  _SETYRST
  YEAR = YEAR(YRSTART) - 1899;
%

MACRO _MONTH
  TEMPMNTH = MONTH(DATEPART(ENDTS));
  IF TEMPMNTH > 4 THEN MONTH = TEMPMNTH - 4;
  ELSE MONTH = TEMPMNTH + 8;
%
```

2. For the $memname (which contains the date macros written in SAS Macro language), for the same situation as above, the code would take the form:

```
%INCLUDE SOURCE($DWMY12);

%MACRO SETYRST;
  ENDDT = DATEPART(ENDTS);
  TEMPMNTH = MONTH(ENDDT);
  IF TEMPMNTH > 4 THEN
   YRSTART = MDY(05,01,YEAR(ENDDT));
  ELSE YRSTART = MDY(05,01,(YEAR(ENDDT)-1));
%MEND SETYRST;

%MACRO YEAR;
  %SETYRST;
  YEAR=YEAR(YRSTART) - 1899;
%MEND YEAR;

%MACRO MONTH;
  TEMPMNTH = MONTH(DATEPART(ENDTS));
  IF TEMPMNTH > 4 THEN MONTH = TEMPMNTH - 4;
  ELSE MONTH = TEMPMNTH +8;
%MEND MONTH;
```

C. SAVING YOUR MODIFICATIONS

Place both the #memname and $memname members in the sharedprefix.MICS.SOURCE library.

D. MAKING THE CHANGES EFFECTIVE

Run a BASPGEN from the prefix.MICS.CNTL library. Ensure that there are no errors in MICSLOG and that the job completes with a condition code of zero.

Run the BASPGEN job for every database unit.

ALTERNATIVE II: For implementing new date routines from multiple units at the same time. This is usually used at sites that previously used the #DWMY12 member or have taken the default in the CPLXDEF and SITE parms members.

A. CODE THE CHANGES FOR #memname AND $memname

Code the #memname and $memname members as in Parts B and C of ALTERNATIVE I.

B. MODIFY THE #DWMY12 AND $DWMY12 MEMBERS

Change the #DWMY12 and $DWMY12 members in the sharedprefix.MICS.SOURCE library by adding the following statement at the end of each member:

```
For #DWMY12    %INCLUDE SOURCE(#memname)
For $DWMY12    %INCLUDE SOURCE($memname)
```

where memname is the name of your members containing the modifications to the date routines.

These modifications are effective immediately.

4.7.2.2 Thirteen Month Calendar Modifications

If your organization needs to operate CA MICS on a 13-month year, consider the following in addition to the information provided for Gregorian-based calendar installations.

Note: The CA MICS Accounting and Chargeback Product includes facilities for defining a unique accounting calendar separate from the global CA MICS calendar definitions. This capability is designed to help you meet your requirements for chargeback and accounting relative to your company's fiscal calendar while continuing to use the standard 12-month calendar for the majority of your CA MICS information. Please refer to the CA MICS Accounting and Chargeback Guide for more information before altering the global CA MICS calendar.

For CA MICS to summarize and store data in a manner consistent with your 13-month year, you must provide CA MICS with information about your fiscal calendar. While you will see later how to specify this information, in general you must determine:

- When each fiscal year begins
- Whether the fiscal year is the same year as the Gregorian calendar year
- How many years of data you need to keep
- What you will call each of the 13 months, henceforth referred to as periods

CA MICS uses this information to arrive at proper Gregorian-to-13-month date conversion routines that enable it to summarize and store data consistent with your fiscal calendar. It is your responsibility to ensure that your 13MONTHYEAR parameter statements and any related SAS code are maintained correctly into the future.

To implement a 13-month fiscal calendar in CA MICS, you must answer the following questions.

- When does each fiscal year begin?

  Ask those in your organization who established or maintain the 13-month calendar your enterprise utilizes. The dates you specify to CA MICS will be the same dates that your Accounting or Financial departments use to describe the year. This information will be used to code the complex-level shareprefix.MICS.PARMS(CPLXDEF) or unit-level prefix.MICS.PARMS(SITE) member, and also to generate a format used in CA MICS processing and by some management support applications like the CA MICS Capacity Planner and CA MICS Accounting and Chargeback.

- Is the fiscal year the same as the Gregorian year?

  The answer to this question directly affects the kinds of modifications you will need to make in order to implement your calendar system in CA MICS.

  An example illustrates how you determine the answer to this question. Suppose that your installation has a year that begins on January 1, 1996. The question to ask is "Is the fiscal year 1996 or is it 1997?"

- How many years of data need to be kept?

  Refer to Section 4.7.2 of this guide for a discussion of this topic.

- What will each period (month) be called?

  CA MICS uses a SAS format, MNTHFMT, to provide the full name of a month from a number (see PIOM, Section 2.3.2.4). Many 13-month installations call their 13 periods P01 through P13. This strategy sufficiently differentiates among the periods and eliminates confusing 13-month periods with Gregorian months. Whatever strategy you adopt to call the periods, they should be unique in the first three characters of the name.

Once the determinations above have been made, you are ready to make the necessary specifications and modifications to CA MICS. For implementing 13-month fiscal calendars in CA MICS, the process is:

1. CODE THE COMPLEX-LEVEL CPLXDEF OR UNIT-LEVEL SITE PARAMETER MEMBER

   If you want to define a global, complex-level special calendar, then code the 13MONTHYEAR parameter in sharedprefix.MICS.PARMS(CPLXDEF) as described below and run sharedprefix.MICS.CNTL(CPLXGEN). The new complex-level special calendar will take effect when you execute the unit-level BASPGEN job.

   Otherwise, specify the 13MONTHYEAR parameter in prefix.MICS.PARMS(SITE) to restrict the special calendar to a specific unit.

Code the parameter as,

```
13MONTHYEAR YES #DWMY=#memname ddmonyy ddmonyy...
```

where #memname is your modified member that begins with #DWMY or $DWMY and ddmonyy is the start date for each of the 13-month years. (See the PIOM, Sections 2.3.1.8 and 2.3.2.4).

Do not code more than 8 years on this statement. Your #memname routine may handle more than 8 years, but since this parameter handles only 8, it must be updated in the future.

BASPGEN is coded to include #DWMYxxx and $DWMYxxx from the #BASMSTR and $BASMSTR modules it generates, respectively. If '#memname' does not begin with #DWMY, BASPGEN generates code to include #memname from the #BASMSTR and $BASMSTR modules.

Proceed through the rest of this checklist and at Step 6, Generating and Loading the YRSTART Format, you will "Redefine" the years (i.e., create your own YRSTART format).

2. CODE THE CHANGES FOR #memname AND $memname

a. The modifications to the default member sharedprefix.MICS.SOURCE(#DWMY13) for #memname (which contains the date macros written with SAS MACRO statements) are shown below.

The highlighted line of code in the YEAR macro is changed according to your site's fiscal year start date. To decide what value to code, use the following technique:

- If the fiscal year begins on January 1, 1996 and your site calls this fiscal year 1997, subtract 1899 in the YEAR macro because the year of the start date (1996) minus 1899 yields 97.

```
- If the fiscal year begins on January 1, 1996
  and your site calls this fiscal year 1996,
  subtract 1900 because the start date year
  (1996) minus 1900 yields 96.

To rewrite the YEAR macro, use one of the
following techniques:

- Code SELECT statements.  For example,

  MACRO _YEAR
    _SETYRST
    SELECT;
      WHEN (YRSTART EQ '01JAN96'D) SUBYEAR=1899;
      WHEN (YRSTART EQ '01JAN97'D) SUBYEAR=1899;
      WHEN (YRSTART EQ '01JAN98'D) SUBYEAR=1899;
      WHEN (YRSTART EQ '30DEC98'D) SUBYEAR=1898;
      WHEN (YRSTART EQ '01JAN99'D) SUBYEAR=1898;
      OTHERWISE DO;
        FILE LOG;
        PUT '>ERR> ' YRSTART DATE9.
            ' NOT IN DEFINED in #DWMY13.';
        ABORT ABEND 510;
      END;
    END;
    YEAR=YEAR(YRSTART)-SUBYEAR;
  %

  Of course, you need to specify start dates for
  your fiscal years where the example above has
  coded the SAS DATE constants.
```

```
   SAS SELECT logic enables you to specify each of
   the years individually, providing a form of
   documentation that, as a System Administrator,
   you may find helpful.

 - Use "IF" or "IF-THEN-ELSE" logic.

   If, for instance, all the years that you
   defined have the same number subtracted from
   them in order to arrive at the fiscal year,
   then the only modification would be to the line
   containing 1900 as discussed above.

   If only one year required subtracting a
   different amount than all others, consider
   coding a statement that identifies that year
   and takes an action; it does something
   different for all other years.


      DEFAULT #DWMY13 MEMBER SHIPPED WITH CA MICS


Note:  Annotations to logic are shown by <== indicating the
       area for comment, and end in !! for separation.  These
       annotations are not SAS code.
```

```
/* ************************************************** */


MACRO _MONTHDF 13%

MACRO _DAY
  _SETYRST
  DAY=MOD(ENDDT-YRSTART+1,28);
  IF DAY=0 THEN DAY=28;
  IF ENDDT-YRSTART>363 THEN DAY=ENDDT-YRSTART-335;
%

MACRO _WEEK
  _SETYRST
  IF WEEKDAY(YRSTART)-_WKSTART >= 0 THEN
   WEEK=INT((ENDDT-YRSTART+WEEKDAY(YRSTART)
   -_WKSTART)/7+1);
  ELSE WEEK=INT((ENDDT-YRSTART+WEEKDAY(YRSTART)
   -_WKSTART+7)/7+1);
%

MACRO _MONTH
  _SETYRST
  MONTH=INT((ENDDT-YRSTART+1)/28);
  IF MOD((ENDDT-YRSTART+1),28) NE 0 THEN MONTH=MONTH+1;
  IF MONTH>13 THEN MONTH=13;
%

MACRO _YEAR
  _SETYRST
  YEAR=YEAR(YRSTART)-1900;             <== changes here!!
%

MACRO _SETYRST
  ENDDT=DATEPART(ENDTS);
  YRSTART=INPUT(PUT(ENDDT,YRSTART.),5.);

  IF YRSTART=. THEN DO;
    FILE LOG;
    PUT '>ERR> ' ENDDT DATE9. ' NOT IN DEFINED YEARS.'/
      '>>>>> START OF YEAR LIST MUST BE UPDATED IN '/
      'MICS.PARMS(SITE) PARAMETER "13MONTHYEAR" AND '/
      'BASPGEN JOB RUN.';
    ABORT ABEND 510;
  END;
%
```

```
MACRO _PREVWK
  /* SELECT ONLY DATA FROM THE PREVIOUS WEEK.       */
  /* CALCULATE PREVIOUS WEEK BY SUBTRACTING 1       */
  /* FROM THE WEEK VALUE OF TODAY.                  */
  /* (USING THE _WEEK MACRO). IF AFTER SUBTRACTION  */
  /* THE WEEK IS FOUND TO BE ZERO,                  */
  /* (THIS WILL HAPPEN THE FIRST WEEK OF THE YEAR), */
  /* THE WEEK OF                                    */
  /* THE LAST DAY OF THE PREVIOUS YEAR IS USED.     */

  RETAIN PREVWK; DROP PREVWK;
  IF PREVWK=. THEN DO;
  /* H_WEEK IS USED TO HOLD THE WEEK VALUE OF       */
  /* THE CURRENT OBSERVATION                        */
  /* WHILE THE PREVIOUS WEEK IS BEING CALCULATED    */
  H_TS=ENDTS;
  H_WEEK=WEEK;
  ENDTS=DHMS(TODAY(),0,0,0);
  _WEEK
  PREVWK=WEEK-1;
  IF PREVWK=0 THEN DO;
    ENDTS=DHMS(YRSTART-1,0,0,0);
    _WEEK;
    PREVWK=WEEK;
  END;
  ENDTS=H_TS;
  WEEK=H_WEEK;
  END;
  IF WEEK = PREVWK;
  %
```

b. For $memname (which contains the date macros written in SAS Macro language), the code illustrated below illustrates the situation described above for the #memname case.

```
%MACRO YEAR;
  %SETYRST;
  SELECT;
  WHEN (YRSTART EQ '01JAN96'D) SUBYEAR=1899;
  WHEN (YRSTART EQ '01JAN97'D) SUBYEAR=1899;
  WHEN (YRSTART EQ '01JAN98'D) SUBYEAR=1899;
  WHEN (YRSTART EQ '30DEC98'D) SUBYEAR=1898;
  WHEN (YRSTART EQ '01JAN99'D) SUBYEAR=1898;
  OTHERWISE DO;
     FILE LOG;
     PUT '>ERR> ' YRSTART  DATE9.
         ' NOT IN DEFINED in $DWMY13.';
     ABORT ABEND 510;
  END;
  END;
  YEAR=YEAR(YRSTART)-SUBYEAR;
%MEND YEAR;
```

Note: The processes for arriving at the dates and values to be subtracted are the same as discussed in the previous section.

You need to specify start dates for your fiscal years where the example above has coded the SAS DATE constants.

You can determine the necessary dates and values by using "IF" or "IF-THEN-ELSE" logic. If, for instance, all the years that you defined have the same number subtracted from them in order to arrive at the fiscal year, then the only modification would be to the line containing 1900 as discussed in the #DWMY13 section.

SAS SELECT logic enables you to specify each of the years individually, providing a form of documentation that, as System Administrator, you may find helpful.

```
          If only one year required subtracting a different
          amount than all others, you might find it simpler
          to code a statement that identifies that year and
          takes an action; it does something different for
          all other years.

        DEFAULT $DWMY13 MEMBER SHIPPED WITH CA MICS

%LET MONTHDF=13;

%MACRO DAY;
  %SETYRST;
  IF ENDDT-YRSTART GT 363 THEN DAY=ENDDT-YRSTART-335;
  ELSE DO;
    DAY=MOD(ENDDT-YRSTART+1,28);
    IF DAY EQ 0 THEN DAY=28;
  END;
%MEND DAY;

%MACRO WEEK;
  %SETYRST;
  IF WEEKDAY(YRSTART)-&WKSTART GE 0 THEN
   WEEK=INT((ENDDT-YRSTART+WEEKDAY(YRSTART)
        -&WKSTART)/7+1);
  ELSE WEEK=INT((ENDDT-YRSTART+WEEKDAY(YRSTART)
            -&WKSTART+7)/7+1);
%MEND WEEK;
%MACRO MONTH;
  %SETYRST;
  MONTH=INT((ENDDT-YRSTART+1)/28);
  IF MOD((ENDDT-YRSTART+1),28) NE 0 THEN MONTH=MONTH+1;
  IF MONTH GT 13 THEN MONTH=13;
%MEND MONTH;

%MACRO YEAR;
   %SETYRST;
   YEAR=YEAR(YRSTART)-1900;            <== changes here!!
%MEND YEAR;

%MACRO SETYRST;
 ENDDT=DATEPART(ENDTS);
 YRSTART=INPUT(PUT(ENDDT,YRSTART.),5.); <== 5. to Date7.!!
 IF YRSTART EQ . THEN DO;
   VARX=PUT(ENDDT,DATE9.);
   CALL MICSLOG('BAS00473');
   ABORT ABEND 510;
 END;
%MEND SETYRST;
```

3. MODIFY THE MNTHFMT FORMAT

The sharedprefix.MICS.SOURCE member MNTHFMT provides the full name for periods or months based on the period or month number. Modify MNTHFMT to use the names you decided upon earlier in this section. The MNTHFMT illustrated below shows periods P01 through P13.

```
PROC FORMAT LIBRARY=MCOLIB.USERFMT2;
VALUE MNTHFMT
   1  = 'P01'
   2  = 'P02'
   3  = 'P03'
   4  = 'P04'
   5  = 'P05'
   6  = 'P06'
   7  = 'P07'
   8  = 'P08'
   9  = 'P09'
   10 = 'P10'
   11 = 'P11'
   12 = 'P12'
   13 = 'P13';
```

Note: These changes become effective when you run BASPGEN.

4. SAVE YOUR MODIFICATIONS

Ensure that you have backed up your databases and libraries before saving both the #memname and $memname members and the MNTHFMT member in the sharedprefix.MICS.SOURCE library.

5. MAKE THESE CHANGES EFFECTIVE

For each unit, submit the job in prefix.MICS.CNTL(BASPGEN). Ensure that there are no error messages in MICSLOG and that the job completes with a condition code of zero.

6. (Conditional) GENERATE AND LOAD THE YRSTART FORMAT

If your site needs more than 8 years of data, or your 'year start' value requires 2 dates that start in the same calendar year (i.e., 01JAN98 and 28DEC98), you can code a PROC FORMAT table to associate the year start date with a date range.

Modify the YRSTART macro to accept a DATE7. Code the format of the year start date for convenience in the format statements.

```
%MACRO SETYRST;
 ENDDT=DATEPART(ENDTS);
 YRSTART=INPUT(PUT(ENDDT,YRSTART.),5.); <== 5. to Date7.!!
 IF YRSTART EQ .  THEN DO;
   VARX=PUT(ENDDT,DATE9.);
   CALL MICSLOG('BAS00473');
   ABORT ABEND 510;
 END;
%MEND SETYRST;
```

Finally, generate a YRSTART format to use in the CA MICS date processing routines that you have modified. The format must take the following form:

```
 PROC FORMAT LIBRARY=MCOLIB.USERFMT1 PRINT;
   VALUE YRSTART
    OTHER='.'
    '01JAN96'D - '31DEC96'D = '01JAN96'

         ...

    '01JAN98'D - '31DEC98'D = '01JAN98';
```

For each year that you wish to specify to CA MICS, you must code a SAS statement like the one above, substituting your first and last dates where the date constants are in the example. The form is:

```
 'first date'D - 'last date'D = 'year start date'
```

Do NOT specify the Date (D) constant modifier for the year start date!

Note: The YRSTART format is recreated and written to the prefix.MICS.MUOLIB MICSFMTS format catalog every time BASPGEN is executed. In order to use your calendar definitions, we recommend you write your YRSTART format to the MCOLIB.USERFMT1 library, as shown above. This way, because of the catalog search order CA MICS specifies at initialization, your format will always be found first, and consequently, used.

The YRSTART format must be a permanent SAS format. Section 4.5 of this guide provides instructions for creating permanent user-defined formats.

Successfully generating and saving this format concludes the implementing process for 13-month calendars.

4.7.2.3 Calendar Modifications for 445/454/544 Calendars

If your organization needs to operate CA MICS on a calendar system with each calendar quarter composed of a 4 week month, 4 week month, and 5 week month (or 4-5-4 or 5-4-4), consider the following addition to the information provided for Gregorian-based calendar installation.

Note: The CA MICS Accounting and Chargeback Product includes facilities for defining a unique accounting calendar separate from the global CA MICS calendar definitions. This capability is designed to help you meet your requirements for chargeback and accounting relative to your company's fiscal calendar while continuing to use the standard 12-month calendar for the majority of your CA MICS information. Please refer to the CA MICS Accounting and Chargeback Guide for more information before altering the global CA MICS calendar.

The topics that are relevant when dealing with this type of special accounting calendar are:

1. When does the first day of the accounting week begin? (e.g., Sunday, Monday etc.)

2. When does each accounting year begin and end? It must begin at the start of a week and the start of accounting month 1.

3. Which accounting year has 53 weeks in it? Generally, there are 52 weeks in a year and then every 6th year there is a 53 week year to adjust for the extra days.

4. Which month within the 53 week year will have the 5th week? The normal pattern of either 4-4-5, 4-5-4, or 5-4-4 weeks in a quarter of a year must have an extra week in one designated month.

5. Which years have starting dates in the same year as the accounting year (e.g., starting date of 02JAN01 is in accounting year 2001).

6. Which pattern fits your accounting calendar? The patterns are either 4-4-5, 4-5-4, or 5-4-4.

7. How many years of data do you need to keep for CA MICS?

After you have resolved the previous questions, the implementation strategy within CA MICS can be addressed.

A. CODE THE COMPLEX-LEVEL CPLXDEF OR UNIT-LEVEL SITE PARAMETER MEMBER

If you want to define a global, complex-level special calendar, then code the 13MONTHYEAR parameter in sharedprefix.MICS.PARMS(CPLXDEF) as described below and run sharedprefix.MICS.CNTL(CPLXGEN). The new complex-level special calendar will take effect when you execute the unit-level BASPGEN job.

Otherwise, specify the 13MONTHYEAR parameter in prefix.MICS.PARMS(SITE) to restrict the special calendar to a specific unit.

Code the parameter as,

'13MONTHYEAR NO #DWMY=#memname'

where memname is your modified member. See PIOM, Sections 2.3.1.8 and 2.3.2.4. 'memname' must exist as a #memname version and as a $memname version if memname begins with the letters DWMY. BASPGEN is coded to include #DWMYxxx from the #BASMSTR module it generates and include $DWMYxxx from the $BASMSTR module it generates. If memname does not begin with DWMY, BASPGEN will generate code to include #memname from the #BASMSTR module and from the $BASMSTR module.

For example, memname might be called DWMY445; therefore a # version would exist called #DWMY445 which contains macro definitions written in SAS Macro statements and a $ version, $DWMY445, containing corresponding macros written in SAS Macro language would also exist. (You will see later how to code changes for both members).

B. CODE THE CHANGES FOR #memname AND $memname

For $memname, modify sharedprefix.MICS.SOURCE($DWMY445). '$memname' contains the date macros written with SAS Macro language.

For each year that you need to specify to CA MICS, code a WHEN statement like the ones shown below in the SETYRST macro. Provide the start date, the ending date, and the value to set the variable YRSTART to, which is the start date again. Be careful that each start date begins on the first of your defined week and the start of accounting month 1.

```
%MACRO SETYRST; ENDDT = DATEPART(ENDTS);

  %* MODIFY THE FOLLOWING CODE AS NECESSARY.  ;
  %* INDICATE THE START      ;
  %* OF YEAR AND THE END OF THE SAME YEAR IN ;
  %* THE FOLLOWING SET OF     ;
  %* STATEMENTS. ;

  SELECT;
    WHEN ('02JAN84'D LE ENDDT LE '30DEC84'D )
         YRSTART='02JAN84'D;
    WHEN ('31DEC84'D LE ENDDT LE '29DEC85'D )
         YRSTART='31DEC84'D;
    WHEN ('30DEC85'D LE ENDDT LE '28DEC86'D )
         YRSTART='30DEC85'D;
    WHEN ('29DEC86'D LE ENDDT LE '27DEC97'D )
         YRSTART='29DEC86'D;
    WHEN ('28DEC87'D LE ENDDT LE '25DEC88'D )
         YRSTART='28DEC87'D;
    WHEN ('26DEC88'D LE ENDDT LE '31DEC89'D )
         YRSTART='26DEC88'D;
    WHEN ('01JAN90'D LE ENDDT LE '30DEC90'D )
         YRSTART='01JAN90'D;
    WHEN ('31DEC90'D LE ENDDT LE '29DEC91'D )
         YRSTART='31DEC90'D;
    OTHERWISE DO;
      CALL MICSLOG('BAS00473');
      ABORT ABEND 510;
    END;
  END;
%MEND SETYRST;
```

The Macro language statements are:

```
%MACRO YEAR;
  %SETYRST;

  SELECT (YRSTART);
    WHEN ('02JAN84'D) YEAR = YEAR(YRSTART) - 1900;
    WHEN ('01JAN90'D) YEAR = YEAR(YRSTART) - 1900;
    OTHERWISE YEAR = YEAR(YRSTART) - 1899;
  END;
%MEND YEAR;
```

As shown above, the year is identified by its start date. The amount of 1900 is subtracted from years coded here. This results in the accounting year being correctly identified for those years in which the start date is in the same year as the accounting year.

Using the example of a fiscal year beginning on January 2, 1984 as coded above, let's see how this works. If we call this fiscal year 1984, we would want to subtract 1900 because the year of our start date--1984--minus 1900 yields 84. If we decide to call the fiscal year 1985, we would subtract 1899 because the start date year of 1984 minus 1899 yields 85.

Since our logic is designed to do this as the normal course of action, we would have omitted 02JAN84 from the SELECT statement. The real intent here is to arrive at a value which, when subtracted from the year of the start date of each fiscal year that we have specified to CA MICS, will result in the year that we intend to have the fiscal year called.

The code shown below pertains to defining which year will have 53 weeks.

```
%MACRO MONTH;
  %YEAR;
  TEMPDAY = ENDDT + 1 - YRSTART;

  %* MODIFY THE FOLLOWING CODE AS  NECESSARY.  EACH  53;
  %* WEEK  YEAR  MUST  BE  CODED  IN  THIS  CHECK.  THE;
  %* SPECIAL FORMAT THAT USES 371 DAYS WILL BE USED FOR;
  %* A 53 WEEK YEAR.  THE EXAMPLE SHOWS 1989 AS THE  53;
  %* WEEK YEAR ;

  SELECT (YEAR);
    WHEN (89) MTHDAYS=PUT(TEMPDAY,YR455WK.);
              %* 1989 IS 53-WEEK YEAR  ;
    OTHERWISE MTHDAYS=PUT(TEMPDAY,YR445WK.);
  END;
  MONTH = INPUT(SUBSTR(MTHDAYS,1,2),2.);
%MEND MONTH;
```

Generally, there are 52 weeks in a year and then every 6th year there is a 53 week year to adjust for the extra days. This implementation in CA MICS consists of creating two formats in the sharedprefix.MICS.LOAD library to be used by the exit coding. The first format, labeled YR445WK, is to be used with the 52 week year. The second format, labeled YR455WK, is to be used with the 53 week year. In our example code above, 1989 is a 53 week year and will be formatted using the 53 week format YR455WK. The YR445WK and YR455WK formats are defined in Step C, below.

The code shown below uses the YR445WK and YR455WK formats in a way similar to the MONTH macro logic. The ordinal number of a day within a year defines the start day of the month in which the day lies. The difference between ordinal year day and month start day is DAY (of month).

```
%MACRO DAY;
  %YEAR;
  TEMPDAY = ENDDT + 1 - YRSTART;

  %* MODIFY THE FOLLOWING CODE AS NECESSARY.  EACH 53;
  %* WEEK  YEAR  MUST  BE  CODED  IN THIS CHECK.  THE;
  %* SPECIAL FORMAT THAT USES 371 DAYS WILL  BE  USED;
  %* FOR  A  53 WEEK YEAR.  THE EXAMPLE SHOWS 1989 AS;
  %* THE 53 WEEK YEAR;

  SELECT (YEAR);
    WHEN (89) MTHDAYS=PUT(TEMPDAY,YR455WK.);
             %* 1989 IS 53-WEEK YEAR  ;
    OTHERWISE MTHDAYS=PUT(TEMPDAY,YR445WK.);
  END;
  DAY = TEMPDAY - INPUT(SUBSTR(MTHDAYS,3,3),3.);
%MEND DAY;
```

EXAMPLE

Below is an example of the coded $DWMY445 that uses a 4-4-5 accounting calendar. The first year starts on 02JAN84 and ends on 30DEC84. Each week begins on Monday. The accounting year that has 53 weeks is 1989. The 11th month has the extra week in the 53 week year. The two years in the sample that begin in the same year as the accounting year have the starting dates of 02JAN84 and 01JAN90.

```
%LET WKSTART=2;

%LET MONTHDF=12;

%MACRO SETYRST;
  ENDDT = DATEPART(ENDTS);

  SELECT;
    WHEN ('02JAN84'D LE ENDDT LE '30DEC84'D )
         YRSTART='02JAN84'D;
    WHEN ('31DEC84'D LE ENDDT LE '29DEC85'D )
         YRSTART='31DEC84'D;
    WHEN ('30DEC85'D LE ENDDT LE '28DEC86'D )
         YRSTART='30DEC85'D;
    WHEN ('29DEC86'D LE ENDDT LE '27DEC97'D )
         YRSTART='29DEC86'D;
    WHEN ('28DEC87'D LE ENDDT LE '25DEC88'D )
         YRSTART='28DEC87'D;
    WHEN ('26DEC88'D LE ENDDT LE '31DEC89'D )
         YRSTART='26DEC88'D;
    WHEN ('01JAN90'D LE ENDDT LE '30DEC90'D )
         YRSTART='01JAN90'D;
    WHEN ('31DEC90'D LE ENDDT LE '29DEC91'D )
         YRSTART='31DEC90'D;
    OTHERWISE DO;
      CALL MICSLOG('BAS00473');
      ABORT ABEND 510;
    END;
  END;
%MEND SETYRST;
```

```
%MACRO YEAR;
  %SETYRST;
  SELECT (YRSTART);
    WHEN ('02JAN84'D) YEAR = YEAR(YRSTART) - 1900;
    WHEN ('01JAN90'D) YEAR = YEAR(YRSTART) - 1900;
    OTHERWISE YEAR = YEAR(YRSTART) - 1899;
  END;
%MEND YEAR;

%MACRO MONTH;
  %YEAR;
  TEMPDAY = ENDDT + 1 - YRSTART;
  SELECT (YEAR);
    WHEN (89) MTHDAYS=PUT(TEMPDAY,YR455WK.);
              %* 1989 IS 53-WEEK YEAR  ;
    OTHERWISE MTHDAYS=PUT(TEMPDAY,YR445WK.);
  END;
  MONTH = INPUT(SUBSTR(MTHDAYS,1,2),2.);
%MEND MONTH;

%MACRO DAY;
  %YEAR;
  TEMPDAY = ENDDT + 1 - YRSTART;
  SELECT (YEAR);
    WHEN (89) MTHDAYS=PUT(TEMPDAY,YR455WK.);
              %* 1989 IS 53-WEEK YEAR  ;
    OTHERWISE MTHDAYS=PUT(TEMPDAY,YR445WK.);
  END;
  DAY = TEMPDAY - INPUT(SUBSTR(MTHDAYS,3,3),3.);
%MEND DAY;
```

For #memname (which contains the date macros written with SAS Macro statements), the modifications to the default member sharedprefix.MICS.SOURCE($DWMY445) you have just completed would need to be copied to sharedprefix.MICS.SOURCE(#DWMY445) and modified to the SAS Macro statement style.

Below is an example of the coded #DWMY445 that is the SAS Macro statement version of the $DWMY445 member we coded in the earlier example.

```
MACRO _WKSTART 2 %

MACRO _MONTHDF 12 %

MACRO _SETYRST;
  ENDDT = DATEPART(ENDTS);
  SELECT;
    WHEN ('02JAN84'D LE ENDDT LE '30DEC84'D )
         YRSTART='02JAN84'D;
    WHEN ('31DEC84'D LE ENDDT LE '29DEC85'D )
         YRSTART='31DEC84'D;
    WHEN ('30DEC85'D LE ENDDT LE '28DEC86'D )
         YRSTART='30DEC85'D;
    WHEN ('29DEC86'D LE ENDDT LE '27DEC97'D )
         YRSTART='29DEC86'D;
    WHEN ('28DEC87'D LE ENDDT LE '25DEC88'D )
         YRSTART='28DEC87'D;
    WHEN ('26DEC88'D LE ENDDT LE '31DEC89'D )
         YRSTART='26DEC88'D;
    WHEN ('01JAN90'D LE ENDDT LE '30DEC90'D )
         YRSTART='01JAN90'D;
    WHEN ('31DEC90'D LE ENDDT LE '29DEC91'D )
         YRSTART='31DEC90'D;
    OTHERWISE DO;
      CALL MICSLOG('BAS00473');
      ABORT ABEND 510;
    END;
  END;

MACRO _YEAR
  _SETYRST
  SELECT (YRSTART);
    WHEN ('02JAN84'D) YEAR = YEAR(YRSTART) - 1900;
    WHEN ('01JAN90'D) YEAR = YEAR(YRSTART) - 1900;
    OTHERWISE YEAR = YEAR(YRSTART) - 1899;
  END;

MACRO _MONTH
  _YEAR
  TEMPDAY = ENDDT + 1 - YRSTART;
  SELECT (YEAR);
    WHEN (89) MTHDAYS=PUT(TEMPDAY,YR455WK.);
              %* 1989 IS 53-WEEK YEAR  ;
    OTHERWISE MTHDAYS=PUT(TEMPDAY,YR445WK.);
  END;
  MONTH = INPUT(SUBSTR(MTHDAYS,1,2),2.);
```

```
MACRO _DAY
  _YEAR
  TEMPDAY = ENDDT + 1 - YRSTART;
  SELECT (YEAR);
    WHEN (89) MTHDAYS=PUT(TEMPDAY,YR455WK.);
              %* 1989 IS 53-WEEK YEAR  ;
    OTHERWISE MTHDAYS=PUT(TEMPDAY,YR445WK.);
  END;
  DAY = TEMPDAY - INPUT(SUBSTR(MTHDAYS,3,3),3.);
%
```

C. GENERATING AND LOADING THE YRWK445 AND YR455WK FORMATS

The implementation in CA MICS consists of creating two PROC FORMATs into the sharedprefix.MICS.MCOLIB library to be used by the exit coding. The two PROC FORMATs that are provided use a 4-4-5 accounting calendar. The first format, labeled YR445WK, is to be used with the 52 week year. The second format, labeled YR455WK, is to be used with the 53 week year.

If you need a pattern other than a 4-4-5 week pattern, then the day ranges must be adjusted to fit the other patterns of 4-5-4 or 5-4-4 weeks in an accounting calendar. The YR445WK and YR455WK PROC FORMATs in the sample coding show a range of values for the number of days in a month (e.g., 1-28). The label value in the PROC FORMATs show the month followed by the number of days from the beginning of the year (e.g., '01000'). The first line of the YR445WK PROC FORMAT sample is:

```
1-28 = '01000'
```

It shows that days 1-28 are in month 01 and there are 000 days from the start of the year to the first day of month 01. Both the YR445WK and YR455WK PROC FORMATs must be adjusted if you are going to use a 4-5-4 or 5-4-4 accounting calendar. The YR455WK PROC FORMAT must be adjusted if the 11th month is not used for the extra week in the 53 week year. The formats can be built with a SAS batch job. To build the format in batch:

```
//... JOB .......
/*PROCLIB DSN=sharedprefix.MICS.PROCLIB
//    EXEC MICSNDBx   /* Where x is the db identifier */
//SYSIN DD *
  PROC FORMAT PRINT DDNAME=MCOLIB.USERFMT1;
   VALUE YR445WK
   1-28    = '01000'
   29-56   = '02028'
   57-91   = '03056'
   92-119  = '04091'
   120-147 = '05119'
   148-182 = '06147'
   183-210 = '07182'
   211-238 = '08210'
   239-273 = '09238'
   274-301 = '10273'
   302-329 = '11301'
   330-364 = '12329'
   OTHER   = 'UNKWN';
  PROC FORMAT PRINT DDNAME=MCOLIB.USERFMT1;
   VALUE YR455WK
   1-28    = '01000'
   29-56   = '02028'
   57-91   = '03056'
   92-119  = '04091'
   120-147 = '05119'
   148-182 = '06147'
   183-210 = '07182'
   211-238 = '08210'
   239-273 = '09238'
   274-301 = '10273'
   302-336 = '11301'
   337-371 = '12336'
   OTHER   = 'UNKWN';
 RUN;
```

D. IMPLEMENTING THE CHANGES

There are two alternatives that can be used to make the exit coding available to the CA MICS daily processing. Choose one of the alternatives based on your requirements and situation. In either case, add the indicated code with any revisions that are necessary to the sharedprefix.MICS.SOURCE library as a new member. As the implementation date of the new date routines into an existing system has multiple considerations, please consult the CA MICS Product Support Group. Use either Alternative I or Alternative II to implement the DWMY445 routines.

```
 ALTERNATIVE I:  TO ADD THE NEW DATE ROUTINE TO ONE UNIT
                 AT A TIME


1.  After you have completed parts A - D above, add the
    following SAS statement before the MACRO _WKSTART
    statement in #DWMY445 and just before the %MACRO
    WKSTART statement in $DWMY445:

      %INCLUDE SOURCE(#DWMY12);  in the #DWMY445 member
      %INCLUDE SOURCE($DWMY12);  in the $DWMY445 member

2.  The #DWMY445 and $DWMY445 members must be added to
    the sharedprefix.MICS.SOURCE library with the
    modifications that you made.

3.  Run BASPGEN from the prefix.MICS.CNTL library.
    Ensure that there are no errors in MICSLOG and that
    the job completes with a condition code of zero.

    Run BASPGEN for every database unit.

After you have added the DWMY445 modules to the source
library and run a BASPGEN, you can run a CA MICS DAILY.
The YEAR, MONTH, WEEK, and DAY fields should be reviewed
for validity after the Daily processing.
ALTERNATIVE II:  ADD TO ALL UNITS SIMULTANEOUSLY

This implementation adds the new date routine to all the
units at the same time, and is usually used by sites that
have previously used the #DWMY12 member or taken the
default sharedprefix.MICS.PARMS(CPLXDEF) member and the
default prefix.MICS.PARMS(SITE) member.

Add the DWMY445 members into sharedprefix.MICS.SOURCE
with the modifications that you have made.

1.  After you have completed parts B -  D above, add the
    following SAS statement to #DWMY12 and $DWMY12 in the
    sharedprefix.MICS.SOURCE library:

     %INCLUDE SOURCE(#DWMY445); for the #DWMY12 member
     %INCLUDE SOURCE($DWMY445); for the $DWMY12 member

2.  The #DWMY445 and $DWMY445 members must be added to
    the sharedprefix.MICS.SOURCE library with the
    modifications that you made.
```

3. Run BASPGEN from the prefix.MICS.CNTL library.
   Ensure that there are no errors in MICSLOG and that the job completes with a condition code of zero.

   Run BASPGEN for every database unit.

Your next Daily will reflect the new date routines and the YEAR, MONTH, WEEK, and DAY fields should be reviewed for validity.

4.7.2.4 Whether to Use DWMY12 or DWMYWK1 Date Macros

The effect of choosing either the #DWMY12/$DWMY12 set of macros or the #DWMYWK1/$DWMYWK1 set of macros is explained below. This discussion may be of particular interest to non-North American CA MICS installations.

Note: When we talk about the "second week of a year," the meaning is different from the parameter WEEK=01, as designated in CA MICS.

Note that the WEEKSTART parameter in MICS.PARMS(SITE) is assumed to be equal to one. (For more information on the WEEKSTART option, see Section 2.3.1.8 in the PIOM.)

When DWMY12 Is Used

This set of date macros is the default. It provides specifications that correspond most closely to the definition of days/weeks/months normally used and understood in North America.

The determination of the calendar week of the year element (WEEK) is an integer from 1 to 54:

o WEEK=01 begins on the first day of the calendar year and extends up to the end of the following Saturday.

o WEEK=53 begins on the last Sunday of the calendar year and extends up to the end of that year.

See sharedprefix.MICS.DIC.TEXT(WEEK) for more details.

When a MICF query requests observations from the previous week, the date macros determine the WEEK value that corresponds to the seventh day prior to the query run date. Then the query selects all observations with matching WEEK values. Consequently, when the query is run during the second week of a year (WEEK=02), query results will appear to be either inaccurate or inconsistent from day to day.

For example, consider a query run in the second week of 2004. If the query is run on January 4th, 5th, 6th, or 7th, observations from December 28th through December 31st are selected. If the query is run on January 8th, 9th, or 10th, observations from January 1st through January 3rd are selected. So depending on the run date, query results will differ.

When DWMYWK1 Is Used

This set of date macros corresponds most closely to the definition of days/weeks/months that supports various international preferences.

If you choose this set of date macros, WEEK=01 begins on the Sunday of the second week of the year, if the first week has three days or less (for example, the first week in 2004). Furthermore, there are no partial weeks at year-end or year-beginning. For example, WEEK=53 can contain observations from December 28 right through to and including January 3.

When a query requests observations from the previous week, the date macros determine the WEEK value that actually corresponds to the week prior to the query run date. However, the current behavior of MICF is such that the query selects only observations with matching WEEK values plus a YEAR value corresponding to the previous year.

Consequently, when the query is run during the second week of a year (WEEK=01 or WEEK=02), observations from the first few days in January are not selected. This occurs because the YEAR value used by the MICF TMFRSEL macro to select observations corresponds to the previous year. This causes the query results to appear to be either inaccurate or insufficient.

4.8 Account Code Determination Routine Coding

Account code determination routines identify cost centers. Typical routines contain table lookup operations, such as the following:

```
  If the jobname is      then the department is
  -----------------      ----------------------
         ACPxxxxx          Accounts Payable
         ACRxxxxx          Accounts Receivable
         ACTxxxxx          Actuarial
         PAYxxxxx          Payroll
         SYSxxxxx          Systems Support
```

A sample implementation might be:

```
   IF JOB = :'ACP' THEN ACCTNO1 = 'ACCOUNTS PAYABLE';
   IF JOB = :'ACR' THEN ACCTNO1 = 'ACCOUNTS RCVBL';
   IF JOB = :'ACT' THEN ACCTNO1 = 'ACTUARIAL';
   IF JOB = :'PAY' THEN ACCTNO1 = 'PAYROLL';
   IF JOB = :'SYS' THEN ACCTNO1 = 'SYSTEMS SUPPORT';
```

This code structure may execute faster if ELSE statements are used, as in the following example:

```
    IF JOB = :'ACP' THEN ACCTNO1 = 'ACCOUNTS PAYABLE';
--> ELSE IF JOB = :'ACR' THEN ACCTNO1 = 'ACCOUNTS RCVBL';
--> ELSE IF JOB = :'ACT' THEN ACCTNO1 = 'ACTUARIAL';
--> ELSE IF JOB = :'PAY' THEN ACCTNO1 = 'PAYROLL';
--> ELSE IF JOB = :'SYS' THEN ACCTNO1 = 'SYSTEMS SUPPORT';
```

IF statements after the statement that satisfies the condition are not executed, which results in faster execution. That is, if the job name is ACRCFM01, the first and second statements would be executed, but the third through the fifth would not. Such a structure of IF ... ELSE IF ... is commonly called an IF cascade.

An alternative construction uses a SELECT statement instead of an IF cascade. The normal use of SELECT is

```
SELECT(select-expression);
 WHEN (when-expression) ... ;
 .
 .
 .
 OTHERWISE  ... ;
END;
```

The advantage of the SELECT statement is that the select-expression is evaluated once and is compared sequentially against each when-expression until a match occurs. If no match occurs, the OTHERWISE statement is executed.

In the first example, the IF cascade is comparing against the first three characters of the job name using the special colon modifier (=:) in the IF statement. Since the special colon modifier is not applicable to the SELECT statement, you must code a SUBSTR function as illustrated in the following example:

```
SELECT(SUBSTR(JOB,1,3));
 WHEN('ACP') ACCTNO1 = 'ACCOUNTS PAYABLE';
 WHEN('ACR') ACCTNO1 = 'ACCOUNTS RCVBL ';
 WHEN('ACT') ACCTNO1 = 'ACTUARIAL    ';
 WHEN('PAY') ACCTNO1 = 'PAYROLL     ';
 WHEN('SYS') ACCTNO1 = 'SYSTEMS SUPPORT ';
 OTHERWISE ... ;
END;
```

Also consider the volume of input data when designing your code. Assume that the following relationship exists between jobnames and volume:

If the jobname is	then the number of jobs per day is
ACPxxxxx	1000
ACRxxxxx	4500
ACTxxxxx	100
PAYxxxxx	400
SYSxxxxx	2000

This volume distribution would mean the IF cascade would operate even more efficiently if the conditional with the largest number of hits were highest in the cascade.

Given our example, the IF cascade should be reordered as follows:

```
IF JOB = :'ACR' THEN ACCTNO1 = 'ACCOUNTS RCVBL';
ELSE IF JOB = :'SYS' THEN ACCTNO1 = 'SYSTEMS SUPPORT';
ELSE IF JOB = :'ACP' THEN ACCTNO1 = 'ACCOUNTS PAYABLE';
ELSE IF JOB = :'PAY' THEN ACCTNO1 = 'PAYROLL';
ELSE IF JOB = :'ACT' THEN ACCTNO1 = 'ACTUARIAL';
```

and the SELECT statements should be reordered as follows:

```
SELECT(SUBSTR(JOB,1,3));
  WHEN('ACR') ACCTNO1 = 'ACCOUNTS RCVBL ';
  WHEN('SYS') ACCTNO1 = 'SYSTEMS SUPPORT ';
  WHEN('ACP') ACCTNO1 = 'ACCOUNTS PAYABLE';
  WHEN('PAY') ACCTNO1 = 'PAYROLL     ';
  WHEN('ACT') ACCTNO1 = 'ACTUARIAL    ';
  OTHERWISE ... ;
END;
```

Another option for coding table lookup operations is to write a user-defined SAS Format. The advantages of using Formats for Account Code Determination routines over other techniques are:

- Can reduce DATA step compile time by replacing lengthy inline IF/THEN/ELSE or SELECT/WHEN logic
- Consistent response time for lookup operations, since Formats are constructed as a highly optimized binary tree
- Account code data is separated from the code itself, affording easier maintenance

PROC FORMAT structure differs significantly from an IF cascade. PROC FORMAT causes a table to be built when the PROC FORMAT is executed. The table is inspected through a SAS PUT function, similar to the PL/I ENCODE operation. The Format might be structured as follows:

```
PROC FORMAT;
 VALUE $USRACTF
 'ACP' = 'ACCOUNTS PAYABLE'
 'PAY' = 'PAYROLL'
 'ACR' = 'ACCOUNTS RCVBL'
 'SYS' = 'SYSTEMS SUPPORT'
 'ACT' = 'ACTUARIAL'
 OTHER = 'OVERHEAD' ;
```

This causes the table to be built. The Format may be built at any time--periodically (such as every time the source table changes) or each day in the course of the daily run. Table lookup operations are done when the PUT function is called. A sample account code derivation routine that illustrates the use of a table lookup using PUT follows.

```
  ACCTNO1 = PUT( SUBSTR(JOB,1,3),$USRACTF.);
```

The PROC FORMAT examples illustrate how to create and use a temporary user-defined Format. Section 4.5 of this guide provides instructions on creating permanent user-defined Formats.

Remember to always benchmark one coding method against another, choosing the method that offers the best performance for your CA MICS system.

Chapter 5: CA MICS/SAS Coding Conventions

This chapter contains a set of conventions for coding in the SAS language since you will use SAS to code CA MICS exits and modifications. We have also included a set of CA MICS-unique considerations for efficient integration of user concepts into the CA MICS operational flow.

You can help to ensure that the code for your modifications and exits is accurate by following the guidelines in this chapter closely. The material presented here is based on user experience and periodic performance studies. Each item has a reason for being noted, and is well worthy of your examination.

This section contains the following topics:

5.1 SAS Coding (see page 263)
5.2 CA MICS Coding (see page 268)

5.1 SAS Coding

SAS is a program product of SAS Institute Inc. Almost all of the logic of CA MICS is programmed in this powerful language.

The SAS language is interpretive; that is, each program is recompiled each time it is executed using a compiler technique. The interpretive nature of the language has several advantages:

- Run-time modifications to CA MICS can be made as a matter of course, with no special operational considerations. It is in the nature of CA MICS to respond to certain execution definitions that may change from run to run. When such a change happens, CA MICS can construct new SAS code appropriate to the situation at hand and execute the new code (in addition to the preprogrammed code).

- User modifications are incorporated into CA MICS
  immediately, with no change to the operational flow.

- There is a single source of compile-time and CA MICS
  execution-time diagnostic messages. This is desirable
  from the aspect of providing timely product support and
  problem diagnosis and correction.

As with any computer language, SAS has some constructions that are more efficient than others. The information that follows highlights considerations of SAS language coding that are in good SAS form and constitute particularly efficient constructions. Following these SAS programming guidelines will result in more readable, more efficient SAS code for any SAS application, including CA MICS.

Macros are used in CA MICS as a way to simplify repetitive coding. Refer to Section 4.3.1 in this guide for information about SAS Macro Statements and SAS Macro Language.

Maintainability Standards

COMMENTS:

SAS supports the use of three comment facilities. If a statement begins with an asterisk (*) and ends with a semicolon (;), SAS ignores the statement. Such comments must not be within other statements. For example,

```
 ...
X=5;
* THIS IS A COMMENT ;         This  comment  construction
Y=3;                          is valid.
...

...
X=5 * THIS IS A COMMENT ;     This  comment  construction
Y=3;                          is invalid.
...
```

The second form of comment is in the form of comments originally found in the IBM PL/I language and used in many of today's programming languages. Any string of characters enclosed by a slash-asterisk, asterisk-slash is treated as a comment anywhere it occurs, except inside text quotes.

```
...
X=5 /* THIS IS A COMMENT */ ;   This comment construction
Y=3;                            is valid.
...

...
A='/* THIS IS A STRING */';     This is invalid.
...
```

Comments of both types may be found in CA MICS code, but most comments are of the "/* - */" variety. This form is used because of the great advantage of being able to put the comment inside a SAS statement, especially long INPUT statements.

A third form of comment, which is only available from within the SAS Macro language, is similar to first comment form. If the Macro language statement begins with a percent-asterisk (%*) and ends with a semicolon (;), the SAS Macro language ignores the statement. Such comments must not be within other statements. For example,

```
%MACRO name ;
%*  THIS IS A COMMENT ;       This  comment  construction
...                           is valid.
%MEND  name ;
```

Below are some guidelines that help provide enough comments without overburdening the reader:

- Use comment blocks for logic descriptions, such as the following:

```
/* ********************** */
/* THE FOLLOWING STATEMENTS */
/* APPLY TO TYPE 30 SMF     */
/* RECORDS ONLY.            */
/* ********************** */
```

```
    - Include a comment in SAS code that changes, showing
      the date of the change, the purpose of the change,
      and the person who coded the change.

    - Where practical, follow the indentation conventions
      when coding comments.

    - Have the comment contain more than just a word for word
      translation of a logic statement.  The comment
       X=5;  /* SET X TO 5 */
      tells the reader nothing.

    - Refer to supporting material in comments whenever
      possible, such as manual numbers, product release IDs,
      or other parts of the program's code.

    - If you are not sure that a comment is necessary, insert
      the comment.  Too many comments may be cumbersome, but
      too few can be a greater problem.

INDENTATION:

Good indentation as described below increases the readability
of a program effectively and inexpensively.

    - Labels begin in column 1.

    - Each level of code is indented 2 columns in from the
      previous level.

    - ELSE statements align with their IF.

    - Continued IF or ELSE statements are indented.

    - END statements align with their DO.

    - DO statements following an IF or ELSE are on the same
      line as the IF or ELSE, and their END aligns with the
      IF or ELSE.
```

The following hypothetical example illustrates each of the guidelines listed above.

```
+-------------------------------------------------+
|                                                 |
|GENSETRC:                                        |
|                                                 |
|INTERVLS=0;                                      |
|SAMPLES=SETSAMPS;                                |
|TRCNOSET=1;                                      |
|TRCTYPE='S';                                     |
| /* COMPUTE APPROXIMATE SET DURATION */          |
|IF SETS > 0 THEN DURATION=DURATION/SETS;         |
|ELSE DURATION=0;                                 |
|                                                 |
| /* CHECK FOR SAMPLE FIELD LENGTH */             |
|IF FLDLENG=4 THEN DO;                            |
|  DO X=1 TO SETS;                                |
|    IF TRCMNFLG THEN INPUT TRCMN PIB4. @;        |
|    IF TRCSMFLG THEN INPUT TRCSUM PIB8. @;       |
|    TRCSSQ=TRCSSQ+(SSQ1*16**9);                  |
|    IF TRCEVFLG THEN INPUT TRCENDVA PIB4. @;     |
|    IF SETS=X THEN SAMPLES=LSTSAMPS;             |
|    IF TRCSMFLG AND SAMPLES > 0 THEN             |
|      TRCAV=TRCSUM / SAMPLES;                    |
|    ELSE TRCAV=.;                                |
|    TRCSETNO=X;                                  |
|    OUTPUT SCPTRC00 ;                            |
|  END;                                           |
|END;                                             |
|ELSE IF FLDLENG=2 THEN DO;                       |
|  DO X=1 TO SETS;                                |
|    IF TRCMNFLG THEN INPUT TRCMN PIB2. @;        |
|    IF TRCSMFLG THEN INPUT TRCSUM PIB6. @;       |
|    IF TRCMXFLG THEN INPUT TRCMX PIB2. @;        |
|    IF SETS=X THEN SAMPLES=LSTSAMPS;             |
|    IF TRCSMFLG AND SAMPLES > 0 THEN             |
|      TRCAV=TRCSUM / SAMPLES;                    |
|    ELSE TRCAV=.;                                |
|    TRCSETNO=X;                                  |
|    OUTPUT SCPTRC00 ;                            |
|  END;                                           |
|END;                                             |
|RETURN;                                          |
+-------------------------------------------------+
```

Performance Guidelines

Refer to the manuals "SAS Programming Tips: A Guide to Efficient SAS Processing," "Efficiency, Improving the Performance of your SAS Applications," and "Tuning SAS Applications in the MVS Environment" from SAS Institute.

5.2 CA MICS Coding

CA MICS has been designed and implemented to perform a wide range of functions. However, CA MICS also recognizes the need for limited user modification of the standard product to accommodate unusual circumstances.

Almost all of the logic of CA MICS is programmed in SAS. Most of the modifications you make will be in SAS.

There are other languages involved in CA MICS. Some assembler code is involved, notably in the CA MICS IMS Analyzer. CA ISPF Dialog Assist and ISPF panel definition logic are used in the CA MICS Workstation Facility (MWF). CA MICS installation and operational JCL are coded in a suitable CA Incorporated language called GCL (Generation Control Language). CA MICS products are generated from a kind of language called Component Generator Definition Statements.

The principles of each of these languages are given in other sections of this document.

Guidelines for coding CA MICS modifications within the framework of CA MICS design targets in the most efficient manner are described below.

Maintainability Standards

Maintainability standards established for CA MICS include tracking user modifications, using comments in code, parameter library coding, and pointers for coding CA MICS exits using SAS. Each topic will be addressed in turn.

One of the most important aspects of making user modifications to CA MICS is the ability to track the actual changes. Such tracking is necessary to:

- verify the logic added in case of questionable results
- determine the impact of vendor-supplied product maintenance on the change
- develop a plan for merging modified data with unmodified data for long-term reporting

The CA MICS System Administrator Guide is the central place for coordinating product maintenance. This document contains a section called User Modification Descriptions. Use this section to log ALL user modifications at the earliest possible time. In the event CA MICS product support is needed at your site, an accurate and timely record of user-applied changes can greatly simplify problem determination and correction.

The User Modification section of the System Administrator Guide is organized in two parts: an index to the changes and a summary description of each change.

The index is a list of user modifications. The index has columns for a change number (user-assigned), date of the last entry for the change, status of the change, whether the change has been superseded by another modification, and other notes that apply to the change.

Creative use of the status column of the index provides a vehicle for entering changes into the System Administrator Guide even before they are implemented. Such logging would help identify possible change interference situations earlier.

The User Modification Description forms in this section provide a handy method for recording the INTENT as well as the logic and implementation of a change.

```
The content of each information category of the User
Modification Description log sheet is free-form.  However,
the contents of each section typically contain the following:

 Log Topic                  Typical Contents
 -------------------------  -------------------------------

 MODIFICATION TITLE:        A one-line title describing the
                            intent of the modification.

 MODIFICATION NUMBER:       A user-assigned number, such as
                            UMOD-001, indicating a user
                            modification to CA MICS, number
                            1 in a series.

 MODIFICATION AUTHOR:       Your name.

 MODIFICATION DATE:         Implementation date.

 MODIFICATION DESCRIPTION:  A meaningful description of the
                            intent and implementation method
                            of the change.  Any data that is
                            possibly pertinent should be
                            included, such as whether the
                            change has a fixed useful life
                            and should be removed at a later
                            date.

 REASON FOR MODIFICATION:   A SHORT history of the need for
                            the change, including such items
                            as requestor ID, date the need
                            was identified, reference to
                            other existing code that uses
                            the method, and reference to the
                            method's source.

 CA MICS MODULES MODIFIED:  List all CA MICS modules
                            modified by the change, and the
                            libraries on which they reside.

 MODIFICATION SOURCE:       The location of the IEBUPDTE
                            stream that applied the change.


Comments
```

The considerations described for coding comments in SAS code (Section 5.1 of this guide) also apply to coding comments in CA MICS code.

Additional guidelines for commenting CA MICS code are:

- Include a comment for each change showing the date of the change, the purpose of the change, and the person who coded the change.

- Log the change in the User Modification section of the CA MICS System Administrator Guide. Accurate and timely logging of these changes is absolutely necessary to assure your ability to install normal vendor maintenance.

The format of comments changes, according to the language being used. Code comments in the format that applies only to the language you are modifying.

- SAS comment formats are described in the previous section.

- Assembler language recognizes comments on a code line as following the operands. Comment lines can be coded by putting an asterisk (*) in column 1 of an assembler statement.

- All CA MICS parameter library members that do not contain JCL or prototype JCL have an asterisk comment facility. If the first non-blank character on any record in the PARMS library member is an asterisk (*), the line is treated as a comment. The asterisk must be the first non-blank character on the line, but does not have to be in column one of the line.

- PROTOLIB members and members of the PARMS library that contain JCL may contain comments at two levels. Code comments that you want to show in the generated JCL with "//@" beginning in column 1 of the line. Code comments that you want in the member but NOT in the generated JCL with ":*" beginning in column 1 of the line. For example,

```
+-------------------------------------------+
|                                           |
|:* THIS COMMENT WILL NOT APPEAR IN         |
|:* GENERATED JCL                           |
|//@ COMMENT FOR JCL                        |
|//INPUTSMF DD ...etc                       |
|                                           |
+-------------------------------------------+
```

- Code comments in CLIST code and MICF panel code with "/*" as the first and second non-blank characters in any line, and not necessarily in column 1 of the line.

- Code comments in MICF dialog skeletons with ")CM " beginning in column 1 (i.e., a right parenthesis in column 1, the letters CM in columns 2 and 3, and a BLANK SPACE in column 4).

CA MICS Parameter Library Member Coding

The prefix.MICS.PARMS library and the sharedprefix.MICS.PARMS library contain a number of members which specify the options to control various parts of the total CA MICS system. Members either contain control statements or are exit routines coded in SAS.

The format of control statements is free-form but positional. This means that any number of blanks many come before and after any variable in a parameter member, but all variables must be specified and specified in the order shown. There is no explicit provision for the continuation of control cards. Our approach has been to allow the repetition of control cards when more variables must be specified than can fit on a single card. Comments are supported in all of the PARMS library members. If the first "variable" is "*", the card is ignored. For those parameter statements that begin with a keyword, the keyword MAY NOT BE ABBREVIATED--it must be coded exactly as shown in the documentation.

CA MICS SAS Coding

In writing SAS code for use with CA MICS, remember the
following guidelines:

- Validate the fields your exits use to make processing
  decisions. While CA MICS does some basic validation,
  it cannot catch the kinds of installation specific
  errors you can (such as an invalid account number on a
  JOB card).

  When one of your exits detects invalid input data, it
  should not pass it into the database because doing so
  increases the size of the database and decreases its
  usefulness for analysis and reporting.

  Rather, your exits should take a default action when
  they detect invalid input data. For example, you could
  assign a job with an invalid account code to an
  overhead category.

- Test your exit code. Exit code is inserted directly
  into CA MICS processing logic, so you should ensure
  that this inserted code will cause subsequent CA MICS
  logic to operate abnormally. Observing the following
  DOs and DON'Ts should assure proper implementation of
  exits:

          **** D O the following: ****

DO: Use the member supplied in the distributed version of
     the CA MICS.PARMS library as the starting point for
     your exit, where appropriate.

DO: Liberally comment all code.

DO: Examine input record length to be sure INPUT requests
     can be satisfied from the current record.

DO: Refer to input field locations with pointer logic
     where possible.

DO: Preserve data element integrity by type (numeric,
     time stamp, character variable) and range (having the
     value agree with the FORMAT and LENGTH specs).

```
DO:  Define new retained data elements by specifying the
     RETAIN statement after LENGTH and FORMAT statements,
     if applicable.

DO:  Test all exit logic under the CA MICS-required SAS
     release.

DO:  Call CA Technical Support if you have ANY questions.

DO:  Code percent sign constants as double %% if such
     constants are necessary in code that appears in a
     CA MICS SAS Macro statement.

          ****  D O N ' T  do the following:    ****


DON'T:  Reset SAS compilation options by using the SAS
        OPTIONS statement without examining the impact of
        such a change on the rest of CA MICS logic.

DON'T:  Use columns 73-80 for SAS statements. CA MICS only
        uses 1-72.

DON'T:  Code these SAS statements in your exit:  RETURN,
        DELETE, or INPUT without a trailing "@".

DON'T:  Code INPUT statements that could cause SAS to
        perform an automatic input statement skip.  Use
        the RECLEN variable to be sure the record is big
        enough to contain data at any intended input
        offset.

DON'T:  Forget the CA MICS naming standards, especially
        for DDnames, file names, and data element file
        prefixes.

DON'T:  Assume that any data element is RETAINed or not
        RETAINed.

DON'T:  Redefine any currently-existing CA MICS data
        element unless the documentation of the specific
        exit in question says it is allowed.

DON'T:  Assume that the SAS automatic variable _N_
        contains the actual input observation number.
```

DON'T: Assume that setting the SAS automatic variable _ERROR_ will cause an input observation to be ignored or CA MICS processing to terminate.

DON'T: Use SAS language features not supported by the SAS release specified for use by the CA MICS update process.

Chapter 6: Database Tailoring and Retrofitting Techniques

The CA MICS Component Generator's database tailoring facility enables the System Administrator to control database content and format, thereby reducing DASD space requirements and run times. The kinds of changes that can be made using the database tailoring facility include:

- o adding elements to CA MICS files
- o changing the timespans in which files are maintained
- o defining which files are included in the Audit and History Archive Files
- o modifying the sequence/summary characteristics of CA MICS files
- o renaming common elements
- o activating data compression

The database tailoring facility also updates the file granularity charts and data elements lists that print as part of the files documentation for each product and are also available online.

The examples in this chapter are based on a network performance analysis (NPA) component that was superseded entirely by the CA MICS SNA Network Analyzer. The references to the NPA Component in this chapter provide an illustration of how such tailoring operations would apply to any hypothetical CA MICS component.

1 - Tailoring Guidelines
2 - Tailoring the Database
3 - Retrofitting Guidelines

This section contains the following topics:

6.1 Tailoring Guidelines (see page 278)
6.2 Tailoring the Database (see page 281)
6.3 Retrofitting Guidelines (see page 349)

6.1 Tailoring Guidelines

The first step in tailoring your CA MICS system is to ensure that you clearly understand your data center's data requirements. Knowing your requirements, the following questions and answers should help you plan your tailoring strategy.

When should database tailoring be performed?

---

o Use the product for at least six months to ensure that you have enough experience to determine what kinds of changes should be made to the database to achieve your objectives.

  If implementing compression is a consideration, one month's data is sufficient if you use the Administrator Utilities provided for analyzing and recommending compression settings.

o If installing a new product requires a disproportionately large amount of DASD space, such that it may not be practical or even feasible to install, evaluate the use of the DBSPLIT facility and ultimately the database tailoring facility to reduce the number of data elements and possibly files that are to be maintained in the database.

Can a product be tailored differently in different database units?

---

o A product is generated at the database complex level and therefore its files are identical in all database units. You can tailor file compression by unit, however, using the prefix.MICS.PARMS(COMPRESS) member.

What other dependencies must be considered when tailoring the CA MICS database?

---

o If a change is made that adds or deletes a file in any of the three archiving systems (audit, weekly history, or monthly history), you must regenerate the database unit's operational jobs by running a JCLGEND in each unit in which the product is installed.

- If compression is activated for a file by adding COMPRESS statements to the cccGENIN member of sharedprefix.MICS.GENLIB, run a COPYLIBU to add FILE statements to prefix.MICS.PARMS(COMPRESS).

- If a change is made to the sort order of a file in the WEEKS, MONTHS, or YEARS timespans (by changing the timespan status definition on the NAME or NAMX statement), retrofit the sequence change in the files already present in the online and archive databases. (Each requires a sort and summarization.) A component generation (cccCGEN) is required to implement the change.

- Review the use of elements before deactivating them from the database. For example, if a data element is marked inactive (deleted) and that data element is used in a critical SAS statement (such as a BY statement) in a dependent program, the SAS compilation will not complete successfully. If the data element is referenced in any other manner, the SAS Log will show that the variable was not initialized, and dependent elements, if any, may be incorrect.

- To add a data element to the database, define it to the Component Generator (sharedprefix.MICS.GENLIB member cccGENIN) and add source code to the DYcccFMT input format routine in sharedprefix.MICS.SOURCE. Then edit sharedprefix.MICS.SOURCE(#cccEXIT)'s USRFMT2 and USRFMT3 macros to ensure that the source data used to provide values to the new data element is available.

Evaluate any change in the generation control statements in light of the information provided in the rest of this chapter. For example, you may not be able to directly activate the DETAIL timespan for files that do not normally exist in the DETAIL timespan. Check the specific instructions in the following sections for details, as well as the modification documentation in specific product guides.

Read this chapter before performing any file tailoring.

File tailoring can be performed easily by following the guidelines in this section. The approximate effort involved for file tailoring is illustrated below.

Tailoring action	Required operations

To add, change, or delete a data element:	
- The data element is not in the sequence key of the file.	- Component generation - User modification to the component format routine DYcccFMT - Test/implement/check
- The data element is in the sequence key of the file.	- Component generation - User modification to the component format routine DYcccFMT - Database retrofit for modified key structure - Test/implement/check
To add a file to an online database timespan:	- Component generation - User modification to the component format routine DYcccFMT - Test/implement/check
To delete a file from an online database timespan:	- Component generation - Delete unused cycles - Test/implement/check
To add or delete a file from archiving:	- Component generation - JCLGEND for WEEKLY, MONTHLY, and GDGSGEN JCL - Run GDGSGEN - Test/implement/check
To compress or uncompress a file's data:	- Edit the COMPRESS member in prefix.MICS.PARMS - Run CYCLEGEN - Run BACKUP - Run RESTORE

6.2 Tailoring the Database

A great amount of site tailoring can be accomplished using the CA MICS parameter library members. For example, most generally used data selection criteria are addressed by CA MICS parameters.

Tailoring the database, as explained in the sections that follow, is the next level of detail. For example, deleting data elements from a file, adding others, creating the file in timespans that are not normally supported, and modifying the CA MICS archive process are not changes that can conveniently be made by the novice user.

The changes described here must be tested and verified to ensure the integrity and usability of the entire CA MICS database.

Following these guidelines will help you tailor your CA MICS database. If, after you read these sections, you still have a question about the feasibility or practicality of an intended change, please call the CA MICS Product Support Group for assistance.

Product tailoring is performed in levels that relate to the structure of the definition of a component. A component is defined by Component Generator Statement sets:

```
Component and generation control statements
- Information area within the component
  - File within the information area
    - Timespans in which the file exists
    - Archiving for the file
    - Data elements in the file
      - Journaling data elements in the file
      - data element 1
      - data element 2
      - ...
      - data element n
```

The lowest level of tailoring involves changing data element statements to modify the data elements within the file. Common data elements (common to many files, described in a special common description area) may be modified in many files in one stroke.

The next level of modification is to modify the file's operational characteristics. Online files may be modified to change the timespans in which the file exists or the sequence of the file within the timespan. Offline files may be modified to change the archiving characteristics of the file.

The examples in the following sections are based on the NPANCP file shown in chapter 4's MCG Illustrations section.

6.2.1 Tailoring Data Elements

The lowest level of tailoring involves changing data element statements to modify the data elements within the file. There are two types of data elements: common and standard.

Common data elements are further defined. Elements common to files across products are defined in a special common description area, and elements common to files in one product are defined in that product's cccGENIN member.

Standard data elements are elements that are used by one standard file in one product. (If the standard file has parallel files, the standard data elements are available to the parallel files as well.)

Refer to Chapter 10 of your product's guide for special considerations and more information on customizing data elements.

The statements listed below define the Network Performance Analyzer (NPA) component with one file, the Network Control Program (NCP) file. This statement would be found in the product's sharedprefix.MICS.GENLIB(cccGENIN) member. Data element definition statements that can be modified by the user are indicated by "-->" in the left margin of the example.

```
   *
   * NPA component generator statements
   *
   GEN GENFILES
   COMP NPA 050 32000 VBS NOACCT NPA Component
-->TYPE XR  3 . 3 . 3 .
-->NAME NPANRRLS  Receiving Line Speed (line/link)
-->NAME NPANRSLS  Sending Line Speed (line/link)
-->TYPE XA  5 TIME11.2  6 TIME15.2  6 TIME15.2
-->NAME NPATMAOT  Actual Observation Time
-->TYPE XR  4 TIME8.  4 TIME8.  4 TIME8.
-->NAME NPATSEIS  Expected Interval Time
-->NAME NPATSFRU  Foreward RU Send Time
-->NAME NPATSLOG  Time Record Logged To SMF
   AREA NPA NPA Activity Information Area
   FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
-->TYPE R 3 . 3 . 3 .
-->NAME NCPNRFBQ  00  0 0 0 0 0 Free Buffer Queue Length
-->NAME NCPNRHQL  00  0 0 0 0 0 NCP Channel Hold Queue Length
-->NAME NCPNRIQL  00  0 0 0 0 0 NCP Channel Intrmd Queue Len.
-->NAME NCPNRSFB  00  0 0 0 0 0 Free Buffer Count At Slowdown
```

```
-->TYPE A 5 TIME11.2 6 TIME15.2 6 TIME15.2
-->NAME NCPTMFCT  00  0 0 0 0 0 Free Cycle Time
-->NAME NCPTMTIS  00  0 0 0 0 0 Time In Slowdown
-->TYPE M 3 . 3 . 3 .
-->NAME NCPMXANP  00  0 0 0 0 0 Max Available NCP Buffers
-->NAME NCPMXFBH  00  0 0 0 0 0 Free Buffer High Water Mark
-->TYPE N 3 . 3 . 3 .
-->NAME NCPMNFBL  00  0 0 0 0 0 Free Buffer Low Water Mark
-->NAME RESNAME   99  0 0 0 0 0 .
-->NAME DAY       99  5 5 N N N .
-->NAME HOUR      99  6 6 6 N N .
-->NAME WEEK      99  0 0 4 N N .
-->NAME YEAR      99  3 3 3 3 3 .
-->NAME MONTH     99  4 4 N 4 N .
-->NAME INTERVLS  99  0 0 0 0 0 .
-->NAME DURATION  99  0 0 0 0 0 .
-->NAME STARTTS   99  0 0 0 0 0 .
-->NAME ENDTS     99  7 0 0 0 0 .
-->NAME ZONE      99  0 0 5 5 4 .
-->NAME DAYNAME   99  0 0 N N N .
-->NAME SYSID     99  1 1 1 1 1 .
-->NAME NCPNAME   99  2 2 2 2 2 .
-->NAME NETNAME   99  0 0 0 0 0 .
```

The NAME statements whose data element option code is 99 are common data elements available to all CA MICS files. Data elements that are common to this product's files are defined prior to the first AREA statement in the product's cccGENIN member.

Common data elements available to all CA MICS files are described in the generation library member $DEGENIN. The statements below describe those data elements, and are extracted from $DEGENIN. These data element definitions can all be user-modified.

```
TYPE XR  2  .  2  .  2  .
NAME DAY      Day of Month
TYPE XR  $3  .  $3  .  $3  .
NAME DAYNAME  Name Of Day Of Week
TYPE XA  5  TIME11.2  6  TIME15.2  6  TIME15.2
NAME DURATION Recording Interval Time
TYPE XM  7  DATETIME19.2  7  DATETIME19.2  7  DATETIME19.2
NAME ENDTS    End Time Stamp
TYPE XR  2  .  2  .  2  .
NAME HOUR     Hour Of Day
TYPE XA 3 . 4 . 6 .
NAME INTERVLS Number of Recording Intervals
TYPE XR  2  .  2  .  2  .
```

```
NAME MONTH    Month of Year
TYPE XR  $8  .  $8  .  $8  .
NAME NCPNAME  Network Ctl Program Name
TYPE XR  $8  .  $8  .  $8  .
NAME NETNAME  Resource Network Name
TYPE XR  $24  .  $24  .  $24  .
NAME RESNAME  Resource Generic Name
TYPE XN  7  DATETIME19.2  7  DATETIME19.2  7  DATETIME19.2
NAME STARTTS  Start Time Stamp
TYPE XR  $4  .  $4  .  $4  .
NAME SYSID    System Identifier
TYPE XR  2  .  2  .  2  .
NAME WEEK     Week Of Year
TYPE XR  2  .  2  .  2  .
NAME YEAR     Year of Century
TYPE XR  $1  .  $1  .  $1  .
NAME ZONE     Time Zone

The following sections describe specific data element
tailoring for each data element type:

    1 - Standard Data Elements
    2 - Common Data Elements
```

6.2.1.1 Standard Data Elements

As discussed in the prior section, standard data elements are data elements that belong to a standard CA MICS file and its parallel files, if any exist.

Most standard data elements begin with the three character identifier of the file to which the data element belongs. For example, the data element NCPMNFBL begins with NCP, showing that it belongs to the NPANCP file.

6.2.1.1.1 Deleting Data Elements

```
There are two facilities available to delete a data element
from a file:

    - The timespan indicators on the CA MICS Component
      Generator (MCG) NAME statement can be used to delete a
      data element from one or more timespans.

    - If a data element belongs to a selectable cluster of
      data elements, the entire selectable cluster can be
      deleted from all timespans of all files within a
      component by modifying the MCG OPTION statement.

*******************************
* NAME Statement Modifications *
*******************************


The example file definition contains the Component Generator
Definition Statements:

...
TYPE N 3 . 3 . 3 .
NAME NCPMNFBL  00  0 0 0 0 0 Free Buffer Low Water Mark

...

Assume that the Free Buffer Low Water Mark had no useful
purpose at your site.  If this was the case, this data
element could be eliminated from the file by changing its
NAME statement to:

...
TYPE N 3 . 3 . 3 .
NAME NCPMNFBL  00  N N N N N Free Buffer Low Water Mark
...

This would remove the data element from all timespans in
which the file appears.  However, in the case of the NPANCP
file, on which this data element appears, the FILE itself is
not active in the YEARS timespan.  In this case,

...
TYPE N 3 . 3 . 3 .
NAME NCPMNFBL  00  N N N N 0 Free Buffer Low Water Mark
...
```

would accomplish the same end, because the whole file does not exist in the YEARS timespan, and the data element is not on any timespan in which the file does exist.

In fact, the data element could be deleted from any timespan at all. This is based on the timespan indicators on the NAME statement. For example, there are cases when

```
...
TYPE N 3 . 3 . 3 .
NAME NCPMNFBL  00  0 N 0 N 0 Free Buffer Low Water Mark
...
```

would be valid and produce the desired results. There are cases in which such a specification would cause timespans other than DETAIL to contain spurious data for NCPMNFBL, also.

The timespan indicators on the NAME statement for each data element must consider the timespan indicators on the FILE statement. If the following combination exists,

```
FILE ... Y N Y N Y ...
...
NAME ... 0 N 0 N 0 ...
```

the data element would be summarized properly. The file's WEEKS timespan would be constructed from summarized DETAIL data, and the YEARS would be constructed from summarized DETAIL data. However, the following construction creates problems:

```
FILE ... Y Y Y Y Y ...
...
NAME ... 0 N 0 N 0 ...
```

Code generated by the MCG would cause summarization for the WEEKS timespans to be driven (summarized) from the DAYS timespan. The summarization to the YEARS timespan is driven from the MONTHS timespan of the file.

The DAYS timespan is used as the source of WEEKS and MONTHS summarization because the DAYS timespan has fewer observations than the DETAIL timespan, and the summarization process is more efficient. The same principle causes MONTHS to be the source of YEARS.

```
However, if the data element has been dropped from the
timespan from which the new timespan is being summarized,
and the data element is any type other than CALCULATED, the
summarization will be based on SAS missing values, and will
not produce the desired results.

The following combinations of FILE and NAME statement
combinations are valid for data elements that are not
calculated:

FILE timespan indicators    NAME timespan indicators
-------------------------   --------------------------------

Y N - Y Y                   0 - - 0 -    where "-" can be any
                            0 - - N N    valid indicator.
                            N N N N N

Y N - N N                   0 - - - -
                            N N N N N

Y Y - Y Y                   0 0 - 0 -
                            0 0 - N N
                            0 N N N N
                            N N N N N

Y Y - N N                   0 0 - - -
                            0 N N N N
                            N N N N N

N E - - -                   - 0 - - -
                            - N N N N

N N N N N                   - - - - -

Data elements that are calculated are identified by the data
element type code in the TYPE statement for the type group to
which the data element belongs.  Our example is
```

```
...
TYPE N 3 . 3 . 3 .
NAME NCPMNFBL  00  0 0 0 0 0 Free Buffer Low Water Mark
...
```

and the data element is noted as a type N, for minimum. Calculated data element types are type C, CN, XC, or XCN. These standard data elements may be dropped from any timespan by modifying the timespan indicators on the NAME statement IF the data elements necessary to calculate the data element's value in its active timespans are present on those timespans. See the MCG statement section of this document for more information.

There is one other way to delete a data element. If a data element is ESSENTIAL, it is active (as marked by the timespan indicators) even if the ESSENTIAL option is set to INITIAL or ONLY. See the MCG statement section COMPOPT and FOPT for more information on the values of the ESSENTIAL option.

If a data element is ESSENTIAL, it can made nonessential by changing the NAME statement keyword to NAMX, as follows:

```
NAME NCPMNFBL  00  0 0 0 0 0 Free Buffer Low Water Mark
```

becomes

```
NAMX NCPMNFBL  00  0 0 0 0 0 Free Buffer Low Water Mark
```

This change causes the data element to be inactive if the ESSENTIAL options INITIAL or ONLY were used. See the description of the NAME statement in Section 4.2.4.4.3 for more information on coding NAMX.

```
*********************************
* OPTION Statement Modifications *
*********************************
```

The original form of the NPA component's definition shown earlier began:

```
   *
   * NPA Component generator statements
   *
   GEN GENFILES
   COMP NPA 050 32000 VBS NOACCT NPA Component
   TYPE XR  3 . 3 . 3 .
   NAME NPANRRLS  Receiving Line Speed (line/link)
   NAME NPANRSLS  Sending Line Speed (line/link)
   TYPE XA  5 TIME11.2  6 TIME15.2  6 TIME15.2
   NAME NPATMAOT  Actual Observation Time
   TYPE XR  4 TIME8.  4 TIME8.  4 TIME8.
   NAME NPATSEIS  Expected Interval Time
   NAME NPATSFRU  Forward RU Send Time
   NAME NPATSLOG  Time Record Logged To SMF
   AREA NPA NPA Activity Information Area
   FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
   TYPE R 3 . 3 . 3 .
   NAME NCPNRFBQ  00  0 0 0 0 0 Free Buffer Queue Length
   NAME NCPNRHQL  00  0 0 0 0 0 NCP Channel Hold Queue Length
   NAME NCPNRIQL  00  0 0 0 0 0 NCP Channel Intrmd Queue Len.
   NAME NCPNRSFB  00  0 0 0 0 0 Free Buffer Count At Slowdown
   TYPE A 5 TIME11.2 6 TIME15.2 6 TIME15.2
   NAME NCPTMFCT  00  0 0 0 0 0 Free Cycle Time
   NAME NCPTMTIS  00  0 0 0 0 0 Time In Slowdown
   ...
```

This component has no selectable data element clusters, and therefore the standard generator statements do not include an OPTION statement.

Assume that there is a valid cluster ID called NET1 and its cluster number is 47. The OPTION statement would be included in the NPA definition member, and data elements belonging to the cluster would have a cluster code of 47 coded in their NAME statements, as follows:

```
   *
   * NPA Component generator statements
   *
   GEN GENFILES
-->OPTION NET1
   COMP NPA 050 32000 VBS NOACCT NPA Component
   TYPE XR  3 . 3 . 3 .
   NAME NPANRRLS  Receiving Line Speed (line/link)
   NAME NPANRSLS  Sending Line Speed (line/link)
   TYPE XA  5 TIME11.2  6 TIME15.2  6 TIME15.2
   NAME NPATMAOT  Actual Observation Time
   TYPE XR  4 TIME8.  4 TIME8.  4 TIME8.
   NAME NPATSEIS  Expected Interval Time
   NAME NPATSFRU  Forward RU Send Time
   NAME NPATSLOG  Time Record Logged To SMF
   AREA NPA NPA Activity Information Area
   FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
   TYPE R 3 . 3 . 3 .
   NAME NCPNRFBQ  00  0 0 0 0 0 Free Buffer Queue Length
-->NAME NCPNRHQL  47  0 0 0 0 0 NCP Channel Hold Queue Length
-->NAME NCPNRIQL  47  0 0 0 0 0 NCP Channel Intrmd Queue Len.
   NAME NCPNRSFB  00  0 0 0 0 0 Free Buffer Count At Slowdown
   ...
```

Deleting the entire cluster can be accomplished by changing the OPTION keyword NET1 to NONET1. All data elements that belong to the cluster (including NCPNRHQL and NCPNRIQL) would be deleted from all timespans of all files in the NPA Component:

```
   *
   * NPA Component generator statements
   *
   GEN GENFILES
-->OPTION NONET1
   COMP NPA 050 32000 VBS NOACCT NPA Component
   ...
```

See the individual product guides for a list of cluster names used in a specific cccGENIN member.

```
****************************
* Making the Modifications *
****************************
```

To delete a data element from one or more timespans, if the data element is not a sequence data element in the file, perform the following:

- Modify the timespan indicators on the NAME statement for that data element, if necessary.

- Modify the option selection codes on the OPTION statement, if necessary.

- Run the component generation for the component in question.

- Drop the data element from the file in the desired timespan(s) since the next daily update for any database unit in which the component has been installed reflects the change.

To delete a data element from one or more timespans if the data element IS a sequence data element in the file, see the sections in this chapter on modifying the sequence of a file. Modify the sequence, and follow the instructions above to delete the data element.

6.2.1.1.2 Adding New Data Elements

There are two facilities available to add a data element to a file:

- If a data element belongs to a selectable cluster of data elements, the entire selectable cluster can be activated in all timespans of all files within a component by modifying the MCG OPTION statement.

- New data elements can be added to individual files within a component by coding CA MICS Component Generator (MCG) TYPE group statements to describe the data elements for each file in which they are to appear.

```
*********************************
* OPTION Statement Modifications *
*********************************
```

The original form of the NPA Component's definition shown earlier began as follows:

```
*
* NPA Component generator statements
*
GEN GENFILES
COMP NPA 050 32000 VBS NOACCT NPA Component
TYPE XR  3 . 3 . 3 .
NAME NPANRRLS  Receiving Line Speed (line/link)
NAME NPANRSLS  Sending Line Speed (line/link)
TYPE XA  5 TIME11.2  6 TIME15.2  6 TIME15.2
NAME NPATMAOT  Actual Observation Time
TYPE XR  4 TIME8.  4 TIME8.  4 TIME8.
NAME NPATSEIS  Expected Interval Time
NAME NPATSFRU  Forward RU Send Time
NAME NPATSLOG  Time Record Logged To SMF
AREA NPA NPA Activity Information Area
FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
TYPE R 3 . 3 . 3 .
NAME NCPNRFBQ  00  0 0 0 0 0 Free Buffer Queue Length
NAME NCPNRHQL  00  0 0 0 0 0 NCP Channel Hold Queue Length
NAME NCPNRIQL  00  0 0 0 0 0 NCP Channel Intrmd Queue Len.
NAME NCPNRSFB  00  0 0 0 0 0 Free Buffer Count At Slowdown
TYPE A 5 TIME11.2 6 TIME15.2 6 TIME15.2
NAME NCPTMFCT  00  0 0 0 0 0 Free Cycle Time
NAME NCPTMTIS  00  0 0 0 0 0 Time In Slowdown
...
```

This component has no selectable data element clusters, and therefore the standard generator statements do not include an OPTION statement.

Assume that there is a valid NPA Component cluster ID called NET1, and its cluster number is 47. The OPTION statement might be included in the NPA definition member, and data elements belonging to the cluster would have a cluster code of 47 coded in their NAME statements, as follows:

```
   *
   * NPA Component generator statements
   *
   GEN GENFILES
-->OPTION NONET1
   COMP NPA 050 32000 VBS NOACCT NPA Component
   TYPE XR  3 . 3 . 3 .
   NAME NPANRRLS  Receiving Line Speed (line/link)
   NAME NPANRSLS  Sending Line Speed (line/link)
   TYPE XA  5 TIME11.2  6 TIME15.2  6 TIME15.2
   NAME NPATMAOT  Actual Observation Time
   TYPE XR  4 TIME8.  4 TIME8.  4 TIME8.
   NAME NPATSEIS  Expected Interval Time
   NAME NPATSFRU  Forward RU Send Time
   NAME NPATSLOG  Time Record Logged To SMF
   AREA NPA NPA Activity Information Area
   FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
   TYPE R 3 . 3 . 3 .
   NAME NCPNRFBQ  00  0 0 0 0 0 Free Buffer Queue Length
-->NAME NCPNRHQL  47  0 0 0 0 0 NCP Channel Hold Queue Length
-->NAME NCPNRIQL  47  0 0 0 0 0 NCP Channel Intrmd Queue Len.
   NAME NCPNRSFB  00  0 0 0 0 0 Free Buffer Count At Slowdown
   ...
```

To activate the entire cluster, change the cluster keyword NONET1 to NET1. All data elements that belong to the cluster (including NCPNRHQL and NCPNRIQL) would be added to all timespans of all files for which they are defined in the NPA Component:

```
   *
   * NPA Component generator statements
   *
   GEN GENFILES
-->OPTION NET1
   COMP NPA 050 32000 VBS NOACCT NPA Component
   ...
```

See the individual product guides for a list of cluster names used in a specific product's cccGENIN member.

```
*********************************
* TYPE Statement Group Additions *
*********************************
```

The example file definition contains the component generator definition statements:

```
    ...
    TYPE A 5 TIME11.2 6 TIME15.2 6 TIME15.2
    NAME NCPTMFCT  00  0 0 0 0 0 Free Cycle Time
```

Assume that your NPA Component does not have a data element for percentage of time the 3705 was busy in the measurement interval. Such a data element could be calculated from the duration of the measurement interval and the free cycle time measured in the interval.

Adding this data element to the file can be done by adding a TYPE group of MCG statements to the file's existing statements in cccGENIN. The TYPE statement itself can be configured from scratch or can be patterned after a TYPE statement for a similar data element in this or another cccGENIN member. For the percent data element, use the following:

```
TYPE C 3 MAPCT.   3 MAPCT.   3 MAPCT.
     - ---------- ---------- ----------
     |     |          |          |
     |     |          |          +- MONTHS/YEARS length/fmt
     |     |          +------------ DAYS/WEEKS length/format
     |     +------------------------ DETAIL length/format
     +------------------------------ type: computed
```

Code the NAME statement as follows:

```
TYPE C 3 MAPCT.   3 MAPCT.   3 MAPCT.
NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
     -------- -- --------- -----------------
        |      |     |            |
        |      |     |            +- data element label
        |      |     +-------------- active in all timespans
        |      +-------------------- no associated cluster
        +--------------------------- name of data element;
                                      begins with file name
                                      (NCP)
```

Assume that such user-added data elements are to be documented in the CA MICS Data Dictionary. In fact, all such user-added NPA percentage data elements can be documented in the same member of the Data Dictionary. Such a Data Dictionary member may be called USRPCNPA. To gain access to that member through the Document Access online browse function, add an ALIAS statement to the type group,

```
TYPE C 3 MAPCT.   3 MAPCT.   3 MAPCT.
ALIAS USRPCNPA
NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
```

(This feature is used in CA MICS for closely-related data elements, such as a set of counters. Thus a set of counter data elements COUNT01-COUNT08 could all be described in a Data Dictionary entry called COUNTXX.)

Since this is a calculated data element, we also need to add statements that describe the calculation necessary to put a value in NCPPCBSY. The value should be

$$\text{percent busy} = \frac{\text{busy time}}{\text{interval duration}}$$

where busy time is the interval duration minus the free time. Thus, the type group has the expression added:

```
TYPE C 3 MAPCT.   3 MAPCT.   3 MAPCT.
ALIAS USRPCNPA
NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
EXP 01 IF NPATMAOT GT 0 THEN
EXP 02  NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
EXP 03 ELSE NCPPCBSY = 0;
```

Also, the calculation of this data element's value depends on other data elements existing in the timespans in which NCPPCBSY is to be active. So, a DEPEND statement has to be added:

```
TYPE C 3 MAPCT.   3 MAPCT.   3 MAPCT.
ALIAS USRPCNPA
NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
EXP 01 IF NPATMAOT GT 0 THEN
EXP 02  NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
EXP 03 ELSE NCPPCBSY = 0;
DEPEND NCPTMFCT NPATMAOT
```

The definition of the data element in the MCG Component definition statements is complete. These statements are to be added at a TYPE group break, such as:

```
   ...
   FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
-->TYPE C 3 PERCENT. 3 PERCENT. 3 PERCENT.
-->ALIAS USRPCNPA
-->NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
-->EXP 01 IF NPATMAOT GT 0 THEN
-->EXP 02  NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
-->EXP 03 ELSE NCPPCBSY = 0;
-->DEPEND NCPTMFCT NPATMAOT
   TYPE R 3 . 3 . 3 .
   NAME NCPNRFBQ  00  0 0 0 0 0 Free Buffer Queue Length
   NAME NCPNRHQL  00  0 0 0 0 0 NCP Channel Hold Queue Length
   ...
```

However, the data element is specified to exist in the DETAIL timespan of the file. The EXP statements generate the code to recalculate the data element at each level of summarization, but no calculation is made at the DETAIL level, because the MCG does not modify the component's daily format routine.

To store a value in the data element in the DETAIL timespan, modify logic in the component's daily format routine using standard CA MICS file exits. (See the User Exit Facilities section in chapter 4 for more information.)

For this NPA example, the file exit for the NPANCP File is _USRSNCP, in sharedprefix.MICS.SOURCE(#NPAEXIT). The calculation must be added to the stub exit macro that currently exists. The current macro is coded as

```
    MACRO _USRSNCP   %
```

and should be modified to

```
    MACRO _USRSNCP
      IF NPATMAOT GT 0 THEN
       NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
      ELSE NCPPCBSY = 0;
    %
```

This code is executed immediately before the format
routine writes a DETAIL observation of the NPANCP File and
the data element value is set when the OUTPUT occurs.

```
**************************
* Making the Modifications *
**************************
```

To add a data element to one or more timespans if the
data element is not a sequence data element in the file,
perform the following:

- Modify the option selection codes on the OPTION
  statement, if necessary.

- Code the MCG TYPE group statements for the data element
  to be added, if necessary.

- Add initial calculations for the data element's DETAIL
  timespan value to the standard file exit in the
  component's standard exit member on the CA MICS source
  library.

- Run the component generation for the component in
  question.

- Add the data element to the file in the desired
  timespan(s) since the next daily update for any
  database unit in which the component has been installed
  reflects the change.

To add a data element to one or more timespans if the data
element IS a sequence data element in the file, see the
sections in this chapter on modifying the sequence of a file.
Modify the sequence, and follow the instructions above to add
the data element.

6.2.1.1.3 Changing Existing Data Element Definitions

Changing the attributes of a data element involves making
modifications to the CA MICS Component Generator (MCG) type
group that includes the data element.

According to the definition in the MCG section of this document, a type group is made up of a TYPE statement, followed by one or more NAME statements. Each NAME statement defines a data element. The NAME statement may be preceded optionally by an ALIAS statement, and may be followed by one or more EXP statements (if it is a computed data element) and one or more DEPEND statements (if the data element depends on other data elements in the file in the same timespans).

The example constructed in the previous section was a TYPE group that added a data element called NCPPCBSY to the NPANCP file. This discussion will refer to that data element for an example.

```
   *
   * NPA component generator statements
   *
   GEN GENFILES
   COMP NPA 050 32000 VBS NOACCT NPA Component
   AREA NPA NPA Activity Information Area
   FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
-->TYPE C 3 PERCENT. 3 PERCENT. 3 PERCENT.
-->ALIAS USRPCNPA
-->NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
-->EXP 01 IF NPATMAOT GT 0 THEN
-->EXP 02  NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
-->EXP 03 ELSE NCPPCBSY = 0;
-->DEPEND NCPTMFCT NPATMAOT
   TYPE R 3 . 3 . 3 .
   NAME NCPNRFBQ  00  0 0 0 0 0 Free Buffer Queue Length
   NAME NCPNRHQL  00  0 0 0 0 0 NCP Channel Hold Queue Length
   ...
```

```
*********************************
* TYPE Statement Modifications *
*********************************
```

Changing the TYPE statement of the type group to which the data element belongs will change the type code, length, or format specification for the data element.

Changing the length or format can be done by changing the parameters on the TYPE statement.

```
TYPE C 4 Z3.2 . . . .
```

```
is a valid length/format specification.  See the MCG section
of this document on the TYPE statement.

However, changing the type code may require more thought.  In
the example, having a type code of "C" requires EXP
statements to be included after the data element.  Changing
the type code to other than calculated, such as "A" for
accumulated, would require removal of the EXP statements.

*********************************
* ALIAS Statement Modifications *
*********************************

If the name of the data dictionary member that documents the
data element ever changes, modify or remove the ALIAS
statement.  The data element in the example was documented in
the member USRPCNPA, and the statement was:

ALIAS USRPCNPA

If the documentation for all user data elements were moved to
a single member, such as one called USRMICS, the statement
would be changed to:

ALIAS USRMICS

If the documentation for this data elements was moved to a
member of its own, and named identically to the data element
(NCPPCBSY), then the ALIAS statement can be removed.

*********************************
* NAME Statement Modifications *
*********************************

The previous two sections of this document showed how to use
the cluster and timespan indicators on the NAME statement to
modify a data element's location on the file and timespans.
Changing the data element tag (short name) actually
constitutes dropping one data element and adding another.
This is because all identification of the data element is
done through the NAME statement tag parameter.
```

The only element left on the NAME statement to modify is the data element label. This parameter on the statement can be modified in free form, according to the rules of coding the NAME statement. If coded in lower case letters with initial capitals, the label will be translated to upper case by the MCG where necessary. For example,

```
NAME NCPPCBSY 00 0 0 0 0 0 Percent of 3705 Busy
                           --------------------
                                new label
```

```
******************************
* EXP Statement Modifications *
******************************
```

EXP statements are only present if the type code from the TYPE statement is C or XC. These statements are used in recalculating the data element's value in summarization.

Let us assume the original calculation for the example were incomplete, such as:

```
TYPE C 3 PERCENT. 3 PERCENT. 3 PERCENT.
ALIAS USRPCNPA
NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
EXP 01 NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
DEPEND NCPTMFCT NPATMAOT
```

This expression could produce a zero-divide condition and SAS termination if the value of NPATMAOT were ever zero. Changing the calculation to take this into account is a matter of rearranging the expression, as

```
   TYPE C 3 PERCENT. 3 PERCENT. 3 PERCENT.
   ALIAS USRPCNPA
   NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
-->EXP 01 IF NPATMAOT GT 0 THEN
-->EXP 02  NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
-->EXP 03 ELSE NCPPCBSY = 0;
   DEPEND NCPTMFCT NPATMAOT
```

Note that computed data elements of the type code CN and XCN do not have EXP statements following their name statements. Rather, calculations for these data elements are usually performed as defined by EXP statements that follow two special NAME statements, @@FIRST or @@LAST. These NAME statements do not define data elements that reside on the file. Their only purpose is to enter calculations that must be performed before (or after, respectively) normally-defined EXP calculations are performed.

Data elements of these types can have their expressions changed by modifying the EXP statements that calculate the data elements' value. These data elements are located in the @@FIRST or @@LAST group.

```
**********************************
* DEPEND Statement Modifications *
**********************************
```

Depend statements define the relationships between the data element in question and other data elements. Reasons for modifying DEPEND statements are rare, but the DEPEND statement may contain any combination of one or more data element names. Each data element named in a DEPEND statement must exist on the file in all the same timespans as the data element in question.

Our original example included

```
DEPEND NCPTMFCT NPATMAOT
```

to insure that the NCP file could not be generated if NCPPCBSY was in a timespan without either NCPTMFCT or NPATMAOT. Either case would define an invalid construction of the summarization for NCPPCBSY. If some other data element were identified as being required for the calculation of the data element, it would be added to the DEPEND list for that element.

```
For example, if you want NCPPCIDL, the percent of 3705 idle
time, to be on the file in every timespan as NCPPCBSY, code:

   TYPE C 3 PERCENT. 3 PERCENT. 3 PERCENT.
   ALIAS USRPCNPA
   NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
   EXP 01 IF NPATMAOT GT 0 THEN
   EXP 02  NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
   EXP 03 ELSE NCPPCBSY = 0;
-->DEPEND NCPTMFCT NPATMAOT NCPPCIDL


****************************
* Making the Modifications *
****************************

To change the data element's description in the above manner,
perform the following:

    - Modify the MCG control statements explained above for
      each data element to be modified.

    - Run the component generation for the component in
      question.

    - The next daily update for any database unit in which
      the component has been installed will reflect the
      change, and add the data element to the file in the
      desired timespan(s).

A Change to the definition of a data element will not
necessarily be reflected accurately in timespans greater
than DAYS.  Accumulating data to the WEEKS, MONTHS, and YEARS
timespans takes some time to complete.  Also, the
accumulation of to-date data in these timespans may result
in combining unlike data.  Changing data elements from
accumulated to maximum or minimum, for example, will always
produce spurious results in these timespans.  This problem
is resolved by itself only after the modification has been in
effect long enough that unlike data is not being combined.

It is therefore necessary to examine the need for
retrofitting existing data in such situations.  If the need
for a retrofit exists, see the section of this document on
database maintenance techniques.
```

6.2.1.2 Common Data Elements

The example file definition contains the component generator definition statements:

```
...
TYPE N 3 . 3 . 3 .
NAME NCPMNFBL  00  0 0 0 0 0 Free Buffer Low Water Mark
NAME NPANRRLS  99  0 0 0 0 0 .
NAME NPANRSLS  99  0 0 0 0 0 .
NAME NPATMAOT  99  0 0 0 0 0 .
...
```

Notice the data elements whose option code is 99. These data elements are NPANRRLS, NPANRSLS and NPATMAOT. They are also contained in the generation library member $DEGENIN:

```
...
TYPE XR  3  .  3  .  3  .
NAME NPANRRLS Receiving Line Speed (line/link)
TYPE XR  3  .  3  .  3  .
NAME NPANRSLS Sending Line Speed (line/link)
TYPE XA  5  TIME11.2  6  TIME15.2  6  TIME15.2
NAME NPATMAOT Actual Observation Time
...
```

The reason for having data elements defined in this common manner is to assure consistency of definition of such common data elements across all files in which they appear. Data elements that are not of this "common" nature are called "standard" data elements.

Most common data elements are named generically, according to their function or contents. Examples of such data elements are ENDTS (end time stamp) and INTERVLS (the number of recording intervals in a measurement summarization).

Some common data elements take their names from the component. The first three characters of the data element name are the component identifier, and the remaining 5 characters are a cryptic abbreviation of the data element's function. Examples are NPANRRLS (NPA component receiving line speed) and NPATMAOT (NPA component actual interval observation time).

It is possible to redefine common data elements into standard data elements. For example, the above extract could have been coded:

```
...
TYPE N 3 . 3 . 3 .
NAME NCPMNFBL  00  0 0 0 0 0 Free Buffer Low Water Mark
TYPE XR 3 . 3 . 3 .
NAME NPANRRLS  00  0 0 0 0 0 Receiving Line Speed (line/link)
NAME NPANRSLS  00  0 0 0 0 0 Sending Line Speed (line/link)
TYPE XA  5  TIME11.2  6  TIME15.2  6  TIME15.2
NAME NPATMAOT  00  0 0 0 0 0 Actual Observation Time
...
```

Redefining these elements in this manner defeats the main purpose of common data elements - single source format definition. Common data elements are often key data elements in the file, used in sequencing the file's observations. If you attempt to bring two files together, and both files have data elements of the same name in the key (eg. both have ENDTS in the file sequence structure), then those data elements must have the same format on both files.

6.2.1.2.1 Changing Attributes of Common Data Elements

Changing the attributes of a common data element involves making modifications to the CA MICS Component Generator (MCG) TYPE group that includes the common data element. This will be done in one of the following sharedprefix.MICS.GENLIB members:

- Elements that are common to all CA MICS files are defined in sharedprefix.MICS.GENLIB($DEGENIN).

- Elements that are common to the files in a single CA MICS product are defined in the applicable cccGENIN member, where ccc is a CA MICS product identifier.

Refer to the information in Section 4.2.4.4.1 of this guide for information about the statements in the TYPE group.

The following example will be used in discussing the considerations for changing common element attributes.

```
...
COMP NPA  050  32000  VBS  NOACCT  NPA Component
TYPE XR  3  .  3  .  3  .
NAME NPANRRLS Receiving Line Speed (line/link)
TYPE XR  3  .  3  .  3  .
NAME NPANRSLS Sending Line Speed (line/link)
TYPE XA  5  TIME11.2  6  TIME15.2  6  TIME15.2
NAME NPATMAOT Actual Observation Time
...
AREA NPA  NPA Activity Information Area
...
```

Changing the description of a common data element changes the element in all files in which it is referenced. Such changes are:

- Changing the format or length specification of the element.

- Changing the label attached to the data element.

- Changing the type code of the data element.

```
For example, the data element NPATMAOT could be changed
as follows:

      New Type                          New Length and
        Code                            Format for
      (Common                           MONTHS and
      Maximum)                          YEARS time-spans
          |                                  |
          V                                  V
    ...  --                           -----------
    TYPE XM  5  TIME11.2  6  TIME15.2  8  TIME18.2
+-->ALIAS NPATIME
|   NAME NPATMAOT Actual Measurement Time
|   ...           ----------------------
|                        New Label
|
+---hypothetical new data dictionary
    member where all NPA time data
    elements would be documented
    (for example only; no such
    convention exists or should).


**********
* CAUTION *
**********

Changes to common data elements affect all files in which
each changed data element appears.  The following items must
be considered:

    - Common data elements are often used as file sequence
      elements for the files in which they appear.  Even if
      they are not in any file's key, they may be very
      important by virtue of being used in many places or
      being used in extended, user-generated report
      processes.  These data elements should not have their
      attributes modified without the advice of the CA MICS
      Product Support Group.  Further considerations are
      discussed in Section 6.2.2.3 of this guide, which
      addresses the impact of changing a file's sequence in a
      time-span.
```

```
    - Section 6.2.1.1.3 of this guide presented
      considerations for changing existing data element
      definitions.  The information presented there also
      pertains to common elements.

    - Changing the definition of a data element will not
      necessarily be reflected immediately in time-spans
      greater than DAYS.  Accumulating data to the WEEKS,
      MONTHS, and YEARS time-spans takes some time to
      complete.  Also, the accumulation of to-date data in
      these time-spans may result in combining unlike data.
      Changing data elements from accumulated to maximum or
      minimum, for example, will always produce spurious
      results in these time spans until a new interval is
      begun in each time-span, and unlike data is no longer
      combined in the archive files.

      You should examine the need for retrofitting existing
      data in such situations.  If the need for a retrofit
      exists, refer to Section 6.3 of this guide.

****************************
* Making the Modifications *
****************************

    To change a common data element's attributes,

    - Modify the MCG control statements explained above for
      each common data element to be modified.

    - Generate the component by running the cccCGEN job for
      each affected product.

    - The next daily update for any database unit in which
      the product has been installed will reflect the change.
```

6.2.1.2.2 Adding New Common Data Elements

```
Adding a common data element involves adding a CA MICS
Component Generator (MCG) TYPE group that includes the common
data element.  Then add the common element to the files in
which you want it reported.
```

If the new common data element is local to any single component, add the TYPE group to the component's cccGENIN member in sharedprefix.MICS.GENLIB. If the new common data element is useful for more than one component, add the TYPE group to the $DEGENIN member in sharedprefix.MICS.GENLIB.

As discussed in chapter 4, a common data element TYPE group is made up of the following statements:

- a TYPE statement, mandatory
- an ALIAS statement, optional
- an ALTNAME statement, optional
- a NAME statement, mandatory

The example file definition contains the component generator definition statements:

```
...
COMP NPA  050  32000  VBS  NOACCT  NPA Component
TYPE XR  3  .  3  .  3  .
NAME NPANRRLS Receiving Line Speed (line/link)
TYPE XR  3  .  3  .  3  .
NAME NPANRSLS Sending Line Speed (line/link)
TYPE XA  5  TIME11.2  6  TIME15.2  6  TIME15.2
NAME NPATMAOT Actual Observation Time
...
AREA NPA  NPA Activity Information Area
...
```

Adding a common data element can be done by adding the TYPE group for the data element into the NPAGENIN member. For example, assume a data element called PHASE were to be added to the NPA component. PHASE is intended to contain the character strings "FULL", "1Q ", "3Q ", or "NEW ", so it should be defined as a character data element of length four. Define the data element in NPAGENIN by keying in the following prior to the first AREA statement:

```
...
TYPE XR $4. . $4. . $4. .
NAME PHASE Phase of Processing Period
...
```

Then add the new common element to each file in which you want the element reported by adding a NAME statement like the following after each FILE statement:

```
NAME PHASE 99 0 0 0 0 0 .
```

Since this is a retained data element, we also need to add statements that describe the calculation necessary to put an initial value in PHASE. The value could be set by any means. Assume that the site has a user-written routine that returns a character value for the "phase" and the argument supplied to the routine is the time stamp of the end of the measurement interval.

To store a value in the data element in the DETAIL timespan, modify logic in the component's daily format routine using standard CA MICS file exits. (See the User Exit Facilities section in chapter 4 for more information.)

For this NPA example, the file exit for the NPANCP file is _USRSNCP, in sharedprefix.MICS.SOURCE(#NPAEXIT). The calculation must be added to the stub exit macro that currently exists. The current macro is:

```
MACRO _USRSNCP   %
```

which should be modified to:

```
MACRO _USRSNCP
  PHASE = UPHASE(ENDTS);
%
```

This code will be executed immediately before the format routine writes a DETAIL observation of the NPANCP file, and the data element value will be set when the OUTPUT occurs.

Such logic would have to be executed to set a value for PHASE before the format routine OUTPUT for each file of each component in which the data element is to be carried.

****
* Making the Modifications *
****

To add a data element to one or more timespans of one or more files within one or more components, if the data element is not a sequence data element in the file, perform the following:

- Code the MCG type group statements for the data element to be added to either $DEGENIN or cccGENIN.
- Add the element to the files in which you want it reported.
- Add initial calculations for the data element's DETAIL timespan value to the standard file exit in the component's standard exit member on the CA MICS source library.
- Run the component generation for the components in question.
- The next daily update for any database unit in which the component has been installed will reflect the change, and add the data element to the files in the desired timespan(s).

To add a data element to one or more timespans, if the data element IS a sequence data element in the file, see the sections in this chapter on modifying the sequence of a file. Modify the sequence, and follow the instructions above to add the data element.

6.2.1.2.3 Assigning Alternate Data Element Names

The CA MICS Component Generator (MCG) contains a facility for changing the name of a common data element for reference purposes.

```
************************************************************
*                                                          *
*    WARNING:  Using the facility described in this        *
*    section can have major, unexpected impacts.           *
*    Very, very few installations truly need this          *
*    capability.  Please contact CA MICS support           *
*    before attempting to assign an alternate name         *
*    to any data element.                                  *
*                                                          *
*    ***WE DO NOT RECOMMEND USE OF THIS FACILITY***        *
*                                                          *
************************************************************
```

Normally, changing the name of a data element is a matter of deleting the old data element and adding a new data element. Such changes for common data elements have more impact than for standard data elements. Common data elements, such as MONTH, are used in VERY MANY places. Changing all the CA MICS logic that references such data elements would be a great effort, prone to errors.

The ALTNAME statement is an attempt to eliminate the need to change logic if it is desirable to change the name (tag) associated with a common data element.

According to the definition in the MCG section of this document, a common data element type group is made up of a TYPE statement, followed by one or more NAME statements. Each NAME statement defines a data element. The NAME statement may be preceded optionally by an ALIAS statement, and may be preceded optionally by an ALTNAME statement.

The example NPANCP file definition contains the component generator definition statements in the NPAGENIN member of sharedprefix.MICS.GENLIB:

```
...
NAME DAY       99  5 5 N N N .
NAME HOUR      99  6 6 6 N N .
NAME WEEK      99  0 0 4 N N .
NAME YEAR      99  3 3 3 3 3 .
NAME MONTH     99  4 4 N 4 N .
...
```

where the common data elements (cluster code 99) point to common data element definitions for these data elements. These definitions are contained in the generation library member $DEGENIN:

```
...
TYPE XR  2  .  2  .  2  .
NAME DAY      Day of Month
TYPE XR  2  .  2  .  2  .
NAME HOUR     Hour Of Day
TYPE XR  2  .  2  .  2  .
NAME MONTH    Month of Year
TYPE XR  2  .  2  .  2  .
NAME WEEK     Week Of Year
TYPE XR  2  .  2  .  2  .
NAME YEAR     Year of Century
...
```

```
Let us assume that there was a need to rename the data
element MONTH to another name, well recognized in your
company's current report structure.  Let us call the new name
PERIOD.

Adding the following ALTNAME statement to the $DEGENIN
member will aid in the rename of the data element.

    ...
    TYPE XR  2  .  2  .  2  .
    NAME DAY      Day of Month
    TYPE XR  2  .  2  .  2  .
    NAME HOUR     Hour Of Day
    TYPE XR  2  .  2  .  2  .
--->ALTNAME PERIOD
    NAME MONTH    Month of Year
    TYPE XR  2  .  2  .  2  .
    NAME WEEK     Week Of Year
    TYPE XR  2  .  2  .  2  .
    NAME YEAR     Year of Century
    ...

WHAT ASSIGNING AN ALTERNATE NAME DOES:

Assigning an alternate name to a common data element adds
a new data element to all files that reference the old data
element name in their file definition MCG statements in
sharedprefix.MICS.GENLIB member cccGENIN.  The old data
element name is dropped from the file.

In our example, the NPAGENIN member defines files that
contain the data element MONTH.  Specifying the ALTNAME of
PERIOD for MONTH in $DEGENIN means that MONTH will not be on
the files, but PERIOD will be there instead.

This means that any report process may reference the item
by the new name.

WHAT ASSIGNING AN ALTERNATE NAME DOES NOT DO:

This modification does not assign an initial value to the
new data element.  In the example, the value for the data
element PERIOD will have to be set by some other CA MICS
modification.  Several methods for such a modification exist.
```

If the data element exists in a large number of CA MICS files, modification of common system routines is indicated.

If the data element exists in a small number of CA MICS files, especially if the files are localized to one component, use the standard CA MICS file exits described in chapter 4 of this guide.

For this NPA example, the file exit for the NPANCP file is _USRSNCP, in sharedprefix.MICS.SOURCE(#NPAEXIT). The calculation must be added to the stub exit macro that currently exists. The current macro is:

```
MACRO _USRSNCP   %
```

which should be modified to:

```
MACRO _USRSNCP
  PERIOD = MONTH;
%
```

Such a calculation is possible because renaming the data element does not drop the original data element from standard CA MICS calculations. In fact, all existing CA MICS logic will reference the data element name by the old name. This allows a flexibility in the file structure without massive CA MICS logic changes.

This code will be executed immediately before the format routine writes a DETAIL observation of the NPANCP file, and the data element value will be set when the OUTPUT occurs.

Such logic would have to be executed to set a value for PERIOD before the format routine OUTPUT for each file of each component in which the data element is to be carried.

****
* Making the Modifications *
****

To assign an alternate name to a common data element, perform the following:

- Code the MCG ALTNAME statements for the data element to be added to $DEGENIN.

  Note: the cccGENIN members do not need to be changed.

- Add initial calculations for the data element's DETAIL timespan value to the standard file exit in the component's standard exit member on the CA MICS source library, if the change is local to a component.

  -or-

  Add initial calculations for the new data element to a common data element determination routine, in the recommended method for such modifications.

- Run the component generation for the components in question.

- The next daily update for any database unit in which the component has been installed will reflect the change, and add the data element to the files in the desired timespan(s).

To assign an alternate name to a data element, if the data element IS a sequence data element in the file, see the sections in this chapter on modifying the sequence of a file. Modify the sequence, and follow the instructions above to assign the alternate name.

6.2.2 Tailoring Database Files

The middle level of tailoring involves changing database file definitions describing online storage. (Offline storage is discussed in section 6.2.3.)

CA MICS database files can exist in any of the following online timespans: DETAIL, DAYS, WEEKS, MONTHS, and YEARS. You control the amount of data that is kept in each timespan (the number of cycles retained online). You also control whether or not the data is compressed.

For example, the NPANCP file might be defined as follows:

DETAIL	DAYS	WEEKS	MONTHS	YEARS
NPANCP01	NPANCP01	NPANCP01	NPANCP01	does not
	NPANCP02	NPANCP02	NPANCP02	exist
	...	...	...	
	NPANCP10	NPANCP53	NPANCP06	

Defining cycles and compression is done in two places.

The CA MICS Component Generator's FILE statement defines the timespans in which the file exists. For the NPA component's NPANCP file, sharedprefix.MICS.GENLIB(cccGENIN) might contain the following statement:

```
FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
              | | | | |
              | | | | +- does not exist in YEARS
              | | | +--- exists in MONTHS
              | | +----- exists in WEEKS
              | +------- exists in DAYS
              +--------- exists in DETAIL
```

The number of cycles that CA MICS will age within a timespan is specified in the DBMODEL member of the unit-level CA MICS parameter library. (Optionally, a CYCLES statement can be coded after the FILE statement in the GENLIB library to perform the same function.) The number of cycles does not override the file's active/inactive status as defined by the FILE statement. (If the file is not active but cycle retention is greater than zero, the cycles are NOT retained.)

The ability to compress a file's data in a timespan is specified in the COMPRESS member of the unit-level parameter library. If you code a COMPRESS statement in the GENLIB library, the COMPRESS parameter member will be created by first COPYLIBU job executed after the COMPRESS statement is added.

For the NPANCP file, the cycle definitions in prefix.MICS.PARMS(DBMODEL) contain a FILE statement for the NPANCP file that states the number of cycles to be aged in each timespan:

```
FILE  NPANCP 01 10 53 06 00 000 053 024
             |  |  |  |  |
             |  |  |  |  +- does not exist in YEARS
             |  |  |  +---- 6 cycles in MONTHS
             |  |  +------- 53 cycles in WEEKS
             |  +---------- 10 cycles in DAYS
             +------------- 1 cycle in DETAIL
```

To tailor the amount of space used by an online database, you can:

- tailor the file cycle definitions in prefix.MICS.PARMS(DBMODEL) or the CYCLES statement in sharedprefix.MICS.GENLIB(NPAGENIN) to adjust the amount of data retained

- tailor the file's data compression status using the definitions in prefix.MICS.PARMS(COMPRESS) and on NPAGENIN's COMPRESS statement in sharedprefix.MICS.GENLIB

To carry a week's worth of DETAIL data instead of only one day, change the FILE statement in DBMODEL to:

```
FILE  NPANCP 07 10 53 06 00 000 053 024
            --
             |
             +-- changed parameter
```

Because the NPANCP file is inactive in the YEARS timespan (the FILE statement in sharedprefix.MICS.GENLIB(NPAGENIN) has N coded for the YEARS timespan definition), the YEARS timespan cycle definition on the NPANCP file's FILE statement in prefix.MICS.PARMS(DBMODEL) must be 00. You cannot activate the file in the YEARS timespan by changing the cycle definition to a number greater than zero.

Likewise, the WEEKS timespan cannot be deactivated by changing the cycle definition from 53 to 00. CYCLEGEN execution errors will occur in either case.

To compress the NPANCP file's online data, first ensure that the COMPRESS statement in sharedprefix.MICS.GENLIB member NPAGENIN looks like the following:

```
COMPRESS  Y  Y  Y  Y  Y
```

Then ensure that the prefix.MICS.PARMS(COMPRESS) member for the unit in which the NPA component is installed contains, at a minimum, the following:

```
DBCOMPRESS ON NPA
IFCOMPRESS ...
FILE NPA NPANCP Y Y Y Y Y
```

6.2.2.1 Delete File From a Timespan

To delete a file from a timespan in which it has been defined to exist, the following must be done:

- Change the FILE statement in the component definition in the component's sharedprefix.MICS.GENLIB member to reflect the fact. The "Y" in that timespan's position must be changed to an "N".

- Run the MCG to regenerate the component from the modified component generator statements. This job will change the component's executable code in the sharedprefix.MICS.SOURCE library, and all database units in which the component is installed will be affected.

- Run the COPYLIBU job for each unit that the component is installed in.

- Run the CYCLEGEN job for all database units in which the component is installed. This will generate aging statements that reflect the new file configuration.

6.2.2.2 Add File in a Timespan

To add a file to a timespan, do the following:

- Review the key structure of the file in the component's sharedprefix.MICS.GENLIB member to insure that enabling the file in the timespan will produce a properly-keyed file.

- Change the FILE statement in the component definition in the component's sharedprefix.MICS.GENLIB member to reflect the fact. The "N" in that timespan's position must be changed to a "Y".

- Run the cccCGEN to regenerate the component from the modified component generator statements. This job will change the component's executable code in the sharedprefix.MICS.SOURCE library, and all database units in which the component is installed will be affected.

- Run the COPYLIBU job for each unit that the component is installed in.

- Verify that the number of cycles to be retained in each FILE statement in prefix.MICS.PARMS(DBMODEL) suits your installation's standards.

- Run the CYCLEGEN job for all database units in which the component is installed. This will generate aging statements that reflect the new file configuration.

6.2.2.3 Change a File's Sequence in a Timespan

The most complicated level of file tailoring involves changing data element statements to modify the sequence of observations within a file.

The statements listed below define an example Network Performance Analyzer (NPA) component with one file, the Network Control Program File. These statements would be found in the components cccGENIN member in sharedprefix.MICS.GENLIB. Data element definition statements that define the sequence of file observations in any timespan are indicated by "-->" in the left margin of the example.

```
   *
   * NPA component generator statements
   *
   GEN GENFILES
   COMP NPA 050 32000 VBS NOACCT NPA Component
   AREA NPA NPA Activity Information Area
   FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
   TYPE R 3 . 3 . 3 .
   NAME NCPNRFBQ  00  0 0 0 0 0 Free Buffer Queue Length
   NAME NCPNRHQL  00  0 0 0 0 0 NCP Channel Hold Queue Length
   NAME NCPNRIQL  00  0 0 0 0 0 NCP Channel Intrmd Queue Len.
   NAME NCPNRSFB  00  0 0 0 0 0 Free Buffer Count At Slowdown
   TYPE A 5 TIME11.2 6 TIME15.2 6 TIME15.2
   NAME NCPTMFCT  00  0 0 0 0 0 Free Cycle Time
   NAME NCPTMTIS  00  0 0 0 0 0 Time In Slowdown
   TYPE M 3 . 3 . 3 .
   NAME NCPMXANP  00  0 0 0 0 0 Max Available NCP Buffers
   NAME NCPMXFBH  00  0 0 0 0 0 Free Buffer High Water Mark
   TYPE N 3 . 3 . 3 .
   NAME NCPMNFBL  00  0 0 0 0 0 Free Buffer Low Water Mark
   NAME NPANRRLS  99  0 0 0 0 0 .
   NAME NPANRSLS  99  0 0 0 0 0 .
   NAME NPATMAOT  99  0 0 0 0 0 .
   NAME NPATSEIS  99  0 0 0 0 0 .
   NAME NPATSFRU  99  0 0 0 0 0 .
   NAME NPATSLOG  99  0 0 0 0 0 .
   NAME RESNAME   99  0 0 0 0 0 .
-->NAME DAY       99  5 5 N N N .
-->NAME HOUR      99  6 6 6 N N .
-->NAME WEEK      99  0 0 4 N N .
-->NAME YEAR      99  3 3 3 3 3 .
-->NAME MONTH     99  4 4 N 4 N .
   NAME INTERVLS  99  0 0 0 0 0 .
   NAME DURATION  99  0 0 0 0 0 .
```

```
   NAME STARTTS   99  0 0 0 0 0 .
-->NAME ENDTS     99  7 0 0 0 0 .
-->NAME ZONE      99  0 0 5 5 4 .
   NAME DAYNAME   99  0 0 N N N .
-->NAME SYSID     99  1 1 1 1 1 .
-->NAME NCPNAME   99  2 2 2 2 2 .
   NAME NETNAME   99  0 0 0 0 0 .
```

Note the timespan indicators. For data elements that merely exist on the file in all timespans, the indicators are of the form:

```
   NAME NETNAME   99  0 0 0 0 0 .
                      ---------
                      indicators
```

Data elements that are part of the key to the file in any timespan have timespan indicators that are numbers other than zero in one or more timespans. Consider ZONE:

```
NAME ZONE 99  0 0 5 5 4 .
              | | | | |
              | | | | +- 4th item in key to YEARS
              | | | +--- 5th item in key to MONTHS
              | | +----- 5th item in key to WEEKS
              | +------- exists, not in key to DAYS
              +--------- exists, not in key to DETAIL
```

The NAME statements which define sequence data elements determine how observations are arranged in the timespans shown. This table gives the data elements used as keys to the sort operations that order the file:

DETAIL	DAYS	WEEKS	MONTHS	YEARS
SYSID	SYSID	SYSID	SYSID	SYSID
NCPNAME	NCPNAME	NCPNAME	NCPNAME	NCPNAME
YEAR	YEAR	YEAR	YEAR	YEAR
MONTH	MONTH	WEEK	MONTH	ZONE
DAY	DAY	ZONE	ZONE	
HOUR	HOUR	HOUR		
ENDTS				

Note that this file does not exist in the YEARS timespan, according to the FILE statement. If it did, the order would be as shown, due to coding in the sequence numbers in the timespan indicators of the NAME statements for the key data elements.

```
$SCOL
 Suppose the desired order included another data element.
 In this example, let us add NETNAME to the order.  We will
 add NETNAME immediately after NCPNAME in all timespans, to
 yield:

     DETAIL   DAYS     WEEKS    MONTHS   YEARS
     -------  -------  -------  -------  -------
     SYSID    SYSID    SYSID    SYSID    SYSID
     NCPNAME  NCPNAME  NCPNAME  NCPNAME  NCPNAME
     NETNAME  NETNAME  NETNAME  NETNAME  NETNAME
     YEAR     YEAR     YEAR     YEAR     YEAR
     MONTH    MONTH    WEEK     MONTH    ZONE
     DAY      DAY      ZONE     ZONE
     HOUR     HOUR     HOUR
     ENDTS

 Such an order can be constructed by coding the timespan
 indicators as:

    NAME RESNAME   99  0 0 0 0 0 .
    NAME DAY       99  6 6 N N N .<--+
    NAME HOUR      99  7 7 7 N N .<--+
    NAME WEEK      99  0 0 5 N N .<--+
    NAME YEAR      99  4 4 4 4 4 .<--+
    NAME MONTH     99  5 5 N 5 N .<--+- data elements modified
    NAME INTERVLS  99  0 0 0 0 0 .   |
    NAME DURATION  99  0 0 0 0 0 .   |
    NAME STARTTS   99  0 0 0 0 0 .   |
    NAME ENDTS     99  8 0 0 0 0 .<--+
    NAME ZONE      99  0 0 6 6 5 .<--+
    NAME DAYNAME   99  0 0 N N N .
    NAME SYSID     99  1 1 1 1 1 .
    NAME NCPNAME   99  2 2 2 2 2 .
+->NAME NETNAME   99  3 3 3 3 3 .
|
+- data element added to sequence


 On the surface, this is a very simple change.  It is, for
 the new user who has been fortunate enough to make such a
 modification before the initial installation of CA MICS.

 For users that have installed CA MICS and wish to preserve
 the existing data, problems arise.

 The first problem is that timespan cycles already closed
 have been sequenced in an order different from the new key.
 To merge yesterday's data with today's data, for example,
```

```
would require special processing.  Such problems occur in the
DETAIL and DAYS timespans of the online CA MICS database,
and with the Audit Archive files associated, if active.

More importantly, existing to-date files in WEEKS,
MONTHS, and YEARS timespans have already been sequenced in
the previous order.  Data following the sequence change
cannot be merged with month-to-date data, for example,
because the keys are unlike.  SAS will terminate the daily
run to update the component that contains such a file if the
to-date files have not been retrofitted with the new key
structure.

This problem is almost identical to the problem of adding or
reconfiguring account codes. See Section 6.3, "Retrofitting
Guidelines," for information on how to retrofit a file to
accommodate a modified file sequence.


****************************
* Making the Modifications *
****************************

To modify the sequence of a file in one or more timespans,
perform the following:

    - Modify the MCG NAME statements for the data elements
      needed to modify the file's sequence.

    - If such a retrofit is needed, follow the instructions
      in Section 6.3, "Retrofitting Guidelines," of this
      document.

    - Run the component generation for the component in
      question.

    - The next daily update for any database unit in which
      the component has been installed will reflect the
      change, and add the data element to the file in the
      desired timespan(s).
```

6.2.2.4 Change a File's Compression Eligibility

To change a file's compression eligibility, do the following:

o Review the COMPRESS statement specification that follows
  the FILE statement in sharedprefix.MICS.GENLIB(cccGENIN).

  If it contains Ns for all timespans and the comment
  "(FILE COMPRESSION NOT SUPPORTED)", do NOT change this
  specification and do not try to compress this file.

  The COMPRESS parameter is generated into
  sharedprefix.MICS.GENLIB(CMPRSDEF) by the cccCGEN job.
  Each database unit's COPYLIBU job adjusts the unit's
  prefix.MICS.PARMS(COMPRESS) for new products. Once the
  COMPRESS member FILE statement exists, COPYLIBU will not
  update it.

o To help you achieve your space savings goal, use the
  CA MICS Administrator Utilities.

  - Run the Candidacy report (MWF;5;4;1).

  - Review the report and choose a compression threshold.

  - Run the Recommendation report (MWF;5;4;2).

  - Review the results. The tabular report shows the effects
    of compression at the threshold you chose. The
    recommended statements for your COMPRESS member in
    prefix.MICS.PARMS are also provided, as is information
    about current and projected database size.

o Review and edit the statements in the COMPRESS parameter
  library (prefix.MICS.PARMS):

  Note: only do so if the COMPRESS statement in
         sharedprefix.MICS.GENLIB(cccGENIN) indicates that
         the file can be compressed (that is, the comment
         "FILE COMPRESSION NOT SUPPORTED" is not present on
         the COMPRESS statement).

- Edit the DBCOMPRESS statement to enable the file to be compressed by setting DBCOMPRESS ON ALL or DBCOMPRESS ON ccc for the product whose file you want compressed.

- If the file you want to compress is an intermediate file (its FILE statement in sharedprefix.MICS.GENLIB has a file type of 50, 51, or 52), edit the IFCOMPRESS statement to enable the intermediate file to be compressed by setting IFCOMPRESS ON ccc.

Note: only do so if the COMPRESS statement in sharedprefix.MICS.GENLIB(cccGENIN) indicates that the file can be compressed (that is, the comment "FILE COMPRESSION NOT SUPPORTED" is not present on the COMPRESS statement).

- Edit the FILE statement to enable compression in the timespans you desire by setting the timespan indicators to Y (enabling compression) or N (disabling compression).

  Remember that changing a specification from Y to N means that your online databases for that timespan will require more DASD. Ensure that there is adequate space in that timespan's data set to accommodate the growth to avoid B37 and D37 abends in the operational jobs.

  Do NOT add FILE statements to the COMPRESS parameter library member. If you add COMPRESS statements to cccGENIN members, run COPYLIBU to insert new FILE statements in the COMPRESS parameter library member.

o Run the CYCLEGEN job for the unit to activate your new specifications. CYCLEGEN resides in the prefix.MICS.CNTL data set.

o Data compression starts with the 01 cycle of the next DAILY job.

o To fully achieve space savings, compress the old cycles following these steps:

- Back up your current CA MICS unit libraries using the Operational Status and Tracking BACKUP command or by executing the batch BACKUP job.

- Delete your current CA MICS unit libraries and reallocate them using the ALLOCATE job and specifying the new database's size from information provided on the Recommendations tabular report.

  Factor in room for growth to prevent out-of-space abends in the future.

- Restore your new specifications to your reallocated unit libraries using the Operational Status and Tracking RESTORE command or by executing the batch RESTORE job.

6.2.3 Tailoring Archive Files

This level of tailoring involves changing database file definitions describing offline storage.

A file in the CA MICS database may be archived in three ways - audit files may be created, weekly history files may be created, and monthly history files may be created. A sample pattern, using the example of the NPANCP file, might be:

Audit Files	Weekly History	Monthly History
no audit file is produced	53 weeks of weekly history is kept	24 months of monthly history is kept

It is necessary here to review the purpose of each offline archive mechanism.

- Audit archives exist to give the user the ability to recall detail data for long periods. The audit files cause DETAIL (or DAYS if the DETAIL timespan is inactive) online files to be copied to tape. Ten cycles of online data (minus the data for the current week) are copied to tape by the CA MICS WEEKLY or the stand-alone AUDIT operational jobs. No old data is merged. Each week's audit file is a new generation of the generation data group (tape) data set for the file's audit data set.

Typically, large numbers of audit generation data groups are kept. The purpose of audit data is to be able to reconstruct activity in the finest detail available should a security breach or major processing anomaly be detected after the fact.

NOTE: Audit archives may be updated weekly, or can be updated more frequently (e.g., 3 times per week) if you are using the stand-alone AUDIT job option.

In this case, prefix.MICS.PARMS(EXECDEF) must be modified to include the following:

```
USERDEF AUDITCWK YES
```

The USERDEF AUDITCWK YES parameter specification overrides the standard, default audit archive processing which drops data from the current week in order to reduce processing time and conserve CA MICS tape database space. When you run AUDIT more frequently than once each week, you are almost certain to be doing so in order to spread a large amount of audit archive data over two or more audit cycles, and by specifying

```
USERDEF AUDITCWK YES
```

you ensure that CA MICS will process the audit archive data according to your requirements.

- Weekly histories exist to give the user the ability to produce reports at the weeks granularity for long periods. Such analyses may be done for capacity planning or performance analysis purposes. Previous histories are merged, giving, every week, a new history of the duration indicated.

- Monthly histories exist to give the user the ability to produce reports at the months granularity for even longer periods. Such analyses may be done for capacity planning or performance analysis purposes. Previous histories are merged, giving, every month, a new history of the duration indicated.

The definition of what files exist offline and how many periods in each is done in three places.

- Archiving active or inactive: The archive group active or inactive configuration applies to all database units in the CA MICS database complex where the component is installed.

- Number of history cycles to keep and audit archive override: The cutoff limit for the number of cycles of weekly or monthly history data to keep is local to the database unit. It can be zero to deactivate history archive processing. Audit archive can be deactivated with the AUDIT(NO) parameter. Each database unit in which the component is installed may have a different specification.

- Number of generations: The number of generation data groups for any archive group is local to the database unit. Each database unit in which the component is installed may have a different specification.

Whether a file exists or not in an archive process, is determined by a combination of the FILE statement of the CA MICS Component Generator (MCG) description for the component, and the FILE statement in prefix.MICS.PARMS(DBMODEL) specifying cutoff limits and activation parameters. A file marked eligible for archiving in the MCG description can be deactivated in DBMODEL. To activate a file for archiving, it must be marked eligible in the MCG description and have a cutoff limit greater than zero or the AUDIT keyword defined in DBMODEL. This specification is used to create executable SAS code that produces the archives and the prototype library members that contain JCL for the archive files to be included (or not) in the WEEKLY, MONTHLY, AUDIT, HISTW, or HISTM operational jobs.

For the example NPA component, assume that sharedprefix.MICS.GENLIB(cccGENIN) contains a statement that describes the NPANCP file. It is:

```
FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
                          | | |
                          | | +- monthly history active
                          | +--- weekly history active
                          +----- audit inactive
```

The number of periods of a file that CA MICS will contain within a history IN WHICH THE FILE EXISTS is specified in the unit-level prefix.MICS.PARMS library. This specification is used to produce macros that limit the amount of offline history data saved. For NPA, the period definitions in prefix.MICS.PARMS(DBMODEL) (for a database unit in which NPA is generated) will contain a FILE statement for the NPANCP file, giving the number of histories to be archived in each history group:

```
FILE NPANCP 01 10 53 06 00 00 053 024
                                |   |
 53 weeks of history are kept -+   |
24 months of history are kept -----+
```

For audit archive, since no AUDIT keyword is specified, the definition from cccGENIN is taken, which is audit archive inactive.

The number of generations of archive data to be saved is specified in prefix.MICS.PARMS(JCLDEF). This specification is used to produce the IDCAMS statements necessary to maintain the required number of GDG indexes in the user's catalog for the archive files. For example, the statements in JCLDEF might be defined as:

```
AUDITGDG 53 ------ 53 generations of audit archive
HISTWGDG 7  ------ 7 generations of weekly history
HISTMGDG 2  ------ 2 generations of monthly history
```

Generation data group index levels apply to all components in a database unit. Generation data group index levels must be specified in the JCLDEF PARMS member, even if all files in all components installed in the database unit have the associated archive function deactivated.

It is possible to tailor the space used by an offline database through the file retention definitions in the DBMODEL member of the PARMS library. To carry 60 weeks worth of weekly history instead of 53 weeks, change the FILE statement in DBMODEL to:

```
    FILE  NPANCP 01 10 53 06 00 00 060 024
                                    ---
                                     |
                                     +-- changed parameter
```

However, having the file inactive in the weekly history by the FILE statement in sharedprefix.MICS.GENLIB(cccGENIN) would force the weekly retention to zero.

6.2.3.1 Add File to Audit Archive Tape

To add a file to the audit archive tape, which is generated by the CA MICS WEEKLY or the standalone AUDIT operational job, do the following:

1. Edit the FILE statement in sharedprefix.MICS.GENLIB(cccGENIN). Change the "N" in that file's audit archive position to "Y".

   Refer to Section 6.2.3 for more information on the GENIN FILES statement.

2. Submit the sharedprefix.MICS.CNTL(cccCGEN) job to regenerate the component.

3. Edit prefix.MICS.PARMS(DBMODEL) for all database units in which the component is installed.

   On the FILE statement for the file to be added, either omit an AUDIT keyword, in which case the definition from cccGENIN is taken, or define AUDIT(YES). Audit archive is created from the DETAIL time-span. If no DETAIL time-span file cycles exist, it is created from the DAYS time-span. To specifically create audit archive from the DAYS time-span, define AUDIT(DAYS). Define AUDIT(NO) in units where you do not want to create this file.

   Refer to Section 2.3.4.1.2 in the PIOM for more information on the FILE statement in DBMODEL.

4. Submit the prefix.MICS.CNTL(CYCLEGEN) job for all database units in which the component is installed.

   This job changes the component's executable code in the prefix.MICS.USER.SOURCE libraries and affects only the database units in which the component is installed.

5. Edit prefix.MICS.PARMS(JCLGENU) for all database units in which the component is installed. Specify:

   ```
   WEEKLY AUDIT
   ```

6. Submit the prefix.MICS.CNTL(JCLGENU) job(s).

   This generates the WEEKLY and/or AUDIT job JCL that contains DD statements for the new audit tape configuration.

7. Submit prefix.MICS.CNTL(GDGSGEN) for all database units in which the component is installed.

   This generates audit tape file catalog indexes for the new audit file, if necessary.

For example, assume the NPANCP file example showed the file appearing in the following archives:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        YES       YES
```

Assume the desired configuration is:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  YES       YES       YES
```

The initial definition of the NPANCP FILE statement in sharedprefix.MICS.GENLIB(NPAGENIN) is:

```
    FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
                            | | |
      no WEEKLY audit-------+ | |
      WEEKLY history ---------+ |
      MONTHLY history-----------+
```

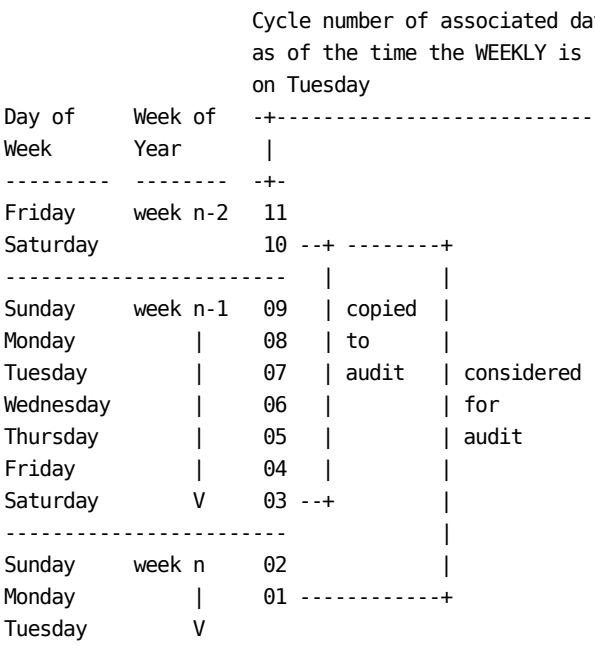
Change this statement to:

```
    FILE NCP 00 1 Y Y Y Y N Y Y Y NCP Activity File
                            |
                            +- WEEKLY audit produced
```

The number of cycles of the file that CA MICS will use as input to the weekly audit archive is fixed at 10. This means the last 10 cycles of DETAIL (or DAYS if DETAIL does not normally exist) will be copied to the new generation of the audit archive tape when the audit is generated in the WEEKLY or the standalone AUDIT operational job. If both DETAIL and DAYS are active, the DETAIL time-span files will be used. You can specifically request audit archive from the DAYS time-span with the AUDIT(DAYS) keyword on the FILE statement in prefix.MICS.PARMS(DBMODEL).

The test of what observations to put on the audit file is: any observation from a week other than the current week will be copied to the audit file. Any observation that belongs to the current week will be ignored, and will not be copied to the audit file. If less than ten cycles of the file exist in the DETAIL time-span, a data loss may occur.

The intent of using ten cycles of DETAIL data is to be able to copy the entire previous week of DETAIL data to the audit file, if the WEEKLY (or AUDIT) job was run as late as WEDNESDAY of the following week. The normal WEEKLY (or AUDIT) job is run by Tuesday. Normal cycle distribution looks like:



Data loss can occur from such conditions as the following:

- less than 9 cycles of DETAIL data are maintained

- the WEEKLY (or AUDIT) job is run as late as Thursday

- the CA MICS data element WEEK has been modified to have a value incompatible with the above structure.

- the standalone AUDIT job is executed more than once in a given week, there is less then 10 cycles, and prefix.MICS.PARMS(EXECDEF) does not contain

```
USERDEF AUDITCWK YES
```

6.2.3.2 Delete File From Audit Archive Tape

There are two ways to delete a file from the audit archive tape, which is generated by the CA MICS WEEKLY or standalone AUDIT operational job. The first only affects one database unit, the second affects all database units in which the component owning the file is installed.

To delete a file from the audit archive tape of a particular unit, without affecting other units, do the following:

1. Edit prefix.MICS.PARMS(DBMODEL).

   On the FILE statement for the file you want to delete, specify the keyword AUDIT(NO).

   Refer to Section 2.3.4.1.2 in the PIOM for more information on the FILE statement in DBMODEL.

2. Submit the prefix.MICS.CNTL(CYCLEGEN) job.

   This job changes the component's executable code in the prefix.MICS.USER.SOURCE library and affects only the database unit in which the component is installed.

3. Edit prefix.MICS.PARMS(JCLGENU). Specify:

   ```
   WEEKLY AUDIT
   ```

4. Submit the prefix.MICS.CNTL(JCLGENU) job.

   This generates the WEEKLY and/or AUDIT job JCL that contains DD statements for the new audit tape configuration.

To delete a file from the audit archive tape, affecting all database units in which the component is installed, do the following:

1. Edit the FILE statement in sharedprefix.MICS.GENLIB(cccGENIN). Change the "Y" in the file's audit archive position to "N".

   Refer to Section 6.2.3 for more information on the GENIN FILES statement.

2. Submit the sharedprefix.MICS.CNTL(cccCGEN) job to regenerate the component.

3. Edit prefix.MICS.PARMS(DBMODEL) for all database units in which the component is installed.

   On the FILE statement for the file you want to delete, either omit an AUDIT keyword, in which case the definition from cccGENIN is taken, or define AUDIT(NO).

   Refer to Section 2.3.4.1.2 in the PIOM for more information on the FILE statement in DBMODEL.

4. Submit the prefix.MICS.CNTL(CYCLEGEN) job for all database units in which the component is installed.

   This job changes the component's executable code in the prefix.MICS.USER.SOURCE libraries and affects only the database units in which the component is installed.

5. Edit prefix.MICS.PARMS(JCLGENU) for all database units in which the component is installed. Specify:

   ```
   WEEKLY AUDIT
   ```

6. Submit the prefix.MICS.CNTL(JCLGENU) job(s).

   This generates the WEEKLY and/or AUDIT job JCL that contains DD statements for the new audit tape configuration.

For example, assume the NPANCP file example showed the file appearing in the following archives:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  YES       YES       YES
```

Assume the desired configuration is:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        YES       YES
```

```
The  initial definition of the NPANCP file statement in
sharedprefix.MICS.GENLIB(NPAGENIN) would have been:

    FILE NCP 00 1 Y Y Y Y N Y Y Y NCP Activity File
                            | | |
      WEEKLY audit   -------+ | |
      WEEKLY history ---------+ |
      MONTHLY history-----------+

    Change this statement to:

    FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
                            |
                            +- no WEEKLY audit produced
```

6.2.3.3 Add File to Weekly History Archive Tape

CA MICS includes a facility to copy a file's data to offline storage (a tape generation data group) every week. Such data may be useful for maintaining long periods of data for large volume historical analyses without using undue amounts of online storage. A file eligible for weekly history archiving must exist in the WEEKS time-span.

To add a file to the weekly history archive tape, which is generated by the CA MICS WEEKLY or the standalone HISTW operational job, do the following:

1. Edit the FILE statement in sharedprefix.MICS.GENLIB(cccGENIN). Change the "N" in the the file's weekly history archive position to "Y".

   Refer to Section 6.2.3 for more information on the GENIN FILES statement.

2. Submit the sharedprefix.MICS.CNTL(cccCGEN) job to regenerate the component.

3. Edit prefix.MICS.PARMS(DBMODEL) for all database units in which the component is installed.

   To add the file, the FILE statement for that file must have a cutoff limit greater than zero in the weekly history archive position. Define a zero cutoff limit in units where you don't want to create this file.

   Refer to Section 2.3.4.1.2 in the PIOM for more information on the FILE statement in DBMODEL.

4. Submit the prefix.MICS.CNTL(CYCLEGEN) job for all database units in which the component is installed.

   This job changes the component's executable code in the prefix.MICS.USER.SOURCE libraries and affects only the database units in which the component is installed.

5. Review prefix.MICS.PARMS(JCLDEF) NULLSER keyword.

   If you have EXISTING history archive tapes for this unit in an automatic tape loader (ATL), you may need to change the NULLSER value to the volume serial number of the existing history archive tape.

   For JES2 customers -- if left unchanged, you may see a tape mount for volume serial NULSER; however, the WEEKLY job will complete normally, regardless of the unsatisfied tape mount request.

   For JES3 customers -- if left unchanged, the WEEKLY job will not continue because the tape mount cannot be satisfied. The job waits.

   Refer to Section 2.3.3.2.1.5 of the PIOM for more information on the use of the NULLSER keyword.

To determine the volume serial number of an existing history archive tape, under TSO option 6, issue the command:

```
 LISTC EN('tapeprefix.HISTW.iiifff') ALL
```

for a file which is currently active for history archive. You may refer to the WEEKLY job WEEK300 step (or the standalone HISTW job if you activated the standalone archive weekly history job option) and choose any one of the HIiiifff DDs.

Repeat this step for every unit which has this component installed.

6. Edit prefix.MICS.PARMS(JCLGENU) for all database units in which the component is installed. Specify:

   ```
   WEEKLY HISTW
   ```

   If you made a change to the NULLSER value as stated in the previous step, specify:

   ```
   WEEKLY HISTW GDGSGEN
   ```

   instead.

7. Submit the prefix.MICS.CNTL(JCLGENU) job(s).

   This generates the WEEKLY and/or HISTW job JCL that contains DD statements for the new weekly history tape files, as well as recreates the GDGSGEN job with the volume serial number that the first generation of the new file is cataloged to (with the NULLSER value you chose).

8. Submit prefix.MICS.CNTL(GDGSGEN) for all database units in which the component is installed.

   This creates weekly history tape generation data groups for the new archive file(s), if necessary.

For example, assume the NPANCP file appeared in the following archives:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        NO        YES
```

Assume the desired configuration is:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        YES       YES
```

The initial definition of the NPANCP file statement in sharedprefix.MICS.GENLIB(NPAGENIN) would be:

```
    FILE NCP 00 1 Y Y Y Y N N N Y NCP Activity File
                              | | |
      no WEEKLY audit  -----+ | |
      no WEEKLY history-------+ |
      MONTHLY history  ---------+
```

Change this statement to:

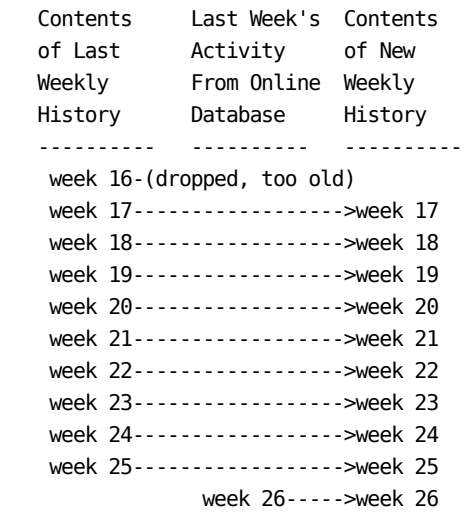
```
    FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
                              |
                              +- WEEKLY history produced
```

The number of cycles of the file that CA MICS will use as input to the weekly history archive is fixed at 1. This means the last cycle of WEEKS time-span data will be copied to the new generation of the weekly history archive tape when the weekly history is generated in the CA MICS WEEKLY or standalone HISTW operational job. The data from the database is added to the previous generation of the file's weekly history tape.

The test of what observations to put on the archive file is: any observation from a date later than the specified cutoff date will be copied to the archive file. No observations from the current week will be included in the input.

The cutoff date is determined by the user specification of the weekly history retention limit in the DBMODEL member of the prefix.MICS.PARMS library.

For example, assume the retention limit was 010. The previous ten weeks of weekly history would be available on any generation of the weekly history tape. Also, assume the week just processed by the current CA MICS WEEKLY or HISTW run (now cycle NPANCP01) is for week 26 of the year.



6.2.3.4 Delete File From Weekly History Archive Tape

There are two ways to delete a file from the weekly history archive tape, which is generated by the CA MICS WEEKLY or standalone HISTW operational job. The first only affects one database unit, the second affects all database units in which the component owning the file is installed.

To delete a file from the weekly history archive tape of a particular unit, without affecting other units, do the following:

1. Edit prefix.MICS.PARMS(DBMODEL).

   On the FILE statement for the file you want to delete, specify a cutoff limit of zero in the weekly history archive position.

   Refer to Section 2.3.4.1.2 in the PIOM for more information on the FILE statement in DBMODEL.

2. Submit the prefix.MICS.CNTL(CYCLEGEN) job.

   This job changes the component's executable code in the prefix.MICS.USER.SOURCE library and affects only the database unit in which the component is installed.

3. Edit prefix.MICS.PARMS(JCLGENU). Specify:

   ```
   WEEKLY HISTW
   ```

4. Submit the prefix.MICS.CNTL(JCLGENU) job.

   This generates the WEEKLY and/or HISTW job JCL that contains DD statements for the new weekly history configuration.

To delete a file from the weekly history archive tape, affecting all database units in which the component is installed, do the following:

1. Edit the FILE statement in sharedprefix.MICS.GENLIB(cccGENIN). Change the "Y" in the file's weekly history archive position to "N".

   Refer to Section 6.2.3 for more information on the GENIN FILES statement.

2. Submit the sharedprefix.MICS.CNTL(cccCGEN) job to regenerate the component.

3. Submit the prefix.MICS.CNTL(CYCLEGEN) job for all database units in which the component is installed.

   This job changes the component's executable code in the prefix.MICS.USER.SOURCE libraries and affects only the database units in which the component is installed.

4. Edit prefix.MICS.PARMS(JCLGENU) for all database units in which the component is installed. Specify:

   ```
   WEEKLY HISTW
   ```

5. Submit the prefix.MICS.CNTL(JCLGENU) job(s).

   This generates the WEEKLY and/or HISTW job JCL that contains DD statements for the new weekly history configuration.

For example, assume the NPANCP file example showed the file appearing in the following archives:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        YES       YES
```

Assume the desired configuration is:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        NO        YES
```

The initial definition of the NPANCP file statement in sharedprefix.MICS.GENLIB(NPAGENIN) would have been:

```
    FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
                             | | |
      no WEEKLY audit  -----+ | |
      WEEKLY history   -------+ |
      MONTHLY history  ---------+
```

Change this statement to:

```
    FILE NCP 00 1 Y Y Y Y N N N Y NCP Activity File
                               |
                               +- no WEEKLY history produced
```

6.2.3.5 Add File to Monthly History Archive Tape

CA MICS includes a facility to copy a file's data to offline storage (a tape generation data group) every month. Such data may be useful for maintaining long periods of data for large volume historical analyses without using undue amounts of online storage. A file eligible for monthly history archiving must exist in the MONTHS time-span.

To add a file to the monthly history archive tape, which is generated by the CA MICS MONTHLY or the standalone HISTM operational job, do the following:

1. Edit the FILE statement in sharedprefix.MICS.GENLIB(cccGENIN). Change the "N" in the the file's monthly history archive position to "Y".

   Refer to Section 6.2.3 for more information on the GENIN FILES statement.

2. Submit the sharedprefix.MICS.CNTL(cccCGEN) job to regenerate the component.

3. Edit prefix.MICS.PARMS(DBMODEL) for all database units in which the component is installed.

   To add the file, the FILE statement for that file must have a cutoff limit greater than zero in the monthly history archive position. Define a zero cutoff limit in units where you don't want to create this file.

   Refer to Section 2.3.4.1.2 in the PIOM for more information on the FILE statement in DBMODEL.

4. Submit the prefix.MICS.CNTL(CYCLEGEN) job for all database units in which the component is installed.

   This job changes the component's executable code in the prefix.MICS.USER.SOURCE libraries and affects only the database units in which the component is installed.

5. Review prefix.MICS.PARMS(JCLDEF) NULLSER keyword.

   If you have EXISTING history archive tapes for this unit in an automatic tape loader (ATL), you may need to change the NULLSER value to the volume serial number of the existing history archive tape.

   For JES2 customers -- if left unchanged, you may see a tape mount for volume serial NULSER; however, the MONTHLY job will complete normally, regardless of the unsatisfied tape mount request.

   For JES3 customers -- if left unchanged, the MONTHLY job will not continue because the tape mount cannot be satisfied. The job waits.

Refer to Section 2.3.3.2.1.5 of the PIOM for more information on the use of the NULLSER keyword.

To determine the volume serial number of an existing history archive tape, under TSO option 6, issue the command:

```
 LISTC EN('tapeprefix.HISTM.iiifff') ALL
```

for a file which is currently active for history archive. You may refer to the MONTHLY job MONTH300 step (or the standalone HISTM job if you activated the standalone archive monthly history job option) and choose any one of the HIiiifff DDs. Repeat this step for every unit which has this component installed.

6. Edit prefix.MICS.PARMS(JCLGENU) for all database units in which the component is installed. Specify:

   ```
   MONTHLY HISTM
   ```

   If you made a change to the NULLSER value as stated in the previous step, specify:

   ```
   MONTHLY HISTM GDGSGEN
   ```

   instead.

7. Submit the prefix.MICS.CNTL(JCLGENU) job(s).

   This generates the MONTHLY and/or HISTM job JCL that contains DD statements for the new monthly history tape files, as well as recreates the GDGSGEN job with the volume serial number that the first generation of the new file is cataloged to (with the NULLSER value you chose).

8. Submit prefix.MICS.CNTL(GDGSGEN) for all database units in which the component is installed.

   This creates monthly history tape generation data groups for the new archive file(s), if necessary.

For example, assume the NPANCP file appears in the following archives:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        YES       NO
```

Assume the desired configuration is:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        YES       YES
```

The initial definition of the NPANCP file statement in sharedprefix.MICS.GENLIB(NPAGENIN) would have been:

```
    FILE NCP 00 1 Y Y Y Y N N Y N NCP Activity File
                            | | |
      no WEEKLY audit   ----+ | |
      WEEKLY history    ------+ |
      no MONTHLY history--------+
```

Change this statement to:

```
    FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
                               |
                               +- MONTHLY history produced
```

The number of cycles of the file that CA MICS will use as input to the monthly history archive is fixed at 1. This means the last cycle of MONTHS time-span data will be copied to the new generation of the monthly history archive tape when the monthly history is generated in the CA MICS MONTHLY or standalone HISTM operational job. The data from the database is added to data on the previous generation of the file's monthly history tape.

The test of what observations to put on the archive file is: any observation from a date later than the specified cutoff date will be copied to the archive file. No observations from the current month will be included in the input.

The cutoff date is determined by the user specification of the monthly history retention limit in the DBMODEL member of the prefix.MICS.PARMS library.

For example, assume the retention limit was 010. The previous ten months of monthly history would be available on any generation of the monthly history tape. Also, assume the month just processed by the current CA MICS MONTHLY or HISTM run (now cycle NPANCP01) is for month 11 of the year.

```
Contents     Last Month's Contents
of Last      Activity     of New
Monthly      From Online  Monthly
History      Database     History
----------   ----------   ----------
month 01-(dropped, too old)
month 02----------------->month 02
month 03----------------->month 03
month 04----------------->month 04
month 05----------------->month 05
month 06----------------->month 06
month 07----------------->month 07
month 08----------------->month 08
month 09----------------->month 09
month 10----------------->month 10
             month 11---->month 11
```

6.2.3.6 Delete File From Monthly History Archive Tape

There are two ways to delete a file from the monthly history archive tape, which is generated by the CA MICS MONTHLY or standalone HISTM operational job. The first only affects one database unit, the second affects all database units in which the component owning the file is installed.

To delete a file from the monthly history archive tape of a particular unit, without affecting other units, do the following:

1. Edit prefix.MICS.PARMS(DBMODEL).

   On the FILE statement for the file you want to delete, specify a cutoff limit of zero in the monthly history archive position.

   Refer to Section 2.3.4.1.2 in the PIOM for more information on the FILE statement in DBMODEL.

2. Submit the prefix.MICS.CNTL(CYCLEGEN) job.

   This job changes the component's executable code in the prefix.MICS.USER.SOURCE library and affects only the database unit in which the component is installed.

3. Edit prefix.MICS.PARMS(JCLGENU). Specify:

   ```
   MONTHLY HISTM
   ```

4. Submit the prefix.MICS.CNTL(JCLGENU) job.

   This generates the MONTHLY and/or HISTM job JCL that contains DD statements for the new monthly history configuration.

To delete a file from the monthly history archive tape, affecting all database units in which the component is installed, do the following:

1. Edit the FILE statement in sharedprefix.MICS.GENLIB(cccGENIN). Change the "Y" in the file's monthly history archive position to "N".

   Refer to Section 6.2.3 for more information on the GENIN FILES statement.

2. Submit the sharedprefix.MICS.CNTL(cccCGEN) job to regenerate the component.

3. Submit the prefix.MICS.CNTL(CYCLEGEN) job for all database units in which the component is installed.

   This job changes the component's executable code in the prefix.MICS.USER.SOURCE libraries and affects only the database units in which the component is installed.

4. Edit prefix.MICS.PARMS(JCLGENU) for all database units in which the component is installed. Specify:

   ```
   MONTHLY HISTM
   ```

5. Submit the prefix.MICS.CNTL(JCLGENU) job(s).

   This generates the MONTHLY and/or HISTM job JCL that contains DD statements for the new monthly history configuration.

For example, assume the NPANCP file example showed the file appearing in the following archives:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        YES       YES
```

Assume the desired configuration is:

```
 WEEKLY   WEEKLY    MONTHLY
 AUDIT    HISTORY   HISTORY
--------  --------  --------
  NO        YES       NO
```

The initial definition of the NPANCP file statement in sharedprefix.MICS.GENLIB(NPAGENIN) would have been:

```
    FILE NCP 00 1 Y Y Y Y N N Y Y NCP Activity File
                            | | |
      no WEEKLY audit  -----+ | |
      WEEKLY history   -------+ |
      MONTHLY history  ---------+
```

Change this statement to:

```
    FILE NCP 00 1 Y Y Y Y N N N Y NCP Activity File
                              |
  no MONTHLY history produced-+
```

6.3 Retrofitting Guidelines

This section deals with remedial modifications to the CA MICS database. There are several situations that may call for such remedial measures.

Bad data may get onto the CA MICS database. This may be caused by a user exit or logic modification not working as intended, or may be due to an operating system or subsystem malfunction (e.g. bad data from SMF). Whatever the cause, the data elements affected should be treated as soon after the anomaly is recognized as possible.

Adding a new control data element to a file can be a complicated procedure. If the file already exists, salvaging as much as possible of the previous data on the file requires careful planning and a lot of attention to detail.

Recovering a bad archive file may be more or less difficult, according to the type of loss of data suffered. Generally, archives contain large amounts of data, and this volume issue can complicate the procedure.

It is necessary to have a thorough understanding of the material presented thus far in chapter 6 of this manual before changing data element content.

6.3.1 Correcting a File After Creation

Bad data values can enter CA MICS files from various sources. Among the reasons for such problems are modifications made to CA MICS logic, operating system and measurement device malfunctions, and misapplied data.

Data integrity problems may be of several types. Some of the data values can accurately be reconstructed, but some cannot. Usually, recovery of maximum and minimum value data elements and accumulated data elements is very difficult, while recovery of computed data elements is relatively easy.

6.3.1.1 General Recovery Job

The general format of a data element recovery job is that of a SAS DATA step. The data step reads the existing file cycle, modifies the value, and replaces the cycle with observations that have the proper data element values. In SAS, the DATA step follows this template:

```
DATA &diiit..iiifffnn (%fffFILE(OP=FILEOPTS,TS=ssssss));
SET  &diiit..iiifffnn;
(correction logic)
RUN;
```

where "d" is the database unit identifier
"iii" is the information area identifier
"t" is the timespan identifier
"fff" is the file identifier
"nn" is the file cycle number
"sssss" is the timespan name

SPECIAL NOTE: Examples in this section are based on a CA MICS component called NPA, the CA MICS Network Performance Analysis Component. Though that component has been superseded entirely by the CA MICS SNA Network Analyzer, references to NPA remain in the section as an illustration of how such tailoring operations would apply to any hypothetical CA MICS component.

For clarity, we will return to the NPA example used thus far in chapter 6 of this manual. Assume that the CA MICS NPA component has been installed in the CA MICS test database unit whose PROC identifier is "T". Treating a bad value in the NPANCP01 file in the DETAIL timespan would be done by a data step such as:

```
DATA &TNPAX..NPANCP01 (%NCPFILE(OP=FILEOPTS,TS=DETAIL));
SET  &TNPAX..NPANCP01;
(correction logic)
RUN;
```

6.3.1.2 Calculated Data Element Correction

The simplest correction to values on the database is the recalculation of calculated data elements, where the data elements used in the calculation are (by component definition) on the database, too.

The data element addition illustrated earlier in this chapter was NCPPCBSY. It was calculated using the actual observation time and the free cycle time. The statements added to sharedprefix.MICS.GENLIB(NPAGENIN) were:

```
...
TYPE C 3 MAPCT.   3 MAPCT.   3 MAPCT.
ALIAS USRPCNPA
NAME NCPPCBSY 00 0 0 0 0 0 3705 Percent Busy
EXP 01 IF NPATMAOT GT 0 THEN
EXP 02  NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
EXP 03 ELSE NCPPCBSY = 0;
DEPEND NCPTMFCT NPATMAOT
...
```

Assume that, originally, there was an error in the MCG EXP statement content so that the value was calculated incorrectly. The error might have been the omission of the 100 multiplier, as:

```
EXP 01 IF NPATMAOT GT 0 THEN
EXP 02  NCPPCBSY = (NPATMAOT - NCPTMFCT) / NPATMAOT;
EXP 03 ELSE NCPPCBSY = 0 ;
```

The first step in the correction is to fix the EXP 02 statement so that it contains the correct expression:

```
EXP 02  NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
```

Then rerun the NPACGEN job. This, however, leaves the existing cycles of the database with bad values in NCPPCBSY. Therefore, to fix cycle 01 of the DETAIL timespan, the SAS data statement would look like this:

```
DATA &TNPAX..NPANCP01 (%NCPFILE(OP=FILEOPTS,TS=DETAIL));
SET  &TNPAX..NPANCP01;
IF NPATMAOT GT 0 THEN
  NCPPCBSY = 100 * (NPATMAOT - NCPTMFCT) / NPATMAOT;
ELSE NCPPCBSY = 0;
RUN;
```

6.3.1.3 Correcting Timespans and Cycles

The prior section showed how to correct a problem in a single file's DETAIL timespan. If a problem existed in several timespans and in multiple cycles in each timespan, then all cycles in all affected timespans need a DATA step like the one shown in the prior section to treat the bad value.

Rather than coding one DATA step for each cycle of the file in each timespan affected, consider using SAS macros to provide substitutable code.

The macro method is relatively easy to use and test. It lends itself to applications where fewer cycles of each timespan are affected. The principle is to make a macro of the DATA statement, with substitutable DDname and file/cycle names, and execute the DATA step a number of times.

The correction logic macro might look like this:

```
%MACRO RETFIT(DDN,FN,FFF,TS);
DATA &DDN..&FN (%&FFF.FILE(OP=FILEOPTS,TS=&TS));
SET  &DDN..&FN;
(correction logic)
RUN;
%MEND RETFIT;
```

It could be called from a sequence such as:

```
%RETFIT(&TNPAX,NPANCP01,NCP,DETAIL)
%RETFIT(&TNPAX,NPANCP02,NCP,DETAIL)
%RETFIT(&TNPAX,NPANCP03,NCP,DETAIL)
...
%RETFIT(&TNPAX,NPANCPnn,NCP,DETAIL)
%RETFIT(&TNPAD,NPANCP01,NCP,DAYS)
%RETFIT(&TNPAD,NPANCP02,NCP,DAYS)
%RETFIT(&TNPAD,NPANCP03,NCP,DAYS)
...
%RETFIT(&TNPAD,NPANCPnn,NCP,DAYS)
...
```

6.3.1.4 Other Correction Types

For data elements of types other than calculated, the correction methods may be more difficult. Fortunately, such cases are relatively rare, and usually are discovered during test, before implementation. This is especially true for components generated with the MCG, because both generated code and documentation reflect the type of the data element.

Methods for recovering maximum, minimum, and accumulated data elements often result in the data element value being reset to zero or missing. This is true because it is often impractical to reprocess all the source data involved, and no other source of correct information is available.

In such cases, contact the CA MICS Product Support Group for suggestions on data element recovery.

6.3.2 Retrofitting Database Files

Changing the control data elements of a file amounts to changing the file's key structure. Adding or deleting a key data element usually affects the sort sequence of the file. Even changing the contents of existing key data elements can affect the sort sequence.

The general recovery job template shown in the previous section works for cases in which the key structure of the file is not changed (there is no sort and summarization of the treated data). If the key structure of the file is affected, there are several possibilities for file action:

- Value change only
- Value change, order change
- Value change, order change, and granularity change

6.3.2.1 Value Change Only

Each of these types involves a change to the value of a key data element. The simplest case is where there is a one-to-one mapping of old key data element values to new values. That is, for every old value, there is one and ONLY one new value.

The changed key will take the same order as the original key. For example, assume a file's key contains a character data element whose values are

```
AA01
AA02
BA01
```

If these values are changed to

```
AB01
AB02
BB01
```

then re-sorting the file observations results in no change to the order of observations. In this case, the retrofit job described in the previous section would perform the change correctly.

Perform the data element content correction according to the previous section. This correction must accompany the change to CA MICS or user exit logic that corrects the data element's value during the daily update cycle. For example, if the change is to sharedprefix.MICS.PARMS(ACCTRTE) to change SMF account level data elements, make the change to ACCTRTE and treat the database without running an intervening daily update job. Note that such complex-level changes require changes to all files on which the data elements occur, in all timespans and for all database units that have the component.

Be sure to apply the change to the test database first. Testing the retrofit ensures that the change is complete and and correct before implementing it in production. This is particularly necessary because, on implementation, complex-level changes affect all database units at once.

6.3.2.2 Value Change With Order Change

Assume that there is still a one-to-one mapping of the old key data element value to the new value. That is, every old key value translates to one and ONLY one new key value. However, in contrast to the last case, either the values will not take the same order as the old values, or it cannot be determined that the new values will take the same order as the old. For example, here is a sample table showing old and new key data element values and file observation order:

Old Value	Old Order	New Value	New Order
AA01	1	AB01	2
AA02	2	AB02	3
BA01	3	AA01	1

Not only must the data values be corrected, but the file must be re-sorted. The template for such a generalized recovery job becomes:

```
DATA WORK.iiifffnn;
  SET  &diiit..iiifffnn;
  (correction logic)
RUN;
PROC SORT DATA=WORK.iiifffnn
    OUT=&diiit..iiifffnn(%fffFILE(OP=FILEOPTS,TS=sssss));
  BY %fffSEQ(TS=sssss); RUN;
```

where d is the database unit identifier
iii is the information area identifier
t is the timespan identifier
fff is the file identifier
nn is the file cycle number
sssss is the timespan name

Putting the macro treatment method into practice can be done by a method similar to that used in the previous section. A template for this method is:

```
//xxx JOB (your job card)
//STEP1 EXEC MICSDBd
//SYSIN DD *
 %INCLUDE USOURCE($BASMSTR);
%MACRO RETFIT(DDN,III,FFF,CYC,TSPAN);
   %LOCAL FILEID;
   %LET FILEID = &III&FFF&CYC;
   DATA WORK.&FILEID (KEEP=
                       %&FFF.KEEP(TS=&TSPAN)
                      );;;;
   SET  &DDN..&FILEID;
   (correction logic)

 RUN;
 PROC SORT DATA=WORK.&FILEID
    OUT=&DDN..&FILEID(%&FFF.FILE(OP=FILEOPTS,TS=&TSPAN));
   BY %&FFF.SEQ(TS=&TSPAN); RUN;
 %MEND RETFIT;
  /* RETROFIT DETAIL FILES */
 %RETFIT(&diiiX,iii,fff,nn,DETAIL); /*RETROFIT 1ST CYCLE*/
  ...
 %RETFIT(&diiiX,iii,fff,nn,DETAIL); /*RETROFIT LAST CYCL*/
  /* RETROFIT MONTHS FILES */
 %RETFIT(&diiiM,iii,fff,nn,MONTHS); /*RETROFIT 1ST CYCLE*/
  ...
 %RETFIT(&diiiM,iii,fff,nn,MONTHS); /*RETROFIT LAST CYCL*/
```

Again, this database retrofit should be done at the same time as permanent corrections to CA MICS logic or user exit code. This assures that a single form of the key element value exists at a given time.

6.3.2.3 Value Change With Order and Granularity Changes

Assume that there is NOT a one-to-one mapping of the old key data element value to the new value. This situation may occur under several conditions:

- A key data element is being added to the file.
- A key data element is being deleted from the file.
- The contents of an existing key data element must be changed in other than a one-to-one mapping of values.
- The length of an existing key data element must be changed, and the added or deleted material may change file granularity.

For example, here is a sample table showing old and new key data element values and file observation order:

Old Value	Old Order	New Value	New Order
AA01	1	AA	1
AA02	2	AA	1
BA01	3	BA	2

Not only must the data values be corrected, but the file must be re-sorted AND resummarized. The template for such a generalized recovery job becomes:

```
%LET   R_BY    = %fffSEQ(TS=sssss) ;
%LET   R_BREAK = %fffSEQ(TS=sssss,OP=BREAK) ;
DATA TEMP (KEEP= %fffKEEP(TS=sssss) );
  SET  &diiit..iiifffnn;
  (correction logic)
RUN;
PROC SORT DATA=TEMP;
  BY &R_BY ;
RUN;
DATA &diiit..iiifffnn (%fffFILE(TS=sssss,OP=FILEOPTS));
  %fffLEN(TS=sssss,TYPE=BOTH);
  %fffFMT(TS=sssss);
  SET  TEMP;
  %fffSUM(SUMBY=&R_BY , SUMBREAK=&R_BREAK );
RUN;
```

```
where fff   is the file identifier
      sssss is the timespan name (DETAIL, DAYS, etc.)
      d     is the database unit identifier
      nn    is the file cycle number
      iii   is the information area identifier
      t     is the timespan identifier (X, D, etc.)
```

Putting the macro treatment method into practice can be done by a method similar to that used in the previous example. This example illustrates the method applied to the example NPA component.

```
    //xxx JOB (your job card)
    //STEP1 EXEC MICSDBS      (assume unit identifier "S")
    //SYSIN DD *
    %INCLUDE USOURCE($BASMSTR);
    %MACRO RETFIT(DDN,III,FFF,CYC,TSPAN);
      %LOCAL FILEID R_BY R_BREAK;
      %LET   FILEID  = &III&FFF&CYC;
      %LET   R_BY    = %&FFF.SEQ(TS=&TSPAN);
      %LET   R_BREAK = %&FFF.SEQ(TS=&TSPAN,OP=BREAK);
      DATA TEMP;
        %&FFF.LEN(TS=&TSPAN,TYPE=BOTH);
        %&FFF.FMT(TS=&TSPAN);
        %&FFF.LBL;
        SET  &DDN..&FILEID;
        (correction logic)
      RUN;
      PROC SORT DATA=TEMP; BY &R_BY; RUN;
      DATA &DDN..&FILEID (%&FFF.FILE(TS=&TSPAN,OP=FILEOPTS)
                         );;;;
        %&FFF.LEN(TS=&TSPAN,TYPE=BOTH);
        %&FFF.FMT(TS=&TSPAN);
        %&FFF.LBL;
        SET  TEMP;
        %&FFF.SUM(SUMBY=&R_BY,SUMBREAK=&R_BREAK);
      RUN;
    %MEND RETFIT;
     /* RETROFIT DETAIL FILES */
    %RETFIT(&SNPAX,NPA,NCP,01,DETAIL); /*RETROFIT 1ST CYCLE*/
     ...
    %RETFIT(&SNPAX,NPA,NCP,nn,DETAIL); /*RETROFIT LAST CYCL*/
     /* RETROFIT MONTHS FILES */
    %RETFIT(&SNPAM,NPA,NCP,01,MONTHS); /*RETROFIT 1ST CYCLE*/
     ...
    %RETFIT(&SNPAM,NPA,NCP,nn,MONTHS); /*RETROFIT LAST CYCL*/
```

Again, this database retrofit should be done at the same time as permanent corrections to CA MICS logic or user exit code. This assures that a single form of the key element value exists at a given time.

Here is a sample table showing the opposite granularity problem. New key data element values have a finer granularity than the old:

Old Value	Old Order	New Value	New Order
AA01	1	AA011	1
		AA012	2
		AA013	3
AA02	2	AA021	4
		AA022	5
BA01	3	BA011	6

The retrofitting of key data element values at the DETAIL timespan can be done easily, because no summarization is necessary. However, higher levels of summarization have already taken place. Data in the database in DAYS, WEEKS, MONTHS and YEARS has been summarized to a higher level of granularity than is desired, and splitting the data back into its component parts can be difficult. In fact, an accurate method of doing this often involves reprocessing input source data. This may be done using a special database. (Special databases are described in the Planning, Installation, Operation, and Maintenance Guide.)

It is possible to avoid such situations by doing a complete analysis of key data element requirements before the files are structured. But special requirements often present themselves after the database has been in production for some time.

It may be more advantageous to treat the database to accommodate the new data elements or adjusted lengths, so the key structures of existing observations are compatible with the new method, but leave old values.

Take the example of adding a new data element to the key. If there is no source for calculating the value of the new data element from summarized data, add the data element to higher timespans with a blank value (zero for numeric data elements). This gives you the ability to extract data by the new key for all data added to the database after the key change, and to access all the data on the database by the higher (old) granularity as well.

Contact CA MICS Product Support for more information on performing this difficult retrofit process.

6.3.3 Retrofitting Archive Files

There are several maintenance issues for archive files. Data element changes and archive reconstructions are the two major types of maintenance discussed here.

Generally, due to the tape involvement, manipulating files in the offline database is more cumbersome than manipulating them in the online database. History archive files contain cumulative data, so applying retrofit logic amounts to creating one new generation of the history file that contains all cumulative data with the change applied. Audit archives contain "snapshot" data, normally at one week of detailed data per generation, so applying a data element retrofit to audit files involves rewriting all generations of a given data set.

6.3.3.1 Data Element Changes

Data element changes apply urgently to the online database, but less urgently to the offline database. That is, archive files must be treated before the next archive update, and do not necessarily have to be made simultaneously with the online database retrofit and the permanent logic change.

In fact, the archive files really do not have to be retrofitted until they are used. The audit archive generally contains one week's picture of the online files, which will hold data of either the old values or the new values, but not both. The weekly and monthly history files are not created in sorted order, and typically have to be sorted and resummarized during use, anyway. Thus it is possible to save the retrofit logic and apply the logic when the archive is used, rather than retrofit the entire archive.

There are several significant compromises with this non-solution to archive maintenance. The archive files contain data for long periods, and may span several database data element changes. This means cumulative change logic would have to be generated and tested, increasing the possibility of harm to the offline database.

Changing data element values on archive files should be done concurrently with the changes to the online database. The general recovery job templates shown in the previous section work for cases in which the input and output SAS files can be on the same DDname. The same possibilities for file action exist, according to the data element changed:

- Value change only
- Value change, order change, and granularity change

6.3.3.1.1 Value Change Only

The simplest case is where there is a change to a non-key data element, or to a key data element where there is a one-to-one mapping of old key data element values to new values. That is, for every old value there is one and ONLY one new value.

For key data elements, the changed key will take the same order as the original key. For example, assume a file's key contains a character data element whose values are

AA01
AA02
BA01

If these values are changed to

AB01
AB02
BB01

then re-sorting the file observations results in no change to the order of observations. In this case, the retrofit job described in the previous section performs the change correctly.

Perform the data element content correction according to the previous sections. This correction must accompany the change to CA MICS or user exit logic that corrects the data element's value during the daily update cycle. For example, if the change is to sharedprefix.MICS.PARMS(ACCTRTE) to change SMF account level data elements, make the change to ACCTRTE and treat the online database without running an intervening daily update job.

Note that such complex-level changes require changes to all files on which the data elements occur, in all timespans and for all database units that have the component. Offline databases need not be retrofitted during the same break in daily update processing, though they must be updated before their next use. Note that the next use of the offline database may be an ad hoc interrogation or may be the next WEEKLY or MONTHLY update run that creates new generations of the archive.

Be sure to apply the change to the test database first. Testing the retrofit ensures that the change is complete and correct before it is implemented in production. This is particularly necessary for archive changes, due to the added complexity of applying the retrofit.

The general format of a data element recovery job is that of a SAS data statement. The data statement reads the existing file, modifies the value, and creates a new file with observations that have the proper data element values. In SAS, the data statement follows this template:

```
DATA outddname.iiifff;
  SET  inddname.iiifff;
  (correction logic)
RUN;
```

where "iii" is the information area ID
"fff" is the file ID

The DDnames referenced refer to the DDname of the input or output tape DDname in the JCL for the archive data set's access. History tape DDnames take the form:

```
outddname - Htiiifff
inddname  - Itiiifff
```

```
where "iii" is the information area ID
      "fff" is the file ID
      "t" is W for weekly history, M for monthly history

    Audit tape DDnames take the form:

    outddname - AUiiifff
    inddname  - AIiiifff

where "iii" is the information area ID
      "fff" is the file ID

The example of retrofitting a new value into a data
element of the SMF files is used again.  The composite
template for this method for treating history files is:

    //xxx JOB (your job card)
    //STEP1 EXEC MICSSHRS     (assume proc identifier "S")
    //HMBATJOB DD DISP=(,CATLG,DELETE),
    //            DCB=(tapeprefix.MICS.MODEL,DEN=density),
    //            DSN=tapeprefix.MICS.HISTM.BATJOB(+1),
    //            UNIT=tapeunit,LABEL=(1,EXPDT=...)
    //IMBATJOB DD DISP=OLD,
    //            DSN=tapeprefix.MICS.HISTM.BATJOB(0),
    //            VOL=(PRIVATE,RETAIN),UNIT=(,,DEFER)
    //HMBATPGM DD DISP=(,CATLG,DELETE),
    //            DCB=(tapeprefix.MICS.MODEL,DEN=density),
    //            DSN=tapeprefix.MICS.HISTM.BATPGM(+1),
    //            UNIT=AFF=HMBATJOB,LABEL=(2,EXPDT=...)
    //IMBATPGM DD DISP=OLD,
    //            DSN=tapeprefix.MICS.HISTM.BATPGM(0),
    //            VOL=(PRIVATE,RETAIN),UNIT=AFF=IMBATJOB
    //SYSIN DD *
    %MACRO RETFIT(DDI,DDO,FN);
      DATA &DDO..&FN;
        SET  &DDI..&FN;
        (correction logic)
      RUN;
    %MEND RETFIT;
    %RETFIT(IMBATJOB,HMBATJOB,BATJOB);
    %RETFIT(IMBATPGM,HMBATPGM,BATPGM);
    ...
```

where the DD statement coding is patterned after the associated coding in the WEEKLY or MONTHLY job 300 step.

After the history retrofit is complete, the next WEEKLY or MONTHLY job would call for extra tape mounts. This is because some of the history data sets would be input from the retrofitted tape just created, and some would still be input from the previous history tape.

Audit tape treatment is significantly more complicated. The audit files exist on one generation data group entry per week of audit data archived. That means every previous generation of audit archive for the file(s) in question must be regenerated.

Assume there were 4 generations of audit archive saved, and that the current processing week is week 41. The data set names cataloged would be, for example:

Data Set Name	Data Contained
dsn(0)	detail data from week 40, untreated
dsn(-1)	detail data from week 39, untreated
dsn(-2)	detail data from week 38, untreated
dsn(-3)	detail data from week 37, untreated

There are various methods to process the generations to arrive at the treated configuration:

Data Set Name	Data Contained
dsn(0)	detail data from week 40, treated
dsn(-1)	detail data from week 39, treated
dsn(-2)	detail data from week 38, treated
dsn(-3)	detail data from week 37, treated

For treating audit files, this is a sample template:

```
//xxx JOB (your job card)
//STEP1 EXEC MICSSHRS     (assume proc identifier "S")
//AUBATJOB DD DISP=(,CATLG,DELETE),
//            DCB=(tapeprefix.MICS.MODEL,DEN=density),
//            DSN=tapeprefix.MICS.AUDIT.BATJOB(+1),
//            UNIT=tapeunit,LABEL=(1,EXPDT=...)
//AIBATJOB DD DISP=OLD,
//            DSN=tapeprefix.MICS.AUDIT.BATJOB(-nn),
//            VOL=(PRIVATE,RETAIN),UNIT=(,,DEFER)
//AUBATPGM DD DISP=(,CATLG,DELETE),
//            DCB=(tapeprefix.MICS.MODEL,DEN=density),
//            DSN=tapeprefix.MICS.AUDIT.BATPGM(+1),
//            UNIT=AFF=AUBATJOB,LABEL=(2,EXPDT=...)
//AIBATPGM DD DISP=OLD,
//            DSN=tapeprefix.MICS.AUDIT.BATPGM(-nn),
//            VOL=(PRIVATE,RETAIN),UNIT=AFF=AIBATJOB
//SYSIN DD *
%MACRO RETFIT(DDI,DDO,FN);
  DATA &DDO..&FN;
  SET  &DDI..&FN;
  (correction logic)
  RUN;
%MEND RETFIT;
%RETFIT(AIBATJOB,AUBATJOB,BATJOB);
```

```
    %RETFIT(AIBATPGM,AUBATPGM,BATPGM);
    ...
```

where the DD statement coding is patterned after the associated coding in the WEEKLY or MONTHLY job 300 step.

The "-nn" in the input data set name relates to the number of generations of audit archive retained. For 4 generations, for example, "-nn" equal to "-3" would get the last retained generation. Coding -3 in our example would yield the following data:

After one run:

Data Set Name	Data Contained
dsn(0)	detail data from week 37, treated
dsn(-1)	detail data from week 40, untreated
dsn(-2)	detail data from week 39, untreated
dsn(-3)	detail data from week 38, untreated

After the second run:

Data Set Name	Data Contained
dsn(0)	detail data from week 38, treated
dsn(-1)	detail data from week 37, treated
dsn(-2)	detail data from week 40, untreated
dsn(-3)	detail data from week 39, untreated

After the third run:

Data Set Name	Data Contained
dsn(0)	detail data from week 39, treated
dsn(-1)	detail data from week 38, treated
dsn(-2)	detail data from week 37, treated
dsn(-3)	detail data from week 40, untreated

After the fourth run:

Data Set Name	Data Contained
dsn(0)	detail data from week 40, treated
dsn(-1)	detail data from week 39, treated
dsn(-2)	detail data from week 38, treated
dsn(-3)	detail data from week 37, treated

This is the configuration we wanted. Coding the JCL for a retrofit of the audit archive files as shown requires that the job step be run a number of times exactly equal to the number of generations of audit archive tapes in the catalog at the time the retrofit is done.

There are various ways to improve the running of such an audit retrofit. Other ways involve uncataloging and recataloging the audit data sets for the audited files affected.

6.3.3.1.3 Value Change With Order and Granularity Changes

If there is not a one-to-one mapping of the old key data element value to the new value, the data values must be corrected, and the file must be re-sorted AND resummarized. This was shown in the previous section. The template for such a generalized recovery job becomes:

```
%MACRO RETFIT(DDI,DDO,III,FFF,TSPAN);
  %LOCAL FILEID R_BY R_BREAK ;
  %LET FILEID = &III&FFF;
  %LET R_BY   = %&FFF.SEQ(TS=&TSPAN) ;
  %LET R_BREAK= %&FFF.SEQ(TS=&TSPAN,OP=BREAK);
  DATA TEMP;
    SET  &DDI..&FILEID;
    (correction logic)
  RUN;
  PROC SORT DATA=TEMP;
  BY &R_BY;
  RUN;
  DATA &DDO..&FILEID (KEEP=
                      %&FFF.KEEP(TS=&TSPAN)
                     );;;;
    SET  TEMP;
    %&FFF.SUM(SUMBY=&R_BY, SUMBREAK=&R_BREAK);
  RUN;
%MEND RETFIT;
```

where parameters are as noted in previous illustrations.

Again, this database retrofit can be done as soon as permanent corrections to CA MICS logic or user exit code are done. This ensures that a single form of the key element value exists at a given time.

Also, it is necessary to treat the history files fully. All history files that have granularity changes MUST be sorted (sorting is required) as a prerequisite to resummarization.

The problem with increasing granularity, noted in the previous section, exists for treating archives. The retrofitting of key data element values at the DETAIL timespan for audit archives can be done easily, because no summarization needs to be done. However, higher levels of summarization have already taken place. Audit archive data in DAYS granularity, and WEEKS and MONTHS history archives, has been summarized to a higher level of granularity than is desired, and splitting the data back into its component parts can be difficult.

Contact the CA MICS Product Support Group for more information on performing this difficult retrofit process.

6.3.3.2 Reconstructing Archive Files

Reconstruction of an archive file is relatively easy, if the data still exists on the online database.

A missing audit archive can be reconstructed by rerunning a part of the WEEKLY job extracted from the WEEK300 step for any database unit (or the standalone AUDIT job if you activated the standalone archive audit job option). However, the audit archive logic will only input the last 10 cycles of DETAIL timespan data (or DAYS data for the CICS and IMS components). To create a SAS program to reconstruct the audit archive, examine the prefix.MICS.USER.SOURCE(WKcccAUD) module that contains standard CA MICS logic to archive the audit data.

A typical WKcccAUD module follows this form:

```
DATA AUiiifff.iiifff  (KEEP=%fffKEEP(TS=ssssssss) );
  SET  &iiit..iiifff01          &iiit..iiifff02
       &iiit..iiifff03          &iiit..iiifff04
       &iiit..iiifff05          &iiit..iiifff06
       &iiit..iiifff07          &iiit..iiifff08
       &iiit..iiifff09          &iiit..iiifff10
       ;
  RETAIN THISWEEK;
  IF _N_ = 1 THEN DO;
    RETAIN HLDWK HLDENDTS; DROP HLDWK HLDENDTS;
    HLDENDTS=ENDTS; HLDWK=WEEK;
    ENDTS="&SYSDATE:00:00:00"DT; %WEEK; THISWEEK=WEEK;
    ENDTS=HLDENDTS; WEEK=HLDWK;
  END;
  IF WEEK NE THISWEEK;
RUN;
```

To consider more online data for input to the audit creation process, increase the number of cycles of the online data. Add cycles 11, 12, and so on, to the maximum number of cycles that pertain to the reconstruction or to the maximum number of cycles of that timespan on the database (whichever is less).

NOTE: The actual code executed for this process is caused by the resolution of a reference to autocall macro WKAUD, member of sharedprefix.MICS.MACAUTOS. This macro is referenced by code generated in the WKcccAUD member above.

A bad history archive can be reconstructed by rerunning a part of the WEEKLY job, extracted from the WEEK300 step (or the standalone HISTW job if you activated the standalone archive weekly history job option), or the MONTHLY job, extracted from the MONTH300 step, (or the standalone HISTM job if you activated the standalone archive monthly history job option), for any database unit. However, the history logic will input the last history file and one cycle of WEEKS or MONTHS timespan data. To create a SAS program to reconstruct the history archive, examine the prefix.MICS.USER.SOURCE(WKcccHST) or MNcccHST module that contains standard CA MICS logic to archive the history data.

A typical WKcccHST module follows this form:

```
DATA HWiiifff.iiifff  (KEEP=%fffKEEP(TS=WEEKS) );
  SET  &iiiW..iiifff01  HIiiifff.iiifff;
  RETAIN CUTOFF;
  IF _N_ = 1 THEN DO;
    CUTOFF=("&SYSDATE"D - ((&Wiiifff + 1) * 7)) * 86400;
  END;
  IF CUTOFF < ENDTS;
RUN;
```

A typical MNcccHST module follows this form:

```
DATA HMiiifff.iiifff  (KEEP=%fffKEEP(TS=MONTHS) );
  SET  &iiiM..iiifff01  HIiiifff.iiifff;
  RETAIN CUTOFF;
  IF _N_ = 1 THEN DO;
    CUTOFF=("&SYSDATE"D - ((&Miiifff + 1) * 31)) * 86400;
  END;
  IF CUTOFF < ENDTS;
RUN;
```

To consider more online data for input to the history creation process, increase the number of cycles of the online data. Add cycles 2, 3, and so on, to the maximum number of cycles that pertain to the reconstruction or to the maximum number of cycles of that timespan on the database (whichever is less).

NOTE: The actual code executed for these two processes is caused by the resolution of references to autocall macros WKHST and MNHST, members of sharedprefix.MICS.MACAUTOS. These macros are referenced by code generated in the WKcccHST and MNcccHST members above. The WKHST and MNHST macros have an automatic second level of history built into them, which causes MONTHS timespan cycles 01 AND 02 both to be input to the history process.

For more information on reducing the number of cycles retained online, see Section 4.3.3.2.19 in the PIOM, AUDIT Operational Job.

Chapter 7: Writing Field-developed Applications

The most significant change that can be made to a CA MICS system is the addition of a field-developed application (FDA). Implementing a field-developed application involves a thorough knowledge of how CA MICS products are structured and considerations of input integrity, reliability, and maintainability that accompany a product's implementation.

This section contains the following topics:

7.1 Development Steps (see page 373)
7.2 Deciding What the Application Must Do (see page 374)
7.3 Writing the Code (see page 388)
7.4 Testing, Documenting and Deploying the FDA (see page 424)
7.5 MSI Definition Notes (see page 429)
7.6 MSI Unit Level Parameters (see page 433)

7.1 Development Steps

A CA MICS product is a processing entity that fits into the normal update flow of the CA MICS system. It processes approximately one day's worth of data from some source and creates a single cycle of one or more CA MICS database files to complement the rest of the CA MICS database.

A field-developed application (FDA, or user-written product) treats some input source of special interest your site, assures the applicability and integrity of the data, formats the data to be compatible with CA MICS, and stores the data on the CA MICS database.

Defining an FDA through CA MICS facilities relieves you of most of the chore of coding the product's code. That is, defining a CA MICS FDA through CA MICS Component Generator (MCG) statements allows the MCG to generate SAS logic that defines the content of the files in each timespan in which each file is supported, summarizes all files to succeeding timespans, and ages the cycles of all timespans for all files in the product.

To define a product to CA MICS, you must:

1. Determine what you need the product to do
   - what is its input source?
   - what reports do you need from the product?
   - what files organize the information for reports?

2. Write the code
   - write the product definition (fdaGENIN)
   - write the input format routine (DYfdaFMT)
   - write the reports (MICF inquiries, DYfdaMBO, and/or DYfdaEXC and fdaEVA)

3. Test and implement the code
   - run the CA MICS generation jobs
   - test the product's processing and reports
   - write documentation
   - turn over to production status

In this chapter, the principles of writing a field-developed application are illustrated through the development of two FDAs, the UNP and ZTM products.

- UNP processes a simple SMF record that contains data describing activity on a 3705 processor.

- ZTM processes the TYPE50 SAS data set, created by MXG from the SMF type 50 record (VTAM Tuning Stats).

Note: ZTM is developed using the CA MICS-SAS Interface (MSI) which even further shortens the FDA development cycle.

7.2 Deciding What the Application Must Do

This section discusses product design. Your product design will assist you in expressing details about your product in a format that CA MICS can use.

The areas you need to address are the input source that needs to be managed, the kinds of information that need to be made available about the input source, and the files that would be useful in making the information available.

7.2.1 Input Source

The need to manage information that is not currently being managed is the reason for writing a field-developed application. The information that your FDA is going to manage is, in CA MICS terminology, an information area.

Having identified the information area, the source or sources of data should be apparent. Consider several criteria for the suitability of the data sources when choosing the source on which your FDA will rely.

- The data must be available on a regular basis (for example, daily) in a reliable fashion.

  The source data needs to be available when the CA MICS daily update cycle is scheduled. Interruption in this data will certainly mean an interruption in data appearing on the online database. Long term interruption may affect the accumulation of data for trend analysis. Even short term interruptions can affect other CA MICS products if the FDA is installed in a production database unit with other products.

- The data must be structured enough to allow its analysis by a SAS program.

- If the data is manually generated, there should be appropriate verification for correct contents.

- The data should describe some measurable quantity for which your site has performance or service objectives.

Documenting the data source or sources is useful when coding or analyzing the data produced.

Specifically, a description of the media and data sets from which the source data is received is useful when coding the MCG statements. Descriptions of the data elements in the records and their source and content are also helpful. These descriptions are used in formulating MCG code for output files, and help identify the type and length coding of data elements.

7.2.1.1 Input Source Example for UNP

```
Sample UNP data source documentation
------------------------------------

The Network Performance Analyzer was an IBM Field Developed
Program (FDP), IBM product numbers 5798-CZR and 5798-CZT.
Information about this FDP was found in the NPA Program
Description and Operations Manual, SB21-2479.

The UNP field-developed application processes NPA records
from the SMF log, SMF record type 38.  The records have a set
structure:

+----------------+
| record header  |
| (fixed format) |
+----------------+
| measurement    | There are 3 types of measurement sections:
| section        |  - NCP resource measurement
| (format varies |  - network component measurement
| with record    |  - monitor exception measurement
| type)          |
+----------------+
```

```
************************************************************
* R E C O R D   H E A D E R  (fixed format, in all records) *
************************************************************
LOGSIZE  0    PIB2.     LOG RDW
         2    PIB2.
LOGSYS   4    PIB1.     SYSTEM INDICATOR
LOGSTYPE 5    PIB1.     RECORD TYPE 38
LOGSTIME 6    PIB4.     SMF WRITE TIME OF DAY, IN SEC/100
LOGSDATE 10   PD4.      SMF WRITE DATE
LOGSYSID 14   $CHAR4.   SMF SYSTEM ID
LOGADDR  18   $CHAR2.   HEX RESOURCE NETWORK ADDRESS
LOGNAME  20   $CHAR8.   NETWORK COMPONENT ID
LOGTOD   28   PIB4.     SMF WRITE TIME IN SECONDS
LOGUNQKY 32   PIB2.     RESERVED
LOGTYPE  34   $CHAR1.   TYPE OF LOG RECORD
*                       01  START COLLECT
*                       04  INTERVAL RECORD
*                       08  TOTAL RECORD
*                       0C  MONITOR EXCEPTION RECORD
*                       0F  STOP COLLECT RECORD
LOGNTYPE 35   $CHAR1.   TYPE OF NETWORK RESOURCE
*                       80  3705/NCP RESOURCE
*                       40  LINK (SDLC) OR LINE (BSC)
*                       20  PHYSICAL UNIT (SDLC)
*                       20  CLUSTER (BSC)
*                       10  LOGICAL UNIT (SDLC)
*                       10  TERMINAL (BSC)
*                       08  COMPONENT
*                       04  SWITCHED RESOURCE (SDLC)
*                       02  NPA PSEUDO DEVICE
*                       01  SDLC RESOURCE (USED WITH ABOVE)
LOGINTER 36   PIB4.     EXPECTED INTERVAL TIME IN SECONDS
LOGFWTOD 40   PIB4.     TOD FORWARD RU WAS SENT TO NCP
LOGLSPDS 44   PIB2.     SENDING LINE SPEED IF LINE/LINK
LOGLSPDR 46   PIB2.     RECEIVING LINE SPEED IF LINE/LINK
LOGNCPNM 48   $CHAR8.   NCP NAME
LOGQUE   56   PIB1.     INTERVAL NUMBER FOR THIS COLLECTION
LOGRSVP2 57   PIB1.     RESERVED
LOGINTCT 58   PIB2.     NUMBER OF INTERVALS IF TOTALS RECORD
LOGTIME  60   PIB4.     ACTUAL COLLECTION TIME THIS RECORD
************************************************************
```

```
NCP Resource Record:

+---------------+
| record header |
| (fixed format)|
+---------------+
| measurement   |
| section --    |
| NCP resource  |
| data          |
+---------------+

************************************************************
* M E A S U R E M E N T   S E C T I O N -                  *
* NCP Resource Records                                     *
************************************************************
LOGFCC   64   PIB4.     FREE CYCLE TIME, 10THS OF SECONDS
LOGFBQ   68   PIB2.     FREE BUFFER QUEUE LENGTH
LOGFBH   70   PIB2.     FREE BUFFER HIGH WATER MARK
LOGFBL   72   PIB2.     FREE BUFFER LOW WATER MARK
LOGCIQ   74   PIB2.     NCP CHANNEL INTERMED. QUEUE LENGTH
LOGCHQ   76   PIB2.     NCP CHANNEL HOLD QUEUE LENGTH
LOGSLH   78   PIB2.     TIME IN SLOWDOWN, 10THS OF SECONDS
LOGSLM   80   PIB2.     NUMBER FREE BUFFERS AT SLOWDOWN ENTRY
LOGMXF   82   PIB2.     MAX AVAILABLE FREE BUFFERS
************************************************************

End of sample data source documentation
```

7.2.1.2 Input Source Example for ZTM

```
Sample ZTM data source documentation
------------------------------------

The ZTM product uses the CA MICS-SAS Interface to process the
TYPE50 SAS data set created by the MXG product from the SMF
Type 50 record.

It is recommended that a SAS PROC CONTENTS be executed
against any input SAS data set to be able to determine
required and sequence variables.  The following listing
shows the output from a PROC CONTENTS for the TYPE50
data set:
```

```
                          The SAS System              12:48 Thursday, December 4, 1997   2
```

```
                                CONTENTS PROCEDURE


Data Set Name: X.TYPE50                          Observations:         1604

Member Type:   DATA                              Variables:            32

Engine:        V609                              Indexes:              0

Created:       14:56 Friday, October 31, 1997    Observation Length:   144

Last Modified: 14:56 Friday, October 31, 1997    Deleted Observations: 0

Protection:                                      Compressed:           NO

Data Set Type:                                   Sorted:               NO

Label:         VTAM TUNING STATISTICS


                    -----Engine/Host Dependent Information-----


                       Data Set Page Size:       27648

                       Number of Data Set Pages: 9

                       File Format:              607

                       First Data Page:          1

                       Max Obs per Page:         191

                       Obs in First Data Page:   161

                       Physical Name:            MDI.MXG.PDB

                       Release Created:          6.090450

                       Release Last Modified:    6.090450

                       Created by:               MELMI06M

                       Last Modified by:         MELMI06M

                       Subextents:               2

                       Total Blocks Used:        9
```

```
                    -----Alphabetic List of Variables and Attributes-----
```

#	Variable	Type	Len	Pos	Format	Label
4	ATTCHTYP	Num	4	16	MG050AT.	ATTACHMENT*TYPE
5	BSIZE	Num	4	20		TRANSMIT*BLOCK*SIZE
6	BUFCAP	Num	4	24		STARTS FROM*DEST BUFF*CAPACITY LIMIT
7	CHMAX	Num	4	28		LARGE SIZE*CHAN PROGRAM*COUNT
8	CHNRM	Num	4	32		NORMAL SIZE*CHAN PROGRAM*COUNT
9	CONTROLR	Char	8	36		INTELLIGENT*CONTROLLER*NAME
10	CTCANAME	Char	8	44		CTCA NAME
11	DEV	Char	4	52		DEVICE*ADDRESS
12	DLRCOUNT	Num	4	56		DUMP-LOAD*RESTART*REQUESTS*(DLRMAX)
13	EXTNLEN	Num	4	60		CTCA EXTENSION LENGTH
14	INPIOS	Num	4	64		INBOUND*PIUS*(IPIU)
15	MPCNAME	Char	8	68		MPC*LINE*NAME
16	MXTRSIZE	Num	4	76		MAXIMUM*TRANSMIT*SIZE
17	NCPSLOWS	Num	4	80		TIMES NCP*ENTERED*SLOWDOWN*(SLODN)
18	NRATTNS	Num	4	84		ATTENTIONS*RECEIVED*(ATTN)
19	NREADATN	Num	4	88		READ*CHANNEL*ATTENTIONS*(RDATN)
20	NRREADS	Num	4	92		READ*CHANNEL*PROGRAMS*(CHRD)
21	NRWRITES	Num	4	96		WRITE*CHANNEL*PROGRAMS*(CHWR)
22	OUTPIOS	Num	4	100		OUTBOUND*PIUS*(OPIU)
23	OVBYTCNT	Num	4	104		OVERFLOW*BYTE*COUNT
24	PRI	Num	4	108		STARTS FROM*HI PRIORITY*REQUEST TRIGGER
25	QDPTH	Num	4	112		STARTS FROM*QUEUE DEPTH*LIMIT TRIGGER
26	RDBUFUSE	Num	4	116		READ*BUFFERS USED*(RDBUF)
27	SIOCOUNT	Num	4	120		SIOS*ISSUED
1	SMFTIME	Num	8	0	DATETIME21.2	SMF*RECORD*TIME STAMP

28	SUBCHPOL	Char	4	124		SUBCHANNEL*POLARITY*(READ OR WRIT)
2	SYSTEM	Char	4	8		SYSTEM*ID
29	TIMERS	Num	4	128		STARTS FROM*TIMER*TRIGGER
30	TRBYTCNT	Num	4	132		TRANSMIT*BYTE*COUNT
31	VERSN50	Num	4	136	MG050TY.	VTAM*RECORD*TYPE
32	WRBUFUSE	Num	4	140		WRITE*BUFFERS*USED
3	ZDATE	Num	4	12	DATE9.	ZEE DATE*ZEE OBS*WAS CREATED

7.2.2 Reports Needed

Identifying your reporting requirements is typically a matter of tracking the information requests you have received and then grouping the requests according to the data source that can be used to fulfill them.

Next, select a method to present the information--online inquiry or paper report; table, graph, pie chart, and so on are all possibilities with the CA MICS system.

Expressing your reporting requirements within the CA MICS framework, described below, enables you to use standard CA MICS operational facilities to generate the reports.

- CA MICS online inquiries are created using the CA MICS Information Center Facility (MICF). You can define an inquiry, control access to it, and include or exclude it from automatic execution. See the MICF User Guide and MICF Reference Guide for more information about using MICF.

- CA MICS Exception Reports provide the detail needed to specifically identify and correlate the problems generated in the measured subsystem among all data processing subsystems managed in your CA MICS system. They are discussed below.

- CA MICS Management Objective Reports provide clear, graphic illustrations of the measured subsystem's performance against predetermined management objectives. They are discussed below.

7.2.2.1 Exception Reports

Exception reporting provides a diagnostic filter that enables you to focus on specific problems or potential problem areas. The exception reporting concept holds that, using trends from past performance, you can predict when a potential problem will occur.

CA MICS exception reporting is described in chapter 2 of the CA MICS Standard Reports Guide.

In designing your field-developed application, identify notable exceptions using these guidelines:

- An exception is the occurrence of a processing anomaly, including missing an objective.

- The exception must be able to be detected by the data source and the logic to detect the exception must be able to be expressed in SAS system code.

If your FDA is measuring the subsystem in throughput rates, like units of things processed per minute or hour, objectives are easy to set. In the UNP example, the number of messages processed per minute may be a good throughput measure on which to base objectives. An exception could be coded for failing to meet a minimum throughput objective.

7.2.2.2 Management Objective Reports

Management objective reports provide a concise, graphic representation of the installation's processing objectives and how well they have been met.

Reports are produced by standard CA MICS operational jobs. They are produced by information area within computer system (SYSID). In addition, you can produce reports that aggregate systems.

Management objective reports show an installation's progress against a management objective on a one-page bar chart quantifying performance against management's pre-determined objective.

In general, a management objective reporting items take two forms:

- measures, over time, of the efficiency of the subsystem being measured (throughput objectives)
- conditions that contribute to improving or degrading throughput objectives

If your FDA is measuring the subsystem in throughput rates, like units of things processed per minute or hour, objectives are easy to set. In the UNP example, the number of messages processed per minute may be a good throughput measure on which to base objectives.

In the UNP example, there is a relationship between the percent of time the 3705 is busy and throughput. Therefore, also consider setting management objectives for items that contribute to the improvement or degradation of throughput.

7.2.3 Files and Elements Needed

The nature of the data source and the structure of its records weigh heavily in the design of CA MICS files for the FDA and its information areas.

Remember that CA MICS products are divided into information areas, and information areas contain files, which are groups of related data elements. In addition, the file data can be maintained online or offline.

A detailed file definition specifies the processes used to construct the file. At its conclusion, you should know which data elements in the input data records are used to build the CA MICS file observations and what validity checks need to be performed.

In deciding how many files to create in an information area, consider the structure of the input data source, its availability, and your intended use of the data.

The grouping of files and information areas is easier to understand by example. The UNP input data source had three record subtypes, as shown in this structure diagram:

```
+----------------+
| record header  |
| (fixed format) |
+----------------+
| measurement    | There are 3 types of measurement sections:
| section        |  - NCP resource measurement
| (format varies |  - network component measurement
| with record    |  - monitor exception measurement
| type)          |
+----------------+
```

The implementation of the UNP product in this example uses one information area with three files. Each file corresponds to one of the subtypes of NPA input data:

```
    Component:----------- NPA Network Activity
       Information area:-    NPA Network Activity
          File:----------       NCP Activity
          File:----------       Network Component Activity
          File:----------       NPA Monitor Exceptions
```

However, several alternative arrangements are possible. If there is a thorough monitoring of all network components through NPA and network anomalies are explained through some manual mechanism suitable for input as an automated incident tracking tool, this file arrangement might be more suitable:

```
    Component:----------- NPA Network Activity
       Information area:-    NCP Processor Activity
          File:----------       NCP Activity
       Information area:-    Network Incidents
          File:----------       NPA Monitor Exceptions
          File:----------       Operational Incidents
       Information area:-    NPA Network Component Activity
          File:----------       Line/Link Component Activity
          File:----------       PU/Cluster Activity
          File:----------       LU/Terminal Activity
```

The processing needed to create files, once you have decided how many files you need, is done in two stages:

- First, the content of the input data source has to be translated into the files that are kept on the CA MICS database.

- Second, CA MICS database files can reside in the online database alone, or on both the online and offline databases.

Addressing the first issue, decide if each data source record produces an observation in your FDA's files. For example, the UNP product has one information area and three files, where one record subtype corresponds to a single file. Does each input observation of the NPA data create an output observation on the CA MICS database (output) file?

The NPA record subtypes produce data at different levels of summarization. That is, NPA produces records that represent the detailed activity for some particular resource over an interval of time. NPA also produces up to four levels of summarization for the SAME resource.

For example, NPA monitors an NCP resource every minute and writes the following records on its SMF log:

```
NCP record, 1 min. of detailed activity
NCP record, 1 min. of detailed activity
NCP record, 1 min. of detailed activity
NCP record, 1 min. of detailed activity
NCP record, 4 min. of summarized activity
NCP record, 1 min. of detailed activity
NCP record, 1 min. of detailed activity
NCP record, 1 min. of detailed activity
NCP record, 1 min. of detailed activity
NCP record, 4 min. of summarized activity
NCP record, 8 min. of summarized activity
```

The summarized records are not needed to calculate an accurate picture of activity measured by NPA. In this case, the summarized input records should be dropped.

Use these steps to design file processing:

1. Define the structure of the output files, including the information areas and files within each information area. Assign names to each, using these guidelines:

   - A three-character information area identifier must be assigned to each information area. The identifier must be unique among all information areas in CA MICS. We strongly recommend that the first character be either a U or Z because CA MICS has reserved those letters for field-developed applications. The information area cannot be MSI or MTI.

     If your FDA has one information area, consider using the product identifier as the information area identifier. For example, UNPUNP would identify a user-written application for network performance data (UNP) that has an information area for network performance.

   - A three-character file identifier must be assigned to each file. The identifier must be unique among all files in CA MICS. We strongly recommend that the first character be either a U or Z because CA MICS has reserved those letters for field-developed applications. File identifiers cannot match product or information area identifiers. The file name cannot be MSI or MTI.

2. Identify and document the input records or SAS data sets you will use. Identify input records to be discard and outline an algorithm for ignoring them.

3. Map input data to output file data elements by defining which input record types apply to creating the output file data elements.

The map you produce may yield a design document appropriate to your site's standards. Examples of such documents are flow charts, IPO/HIPO diagrams, and Chapin charts.

4. Define which files will be maintained in the online and offline databases. Consider the following in determining each file's eligibility for archiving:

   - If the information in the file describes some detailed activity that represents resource consumption or a potential security problem that could be attributed to an identified user, consider maintaining audit archive offline files.

   - If the file exists in the WEEKS timespan in the online database and the data in the file would be useful for long term (trend) analysis, consider maintaining weekly history offline files. The data in weekly history archives is summarized by smaller time intervals, so its analysis yields more detailed results than monthly history archives.

   - If the file exists in the MONTHS timespan in the online database and the data in the file would be useful for long term (trend) analysis, consider maintaining monthly history archives. The data in monthly history archives is summarized by larger time intervals, so its analysis provides longer term information than that provided by weekly history archives.

5. Define data element content.

   Document the data elements in your files using the format used for the CA MICS Data Dictionary.

   While you do not need to key this documentation into machine readable form (the first pass of the MCG will create skeleton members in the sharedprefix.MICS.INC.TEXT library for all data elements in all files in the component), the information you need to record includes:

- the element's identifier and descriptive name
- information about the element's content
- the data element's coding description, including how it is accumulated, what format it takes, and what unit of measure is used (for example, a maximum time value whose measure is a SAS time value)
- references to related data elements, if any
- information on how the element's value is derived, such as its input source record and field or the calculation used to derive the value at the lowest supported timespan
- information about the element's valid values, which could be a specific list (such as department names) or a list of anticipated norms (such as a percentage that ranges from zero to one hundred)
- usage notes that describe why the element is created, how it should be used, or how it should NOT be used

7.3 Writing the Code

This section discusses writing code for your FDA. It will use the product design information prepared in the last section as well as the shell generation facilities provided by the CA MICS Component Generator (MCG).

Code creation starts with defining the product using the MCG statements--you create an fdaGENIN module, where fda is your product's identifier and begins with either a U or Z. Then you generate the shells for the input format routine (DYfdaFMT) and reports (DYfdaEXC, fdaEVA, and DYfdaMBO).

7.3.1 Define the Application (fdaGENIN)

Section 4.2.4 in this guide describes the component generator statements in depth. This section assumes that you are familiar with the material in that section. Sections 4.3.13.1 and 4.3.13.2 in the PIOM contain additional information about the COMPJOB statement.

The generation input statements for every product that CA MICS will process must be in the generation library (sharedprefix.MICS.GENLIB). Distribution material for CA MICS includes one member in this library for each CA MICS product licensed. The names of members on the library have the form "cccGENIN", where ccc is the component identifier.

To ensure uniqueness, field-developed applications MUST use an identifier that begins with either a U or Z.

7.3.1.1 Define the Application (UNPGENIN)

```
The statements listed below define a field-developed Network
Performance application (UNP) with one file, the Network
Control Program File (UNA).

These statements are presented here to illustrate
field-developed application definition.  This description of
the UNP product does not look like the definition shown in
Section 4.2.6 of this guide because the names of the product,
information area, files, and data elements were changed to
simulate the coding techniques you would use in coding your
own network performance field-developed application.


************************************************************
*       E X A M P L E   U N P   P R O D U C T              *
************************************************************

Hypothetical contents of sharedprefix.MICS.GENLIB(UNPGENIN):

*
* UNP component generator statements
*
* The following GEN statement causes the MCG to generate
* documentation shells and file description members.
*
GEN GENSHELL
*
* If an OPTION statement existed, it would be placed here.
*
* The COMP statement identifies the UNP product.  This
* product is step 150 of the CA MICS operational jobs, so the
* UNP update step of the CA MICS DAILY job, for example,
* would be step DAY150.  The input source for reading UNP
* data is a VBS data set of default blocksize.  The
* accounting component will not be active for UNP (because
* there is no user-related identifier on any NPA data).
*
COMP UNP 150 32000 VBS NOACCT User NPA Component
COMPOPT DEFAULT DEFAULT
COMPJOB DWMY EDWM AWM
INPUTSOURCE SMF 37
DBMDATA1
*
* The following elements are common to UNP files.
*
TYPE XR  3 . 3 . 3 .
NAME UNPNRRLS  Receiving Line Speed (line/link)
```

```
NAME UNPNRSLS  Sending Line Speed (line/link)
TYPE XA  5  TIME11.2  6  TIME15.2  6  TIME15.2
NAME UNPTMAOT  Actual Observation Time
TYPE XR  4  TIME8.  4  TIME8.  4  TIME8.
NAME UNPTSEIS  Expected Interval Time
NAME UNPTSFRU  Foreward RU Send Time
NAME UNPTSLOG  Time Record Logged To SMF
*
* UNP's information area is also called UNP.
*
AREA UNP User NPA Activity Information Area
*
* This FILE statement describes the  UNP area's User NCP
* Activity File (UNA).  It is a standard CA MICS file (the
* type 00 after UNA), is not considered for introduction to
* the YEARS time-span, and has no archive audit application.
*
FILE UNA 00 1 Y Y Y Y N N Y Y User NCP Activity File
FOPT DEFAULT DEFAULT STD
CYCLES 10 33 26 06 00 00 054 024
COMPRESS Y Y Y Y N
*
*  Include common elements.
*
NAME UNPNRRLS     99  0 N N N N
NAME UNPNRSLS     99  0 N N N N
*
* The next four data elements are retained numeric, saved on
* the CA MICS database in the UNPUNA files with a length of 3
* bytes.
*
TYPE R 3 . 3 . 3 .
NAME UNANRFBQ     00  0 0 0 0 0 Free Buffer Queue Length
NAME UNANRHQL     00  0 0 0 0 0 NCP Channel Hold Queue Length
NAME UNANRIQL     00  0 0 0 0 0 NCP Channel Intrmd Queue Len.
NAME UNANRSFB     00  0 0 0 0 0 Free Buffer Count At Slowdown
*
* The next data elements are accumulated numeric elements,
* and will be summed in the generated summarization process.
* Elements kept in time-spans higher than DETAIL should have
* a slightly larger saved  data element length to ensure that
* they contain all the precision  pertinent to the data
* element.
*
* Note that a numeric time data element of length 6 can
* contain enough precision to carry a year of elapsed time.
* The format length also changes with the increased length.
*
TYPE A 5 TIME11.2 6 TIME15.2 6 TIME15.2
```

```
NAME UNATMFCT     00  0 0 0 0 0 Free Cycle Time
NAME UNATMTIS     00  0 0 0 0 0 Time In Slowdown
TYPE M 3 . 3 . 3 .
NAME UNAMXANP     00  0 0 0 0 0 Max Available NCP Buffers
NAME UNAMXFBH     00  0 0 0 0 0 Free Buffer High Water Mark
TYPE N 3 . 3 . 3 .
NAME UNAMNFBL     00  0 0 0 0 0 Free Buffer Low Water Mark
*
* The following data elements are common to files in more
* than one CA MICS product. The type and label descriptions
* for those data elements are contained on
* sharedprefix.MICS.GENLIB($DEGENIN).  Common data elements
* are generically named, according to their purpose or
* content, such as DURATION.  It is not necessary to code
* labels for common data elements.  A period (.) is coded to
* note the absence of a label.
*
NAME RESNAME      99  0 0 0 0 0 .
*
* DAY, below, appears in the DETAIL and DAYS time-spans, but
* is dropped from the file in WEEKS and MONTHS.  If this file
* was active in the YEARS time-span (which it is not), DAY
* would have been dropped there, too.  Also, DAY is a part of
* the key structure for the UNPUNA file.  In the DETAIL and
* DAYS time-spans, DAY is the 5th data element in the key.
*
NAME DAY          99  5 5 N N N .
NAME HOUR         99  6 6 6 N N .
NAME WEEK         99  0 0 4 N N .
NAME YEAR         99  3 3 3 3 3 .
NAME MONTH        99  4 4 N 4 N .
NAME INTERVLS     99  0 0 0 0 0 .
NAME DURATION     99  0 0 0 0 0 .
NAME STARTTS      99  0 0 0 0 0 .
NAME ENDTS        99  7 0 0 0 0 .
NAME ZONE         99  0 0 5 5 4 .
NAME DAYNAME      99  0 0 N N N .
NAME SYSID        99  1 1 1 1 1 .
NAME NCPNAME      99  2 2 2 2 2 .
NAME NETNAME      99  0 0 0 0 0 .
*

End of example for UNPGENIN
```

7.3.1.2 Define the Application (ZTMGENIN)

```
The statements listed below define an MSI field-developed
application for VTAM Tuning Statistics (ZTM), derived from
the MXG file TYPE50, with one file VTAM Tuning Stats (ZTS).

These statements are presented here to illustrate
field-developed application definitions.

Please refer to Section 4.2.4.3 within this guide for
information on the required order of these statements.


*                                                                MICS6
* READ DATA FROM MXG FILES created from the SMF Type 50          00001000
*                                                                00003000
************************************************************
*       E X A M P L E   Z T M   P R O D U C T         *
************************************************************

Hypothetical contents of sharedprefix.MICS.GENLIB(ZTMGENIN):

*
* ZTM component generator statements
*
* The following GEN statement causes the MCG to generate
* documentation shells and file description members.
*
GEN GENSHELL                                                     00004000
*
* If an OPTION statement existed, it would be placed here.
*
* The COMP statement identifies the ZTM product.  This
* product is step 198 of the CA MICS operational jobs, so the
* UNP update step of the CA MICS DAILY job, for example,
* would be step DAY198.  The input source for reading ZTM
* data is a VBS data set of default blocksize.  The
* accounting component will not be active for ZTM (because
* there is no user-related identifier on any VTAM Tuning
* data).
*
```

```
COMP ZTM 198 . .  NOACCT VTAM Tuning STATS                          00005000
COMPOPT DEFAULT DEFAULT                                             00006000
*                                                                   00007000
* In order to use MSI features, you must specify MSI as the
* input source.
*
INPUTSOURCE MSI                                                     00009000
COMPJOB DWMY EDWM AWM                                               00010000
DBMDATA1 15 100 50 30000 0 0                                        00011000
OPTION                                                              00012000
*                                                                   00013000
* Special options for MSI are set here. Note that options
* specified here are global and will apply to all files,
* and that MSIOPTS can be used after the INPUTSAS statement
* to specify or override options.
*
* In this example, the following options are used:
* SPLITCHAR translates '*' to a space (the * is used in MXG
* as a new line character when printing labels).
*
* FILTER will automatically ignore variables that conflict
* with existing CA MICS variables for specific data sources
*

MSIOPTS SPLITCHAR *
MSIOPTS FILTER MXG
*                                                                   00013000
* ZTM's information area is also called ZTM.
*
AREA ZTM  VTAM Tuning STATS INFO AREA                               00014000
*                                                                   00015000
FILE ZTS   00  1  Y Y Y Y Y N Y Y Tuning STATS                      00016000
FOPT DEFAULT DEFAULT STD                                            00017000
CYCLES 07 33 09 06 01 00 053 024                                    00018000
COMPRESS Y Y Y Y Y                                                  00019000
*
```

```
                                                                  00020000
* The INPUTSAS statement is used to specify the SAS data
* set that will be used as input in the DAYxxx step.
*
INPUTSAS PDB.TYPE50                                                 00021000
*                                                                   00020000
* Each file that uses INPUTSAS or INPUTMODEL is designated
* as an MSI file and MUST define ENDTS, STARTTS, and
* ORGSYSID.  Failing to specify these data elements will
* result in an error.
*
ENDTS    SMFTIME                                                    00030000
STARTTS  SMFTIME                                                    00031000
ORGSYSID SYSTEM
                                                 00032000
*
* The following statements change a data element's type.
* Here BSIZE is changed to RETAIN, and MXTRSIZE is
* changed to MAXIMUM.
*
RETAIN BSIZE
MAXIMUM MXTRSIZE

*
* The KEEP/DROP statements allow for elements to be removed
* from the file.
*
DROP VERSN50
*
* The SEQUENCE statement (below) sets the user sort sequence
* for the ZTS file. This sequence will apply to all
* timespans.  Note that it is unnecessary to specify things
* like MONTH, DAY, YEAR, etc.
*
SEQUENCE SYSID CONTROLR CTCANAME MPCNAME
*
* MSI allows modifications of the sort sequences to specific
* timespans.  Here the APPENDSEQUENCE statement adds the
* data element DEV to the sort sequence after MPCNAME, only
* for the detail timespan.
*
DETAIL
APPENDSEQUENCE DEV

  End of example for ZTMGENIN
```

7.3.1.3 Generate Shells

Once the shell is coded, check the syntax of the fdaGENIN member as follows:

- First, ensure that the GEN statement in sharedprefix.MICS.GENLIB(fdaGENIN) reads as follows:

  ```
  GEN SYNTAX
  ```

- Then, code the following JCL in sharedprefix.MICS.CNTL(fdaCGEN), supplying your own job statement parameters:

  ```
  //fdaCGEN JOB  ....
  //fdaCGEN EXEC MICSGU,COMPT=fda
  ```

  and submit the job. Review the listing, make any corrections to the fdaGENIN member as indicated in the MICSLOG output, and resubmit the syntax check job as many times as necessary to produce error-free output.

When the syntax check job runs without error, you are ready to generate the documentation shells and file description members. To generate them,

- Ensure the GEN statement in sharedprefix.MICS.GENLIB(fdaGENIN) reads as follows:

  ```
  GEN GENSHELL
  ```

- If you are not building an MSI component, modify the fdaCGEN JCL created earlier. Add the name of your complex-level hold user source library following the USOURCE= designator.

  ```
  //fdaCGEN EXEC MICSGU,COMPT=fda,
  //  USOURCE='sharedprefix.MICS.HOLD.USER.SOURCE'
  ```

- Submit the fdaCGEN job. The modules generated by the shell generation are listed in Figure 7-1.

  Note: The following modules are not generated by GENSHELL. If you want to exercise CA MICS exception processing with your field-developed application, code the following:

  - DYfdaEXC (Daily Exception Analyzer Routine) in sharedprefix.MICS.SOURCE
  - DYfdaEXC (Daily Exception Analyzer Routine) in sharedprefix.MICS.HOLD.USER.SOURCE
  - fdaEVA (Exception Value Analyzer Routine) in sharedprefix.MICS.SOURCE

Generated Type	Generated Module Name	Generated Module Title	MCG Program Name	Data Set Into Which Module Is Generated	Notes
SAS Code	$cccMSTR	Master Options MACROs	GENSASMI	sharedprefix.MICS.HOLD.USER.SOURCE	
SAS Code	$cccMSTR	Master Options MACROs	GENSASMI	sharedprefix.MICS.SOURCE	
SAS Code	$cccEXIT	User Exit MACROs	GENSASEX	sharedprefix.MICS.HOLD.USER.SOURCE	
SAS Code	$cccEXIT	User Exit MACROs	GENSASEX	sharedprefix.MICS.SOURCE	
SAS Code	#ffftSUM	Special Summarization Routines	GENSASEX	sharedprefix.MICS.SOURCE	see Note 1
SAS Code	DYcccFMT	Daily Input Format Routine	GENSASFM	sharedprefix.MICS.SOURCE	
SAS Code	DYcccMBO	Daily MBO Report Routine	GENSASMI	sharedprefix.MICS.HOLD.USER.SOURCE	
SAS Code	DYcccMBO	Daily MBO Report Routine	GENSASMI	sharedprefix.MICS.SOURCE	
SAS Code	WKcccMBO	Weekly MBO Report Routine	GENSASMI	sharedprefix.MICS.HOLD.USER.SOURCE	
SAS Code	WKcccMBO	Weekly MBO Report Routine	GENSASMI	sharedprefix.MICS.SOURCE	
SAS Code	MNcccMBO	Monthly MBO Report Routine	GENSASMI	sharedprefix.MICS.HOLD.USER.SOURCE	
SAS Code	MNcccMBO	Monthly MBO Report Routine	GENSASMI	sharedprefix.MICS.SOURCE	

```
|SAS Code | DAYsss   | %INCLUDE List for Daily DB Update       | GENSASIN | sharedprefix.MICS.INCLLIB         |          |
|SAS Code | WEEKsss  | %INCLUDE List for Weekly DB Update      | GENSASIN | sharedprefix.MICS.INCLLIB         |          |
|SAS Code | MONTHsss | %INCLUDE List for Monthly DB Update     | GENSASIN | sharedprefix.MICS.INCLLIB         |          |
|SAS Code | YEARsss  | %INCLUDE List for Yearly DB Update      | GENSASIN | sharedprefix.MICS.INCLLIB         |          |
|SAS Code | #cccMACS | %INCLUDE List for Summarization         | GENSASIN | sharedprefix.MICS.INCLLIB         |          |
|SAS Code | cccGTEST | %INCLUDE List for Testing Online DB     | GENSASTS | sharedprefix.MICS.INCLLIB         |          |
|SAS Code | cccATEST | %INCLUDE List for Testing Archive DB    | GENSASTS | sharedprefix.MICS.INCLLIB         |          |
|SAS Code | cccDATAS | PROC DATASETS for DB Checking           | GENSASTS | sharedprefix.MICS.SOURCE          |          |
|SAS Code | #DBMccc  | Database Space Modelling Logic          | GENSASIM | sharedprefix.MICS.SOURCE          |          |
|         |          |                                         |          |                                   |          |
|JCL/GCL  | DYccccss | Step JCL for Daily Job                  | GENPROST | sharedprefix.MICS.PROTOLIB        |          |
|JCL/GCL  | WKccccss | Step JCL for Weekly Job                 | GENPROST | sharedprefix.MICS.PROTOLIB        |          |
|JCL/GCL  | MNccccss | Step JCL for Monthly Job                | GENPROST | sharedprefix.MICS.PROTOLIB        |          |
|JCL/GCL  | YRccccss | Step JCL for Yearly Job                 | GENPROST | sharedprefix.MICS.PROTOLIB        |          |
|         |          |                                         |          |                                   |          |
|Text     | PICccc   | Data Dictionary PIC List                | GENTXTDD | sharedprefix.MICS.INC.TEXT        |          |
|Text     | eeeeeeee | Data Dictionary Shells                  | GENTXTDD | sharedprefix.MICS.INC.TEXT        |          |
|         |          |                                         |          |                                   |          |
+---------+----------+-----------------------------------------+----------+-----------------------------------+----------+
```

Note 1: Module #ffftSUM is generated when "E" is specified on the FILE statement in cccGENIN for the timespan "t".

Note 2: You can copy the dictionary shells generated by the GENSHELL from sharedprefix.MICS.INC.TEXT to
sharedprefix.MICS.DIC.TEXT but be sure the members being copied do not already exist in
sharedprefix.MICS.DIC.TEXT.

Figure 7-1. Shell Modules Generated by GENSHELL Process

7.3.2 Define the Input Format Routine (DYfdaFMT)

The input format routine is the SAS logic that translates the input source into CA MICS data elements. The following sections show example input format routines for the UNP and ZTM products. The ZTM product uses the CA MICS-SAS Interface that greatly reduces the amount of code that needs to be written.

7.3.2.1 Define a non-MSI Input Format Routine (DYfdafmt)

The non-MSI input format routine is generated during the GEN GENSHELL process executed in conjunction with testing the fdaGENIN.

There are several methods used by CA MICS standard components for structuring input data format routines. A sample input format routine shell generated from the UNPGENIN module follows. Details about writing your specific FDA's input format code follows this sample.

```
/*-------------------------------------------------------------------*/
/* Copyright (C) 2010 CA.  All rights reserved.                      */
/*-------------------------------------------------------------------*/
/* CHANGE | PROBLEM|MMYY|VENDOR CHANGE DESCRIPTION                   */
/*--------+--------+----+----------------------------------------------*/
/*-------------------------------------------------------------------*/
/* CHANGE |MMYY|USER CHANGE DESCRIPTION                               */
/*--------+----+-----------------------------------------------------*/
/*-------------------------------------------------------------------*/

/* ******************************************************************
* COMPONENT:    UNP USER COMPONENT                                  *
* MODULE GENERATED = DYUNPFMT                                       *
* DATE GENERATED Thu, Apr 23, YYYY      TIME GENERATED  6:29:59     *
* CA MICS COMPONENT GENERATOR - BAS6000                             *
*******************************************************************/
/* ******************************************************************
* THIS MEMBER PROCESSES THE RAW INPUT DATA USED                     *
* TO UPDATE THE CA MICS DATABASE. THE INPUT                         *
* DATA IS READ, VERIFIED, INTERPRETED, AND                          *
* ORGANIZED TO BUILD THE RECORDS USED TO UPDATE                     *
* THE CA MICS DATABASE DETAIL (OR DAYS) TIMESPAN.                   *
*                                                                   *
* ADDITIONALLY, THIS MEMBER ENSURES THAT THE NECESSARY              *
* AUDIT TRAILS ARE BUILT, DUPLICATE DATA ELIMINATION                *
* IS PERFORMED, AND COMPONENT CHECKPOINT UPDATING                   *
* IS ACCOMPLISHED.                                                  *
*******************************************************************/
TITLE "CA MICS UNP COMPONENT INPUT FORMAT ROUTINE";
DATA

     WORK.UNPUNA00   (KEEP=%UNAKEEP(TS=DETAIL ))
     &ADMX..CKPTDATA (LABEL='INPUT STATISTICS FOR CURRENT DATA'
                      KEEP=%WHLKEEP(TS=DETAIL))
     %USRFMT1
     ;;;;
  /*EXIT RETURN INDICATOR SET                                       */
```

```
  %EXITCK;

  /* COMMON LENGTH/FORMAT AND LABELS DEFINITION                    */
  %UNPCLEN(TS=DETAIL);

  /* FIELD LENGTH/FORMAT/INFORMAT DEFS BY FILE                     */

  /* USER NCP ACTIVITY FILE                  */
  %UNALEN(TS=DETAIL );
  %UNAFMT(TS=DETAIL );
  %UNALBL;

  LENGTH BLANKS $200.;
  %LET BLANKS = 1;
  LENGTH ORGSYSID $4 COMPT $8;
  RETAIN  BLANKS ' ' ZERO 0 ONE 1 HUN 100
  COMPT 'UNP     ' ROUTINE 'DYUNPFMT' LINKSW 0 ;

  /* Invoke CHECKPOINT retrieval routine                           */
  %CKRTN1;

  /* INPUT FILE DEFINITION                                         */
  /* The DD name specified must be INPUTSAS when fdaGENIN specifies */
  /* INPUTSOURCE SMF, and INPUTfda when fdaGENIN specifies         */
  /* INPUTSOURCE NONSMF.  In this FDA's prefix.MICS.PARMS library,  */
  /* a corresponding member named INPUTSAS or INPUTfda must be      */
  /* supplied to specify this DD name for JCL generation.          */
  INFILE INPUTSAS STOPOVER EOF=EOF_1ST LENGTH=RECLEN COLUMN=COL ;
  INPUT @;
  INSMF+1;
  COMPT='UNP';

  /* USER INPUT STATEMENT DEFINITION EXIT                          */
  /* INPUT ORGSYSID AND ENDTS, FOR THE PROPER OPERATION */
  /* OF CKRTN2 - DATA RANGE CHECKING OF INPUT.          */
  %USRFMT2;

/* COMMON RECORD PROCESS ROUTINE                                   */
COMPROC: ;
  _USRSEL ;  /* INVOKE USER SELECT EXIT ROUTINE IF CODED */

  /***  PROCESS THE RECORD  ***/
  /***  PROCESS THE RECORD  ***/
    ENDTS=TODAY();
    UNAAGTCT=8;
    OUTPUT UNPUNA00;
  /***  PROCESS THE RECORD  ***/
  /***  PROCESS THE RECORD  ***/
```

```
  /* Invoke CHECKPOINT routine to verify the data (ENDTS) as to one */
  /* of the following conditions:                                   */
  /*   ENDTS within CHECKPOINT range - drop duplicates              */
  /*   ENDTS below CHECKPOINT range  - drop prior data              */
  /*   ENDTS above today's date - drop future data                  */

  %CKRTN2;

  /* Invoke common fields derivation routine.                       */
  /* YEAR, MONTH, WEEK, ZONE, DAY, DAYNAME, and HOUR                */
  /* are built from the ENDTS field.                                */

  %CMNFLDS;

  /* Invoke SYSID lookup routine to build logical SYSID, MIPS rate, */
  /* CPUMODEL, SCP level, and RMF CPU conversion factor.            */

  %SYSID;

  /* Input statements are defined for each record to be used in the */
  /* update procedure along with any other edit or manipulative     */
  /* testing of the data.  The RETURN statement must not be used.   */

  %USRFMT3;

RETURN;

/* ********************************************************************
* AN OUTPUT ROUTINE IS ESTABLISHED FOR EACH DETAIL                    *
* OR DAYS FILE TO BE PROCESSED.                                       *
*********************************************************************/

/* OUTPUT ROUTINE FOR USER NCP ACTIVITY FILE                          */
OUTUNA:
  /* USER OUTPUT LOGIC FOR USER NPC ACTIVITY FILE                     */
  /* PERFORM SFD TEST AND OUTPUT FILE                                 */
  IF &UNASFD THEN DO;
    %UNADERV;
    /* USER COSTING AND MANIPULATION EXIT */
    %FXIT(NAME=USRSUNA);
    OUTPUT WORK.UNPUNA00;
  END;
RETURN;

/* OPTION DELETE ROUTINE PROVIDES METHOD TO                           */
/* ADJUST RECORD COUNTS FOR USER DELETED                              */
/* ACTION                                                             */
OPTN_DEL:
  RECKPT=RECKPT-1;
```

```
  RECODLT+1;
RETURN;
/* USER END OF FILE ROUTINE EOF_1ST                                    */
/* USRFMT4 IS FOR USER END OF FILE CODE                                */
EOF_1ST:;
  %USRFMT4;
  GOTO EOF_RTN;
  /* EOF_RTN IS DEFINED IN CKRTN3.                                     */
  %CKRTN3;
RUN;

/* INVOKE CHECKPOINT ROUTINE TO UPDATE THE                             */
/* CHECKPOINT TIME-RANGES FOR THE SYSIDS AND                           */
/* COMPONENT UPDATED, AND RECORD THE DATA                              */
/* REQUIRED FOR UPDATING THE INPUT HISTORY                             */
/* LOG ON THIS COMPONENT'S UPDATE ACTIVITY.                            */
%CKRTN4;
RUN;

/* USER POST-FORMAT AND PRE-SORT EXIT                                  */
%USRFMT5;

/* ********************************************************************
* EACH OF THE FILES ARE SORTED TO ORDER THE DATA FOR                   *
* ITS NEXT USE IN THE UPDATE PROCESS AND                              *
* TO AUTOMATICALLY ELIMINATE DUPLICATE DATA                           *
* AS DEFINED BY THE SEQUENCE MACROS.  FINALLY, THE                    *
* IN-PROCESS FILE IN THE DETAIL TIMESPAN                              *
* IS BUILT AND THE TEMPORARY WORK FILE IS                             *
* DELETED IF DETAIL IS ACTIVE.                                        *
**********************************************************************/

/* USER NCP ACTIVITY FILE                                              */
PROC SORT NODUP DATA=WORK.UNPUNA00
  OUT=&UNPX..UNPUNA00(%UNAFILE(TS=DETAIL,OP=FILEOPTS));
  BY %UNASEQ(TS=DETAIL ); RUN;
PROC DATASETS DDNAME=WORK NOLIST NOWARN;
  DELETE UNPUNA00;
RUN;
QUIT;

/*USER END OF FORMAT ROUTINE EXIT                                      */
%USRFMT6;

RUN;
```

```
The points where your FDA's input format code executes are
defined by SAS macro references.  The macros themselves,
named USRFMTn, reside in member
sharedprefix.MICS.SOURCE($fdaEXIT).  You put your code in
this member.  The macros perform the following tasks:

    USRFMT1 - work file definition code
    USRFMT2 - input statement definition code
    USRFMT3 - input/output processing code
    USRFMT4 - input end-of-file code
    USRFMT5 - post-format, pre-sort code
    USRFMT6 - end-of-format routine code

The first four macros are referenced in the main SAS data
step.  The input format routine begins with a DATA step that
inputs source data and outputs observations to work file
versions of DETAIL timespan files you defined in
sharedprefix.MICS.GENLIB(fdaGENIN).

After the main DATA step, the USRFMT5 macro is referenced.
It is followed by code that sorts the work file versions of
your FDA's DETAIL timespan files into the order needed and
writes the DETAIL database.  After the sort completes, the
USRFMT6 macro is referenced.

For more information about what should be coded in each
macro, see the following sections:

    1 - Define Non-database Files (USRFMT1)
    2 - Define Input Record Validation (USRFMT2)
    3 - Define Input/Output Processing (USRFMT3)
    4 - Define Additional Process Code (USRFMT4-6)
    5 - Validate Your Code
```

7.3.2.1.1 - Define Non-database Files (USRFMT1)

The USRFMTn macros are executed as part of the main SAS DATA step in DYfdaFMT.

Use the USRFMT1 macro to add non-database oriented files to the SAS work files that will be written by the input format routine. (One work file is created for each database file that will exist in the DETAIL timespan.)

If you need an additional work file, you must code its file name and the KEEP list in USRFMT1.

Example:

```
%MACRO USRFMT1;
  WORK.MYFILE (KEEP ENDTS STARTTS INTERVLS)
%MEND USRFMT1;
```

resolves to

```
DATA
 WORK.UNPUNA00  (KEEP=%UNAKEEP (TS=DETAIL))
 &ADMX..CKPTDATA (LABEL='INPUT STATISTICS FOR CURRENT DATA'
                  KEEP=%WHLKEEP (TS=DETAIL))
 WORK.MYFILE (KEEP=ENDTS STARTTS INTERVLS);
```

and enables you to generate observations to WORK.MYFILE whenever you need to.

7.3.2.1.2 Define Input Record Validation (USRFMT2)

The USRFMT2 macro is used to do initial validation of each input record immediately after it is read. In addition, some essential data elements must be populated at this point.

Each record entering your USRFMT2 routine must experience one of the following outcomes:

- It is completely rejected by incrementing the WEIRDREC counter and branching as explained below.
- It is found to be unwanted, is counted as such, and is dropped.
- It is accepted, validated, and so on; and reaches the end of your USRFMT2 routine. This allows the code that contains the _USRSEL exit point, common field determination, and observation checkpointing to be executed next. This must occur for every observation of every FDA file created.

Consider the following items and code your routine appropriately:

- Occasionally, an input data stream contains troublesome records. One such event may involve completely unrecognizable data. These records must be handled by incrementing the MICS-supplied counter named WEIRDREC and then causing the next input record to be read:

  ```
  WEIRDREC+1;
  GOTO EOF_CK;
  ```

- Another occasional occurrence is that the total length of the incoming record is not proper for the INPUT process that will follow. Logic in your USRFMT2 routine should validate the input record length.

  The SAS variable RECLEN is specified in the INFILE statement in DYfdaFMT. RECLEN contains the current input record's length. Ensure that USRFMT2 checks the value of RECLEN prior to code that performs an INPUT of fields from the input record to avoid the error of attempting to read beyond the end of the record.

  When USRFMT2 determines that the current input record is not the expected length or is shorter than some minimum length, again the MICS-supplied counter named WEIRDREC must be incremented and the next input record read:

```
    WEIRDREC+1;
    GOTO EOF_CK;
```

- Your USRFMT2 logic must assign the variables ENDTS and ORGSYSID to non-missing values for use by the CA MICS checkpoint process. If you do not assign these variables properly, the prefix.MICS.CHECKPT.DATA data set will not be updated with valid ORGSYSID/COMPONENT checkpoint information. If the timestamp to be associated with the end of an event is not immediately apparent given the input data source, call CA Technical Support for assistance in determining what value you should assign to ENDTS.

  The variable ENDTS is used to compute values for common CA MICS variables such as MONTH, YEAR, HOUR, and DAY. These variables will be computed in the %SYSID macro, which is invoked between the %USRFMT2 and %USRFMT3 macros in DYfdaFMT.

- If your FDA is processing SMF records, you are probably interested in only one or two SMF record types. Read the record type information from the input record and examine it in USRFMT2. When a record type is to be dropped, you should increment the MICS-supplied counter named RECNOTUS to report these dropped records. Add the following SAS statements:

  ```
  RECNOTUS+1;
  GOTO EOF_CK;
  ```

  Note that something similar to this may be accomplished in the common _USRSEL input record selection exit. See section 4.3.2.1 General Operational Exits in this guide.

- The MICS-supplied label COMPROC can be used when USRFMT2 needs to branch to the end of its processing. This label begins the routine that contains the _USRSEL exit point, common field determination, and observation checkpointing.

- DO NOT issue commands that cause the DATA step to return prematurely. (RETURN and DELETE cause premature returns.) Each pass of an input record must "fall through" to the bottom of USRFMT2 or result in a GOTO to a label as described above.

You must not write output files in USRFMT2 because duplicate

```
data is not eliminated until later processing steps execute
CA MICS-supplied CKRTN2.  If the DAILY job that invokes your
format routine and exits is processing the same data as a
previous run of the DAILY, CKRTN2 will eliminate the records
that have been processed before.  Records output from within
USRFMT2 will bypass the duplicate data checking performed by
CKRTN2.
```

7.3.2.1.3 Define Input/Output Processing (USRFMT3)

Use the USRFMT3 macro to read fields from the input data, validate each input field, and format these fields (data elements) according to the needs of your field developed application. Thereafter, build output file observations and direct the writing of output files.

Consider the following items and code your routine appropriately:

- Validate all data that you input. Use either standard or user-defined SAS INFORMATs to validate input data fields, or use logic in your USRFMT3 macro. The following are some areas to examine:
  - impossibly high or low values
  - unexpected negative numbers
  - check the data against a list of permissible values
- Remember to use a trailing "@" (at sign) on INPUT statements that will be followed by a subsequent INPUT statement.
- Create output file data elements from input record fields and other processing decisions.

  The values of calculated data elements (elements whose TYPE statement is XC, C, or a variation) are calculated through the %fffDERV macro. The file's data element calculation macro is invoked by the format routine shell code directly before the file's constructed observation is output, so the code you write in USRFMT3 does not need to duplicate these operations.
- Some input record processing may require you to read a subtype field in the front of the input record and then branch to a particular logic subroutine based on the subtype value.
- If your input records have a segmented structure such that a single input record will result in multiple observations in the database, use USRFMT2 to decode the common information from the record (including ENDTS and ORGSYSID) and use USRFMT3 to decode each segment. Use the SAS LINK statement to branch to the output processing subroutine OUTfff in DYfdaFMT and then return to input the next segment. This is described in more detail below.

- Further in the USRFMT3 logic, if it is determined that the current input record will not contribute to ANY FDA output file, this decision must be handled by branching to label OPTN_DEL so that this record is counted as 'option deleted'.

- Input records whose length is not sufficient for the INPUT process that follows may be encountered. Logic in USRFMT3 may need to validate the (remaining) input record length more than once.

  When USRFMT3 determines that the current input record is not the expected length or not long enough, branch to label SHORTREC so that this record is counted as 'short record deleted'.

At some point, USRFMT3 will reach an output decision point. One observation suitable for output to one or more work files has been built. Its timestamp has been found to allow it into the database based on checkpointing algorithms, and the record's data elements have been filled in. The output branch logic is now referenced.

DYfdaFMT's shell includes code generated from the following template:

```
OUTfff:
   IF &fffSFD THEN DO;
     %fffDERV;
     /* USER COSTING AND MANIPULATION EXIT */
     %FXIT(NAME=USRSfff);
     OUTPUT WORK.iiifff00;
   END;
   RETURN;
```

DYfdaFMT contains one of these routines for each file that appears in its main DATA step. To write an observation to a file, this routine must be referenced by the code in USRFMT3. Choose one of the following methods to reference this code:

- Link to it if you have further processing to perform on the input record. If you have segmented records and each segment will be an output observation or if you will be building two different records in the database for each input record, linking is the best technique to use. The statement format is as follows:

  ```
  LINK OUTfff;
  ```

```
  o Branch to it if no further processing is to be done on
    the current input record.  This will cause the
    observation that you have built to be put to a work
    file and the format routine to return for another record.
    The statement format is as follows:

      GOTO OUTfff;

Two equivalent USRFMT3 macros to illustrate this follow:

  %MACRO USRFMT3;
    detailed processing
    IF condition-favorable THEN LINK OUTfff;
    ELSE GOTO OPTN_DEL;
  %MEND USRFMT3;

  %MACRO USRFMT3;
    detailed processing
    IF condition-unfavorable THEN GOTO OPTN_DEL;
  %MEND USRFMT3;

Remember, if you choose not to use an input record to build
an observation in your database, code GOTO OPTN_DEL.
OPTN_DEL adjusts the input records processed count for the
audit trail and causes the format routine to get the next
input record, if any.
```

7.3.2.1.4 Define Additional Process Code (USRFMT4-6)

```
The USRFMT4, USRFMT5, and USRFMT6 macros exist to support
site-specific requirements.  Use them if you need to perform
a function not directly related to translating input record
fields to output observations.

USRFMT4 - Input end-of-file code
--------------------------------

USRFMT4 is referenced in DYfdaFMT's main SAS DATA step.  The
code you put in USRFMT4 is executed at the time an
end-of-file on the input data set has been detected by SAS
and the last input observation has been processed.

USRFMT4 code should contain valid SAS logic suitable for
inclusion in a DATA step and must not begin with a label.
This macro is typically used to produce a run status report.
A run status report provides the number of records of each
type and subtype that were processed, records validity check
counts, and so on.

USRFMT5 - Post-format, pre-sort code
------------------------------------

After the main DATA step, USRFMT5 is referenced.  The
reference is followed by code that sorts the work file
versions of the component's files into the prescribed order
onto the DETAIL database.

Because this macro is referenced outside of an existing SAS
DATA or PROC step, it must contain DATA or PROC statements of
its own.

It can be used to analyze work files through SAS PROCs, such
as PROC CONTENTS or PROC FREQ, or it can:

  o contain DATA statements to process work files further,

  o incorporate additional control information, or

  o produce additional database files from the original work
    files created in the main DATA step.

On DATA statements, remember to use macros of the form

   ... &iiit..iiifffnn
            (%fffFILE(OP=FILEOPTS,TS=timespanname)) ...
```

```
and on PROC statements, remember to use macros of the form

   PROC SORT DATA=WORK.iiifffnn OUT=&iiit.iiifffnn
            (%fffFILE(OP=FILEOPTS,TS=timespanname));

when writing DETAIL timespan files to the database.

USRFMT6 - End of format routine code
-----------------------------------

Like the USRFMT5 macro, the USRFMT6 macro is referenced
outside of an existing SAS DATA or PROC step, so it must
contain DATA or PROC statements of its own.

It can be used to analyze work files through SAS PROCs, such
as PROC CONTENTS or PROC FREQ, or it can:

  o contain DATA statements to process work files further,

  o incorporate additional control information, or

  o produce additional database files from the original work
    files created in the main DATA step.

On DATA statements, remember to use macros of the form

   ... &iiit..iiifffnn
            (%fffFILE(OP=FILEOPTS,TS=timespanname)) ...

and on PROC statements, remember to use macros of the form

   PROC SORT DATA=WORK.iiifffnn OUT=&iiit.iiifffnn
            (%fffFILE(OP=FILEOPTS,TS=timespanname));

when writing DETAIL timespan files to the database.
```

7.3.2.1.5 Validate Your Code

Two kinds of tests are needed for your DYfdaFMT routine. First is a syntax and operational test and second is a content test, which checks the actual content of data on the database files produced by DYfdaFMT.

The best way to test the syntax and operation of DYfdaFMT is to install the FDA into a test database unit, preferably one constructed solely for the purpose of new product development. Following are the advantages of this method:

- The checkpointing routines operate exactly as they do in a normal production environment.
- You have full control over allocation parameters in your test unit.
- Database data sets, including CHECKPT.DATA and the DETAIL, DAYS, WEEKS, MONTHS, and YEARS timespans, are permanent data sets, enabling you to interrogate the FDA's files in depth.
- The test unit's database files are available to the CA MICS online database access tools, including MSAS (SAS/DM with CA MICS macros available) and MICF (if your site permits it through the Test CA MICS user profile parameter). MICF will know about the field-developed application and will be able to generate inquiries into the new product's data.

To test your DYfdaFMT routine in a test database unit:

- Change sharedprefix.MICS.GENLIB(fdaGENIN) from GEN GENSHELL to GEN GENFILES.
- Install the field-developed application in a test database unit.

  Use the checklist in section 3.8.2 of the PIOM if you have an existing test database unit. Use the checklist in section 3.8.3 of the PIOM if you need to create a test database unit.

  Running the first daily may require several iterations. You can delete the database data sets at any time before the FDA enters a production status and reallocate them for the next test using the standard ALLOCATE or RESTORE jobs. Changes to file structures require resubmitting

the sharedprefix.MICS.CNTL(fdaCGEN) job to take effect.

In reviewing the output of the DAILY job, verify that:

- o data elements contain reasonable values
- o data elements are summarized correctly
- o exception conditions are identified
- o management objective report is formatted correctly
- o checkpoint data set is valid

In reviewing the output of the WEEKLY and MONTHLY jobs, verify that:

- o data elements contain reasonable values
- o archive files contain valid data
- o management objective report is formatted correctly

In reviewing the output of the YEARLY job, verify that:

- o the MICSLOG and SAS Log do not contain error messages

In reviewing the output of the fdaCHECK job:

- o verify that the file keys for your higher timespan files are constructed correctly.

- o consider running the job in the fdaCNTS member in sharedprefix.MICS.SOURCE. This job will perform a PROC CONTENTS on each cycle of each file in your FDA. Be sure that the MONTHLY job has been run before you run the fdaCNTS job.

- o consider running the job in the fdaDATAS member in sharedprefix.MICS.SOURCE. This job will perform a PROC DATASETS on each timespan in the unit that contains your FDA. Be sure that the MONTHLY job has been run before you run the fdaDATAS job.

7.3.2.2 Define the MSI Input Routine

One of the advantages of using MSI is that there is no input format routine to code. In fact, most FDAs using MSI require no additional coding beyond the fdaGENIN member.

This is not say that FDAs using MSI cannot be modified. MSI provides many user exits that allow for customization.

1 - Original Sysid Assignment Exit (cccSYSID)
2 - MSI File User Exits
3 - MSI User Exits

7.3.2.2.1 Original Sysid Assignment Exit (cccSYSID)

The data element ORGSYSID is used by CA MICS to identify the originating system of the data. The cccSYSID exit is used for specialized assignment of the ORGSYSID over and above the ORGSYSID statement. As an example, many data sources contain ORGSYSIDs up to 256 characters. Since CA MICS requires a four-byte ORGSYSID, this exit can be used by the installation to recode extra-long ORGSYSIDs to four bytes. Another use of cccSYSID is to assign an ORGSYSID when the input data does not provide one.

cccSYSID is located in prefix.MICS.PARMS and is automatically copied as part of the installation of the FDA.

7.3.2.2.2 MSI File User Exits

MSI provides user exits for each file to allow for custom processing, without modification to MSI code. These exits are automatically generated as part of the normal MCG generation process (GEN GENFILES) and are implemented using the FXIT facility. Those marked with an asterisk (*) are Open Code Exits that are not implemented in a data step.

Exit Name	Description
USRXfff	Exit prior to output to detail timespan. If the detail timespan is not active, then this exit is not invoked.
USRTfff	Detail tape processing file (this exit is currently inactive).
fffFMT2	Exits before Common (SYSID, ZONE, etc.) and CKRTN2 processing. This exit can be used to set the ENDTS, STARTTS, and ORGSYSID variables.
fffFMT3	Exit after CKRTN2. This exit can be used to perform additional processing of the input SAS data sets.
fffFMT4	Exit at End-of-File.
*fffFMT5	Exit after the input SAS data has been processed.
*fffFMT6	Exit after all processing for fff has been completed.

MSI File User Exits
(*) Open code exits

Notes:

o For any file that specifies the CREATEWITH option, the fffFMT3 exit must be used to populate this file. At a minimum, the fffFMT3 example must contain:

```
LINK OUTfff; * where fff is the file id;
```

o Do not output any observations in fffFMT2. Duplicate data checking is performed by CKRTN2, AFTER fffFMT2 is invoked on any observation.

o Presently, exits fffFMT2, fffFMT4, and fffFMT6 are not taken for files specifying the CREATEWITH option.

7.3.2.2.3 MSI User Exits

MSI provides user exits for each file to allow for custom processing, without modification to MSI code. These exits are automatically generated as part of the normal CA MICS Component Generator (MCG) generation process (GEN GENFILES), and are implemented using the FXIT facility. Those marked with an asterisk (*) are Open Code Exits that are not implemented in a data step.

*cccFMT6	Exit after the processing of all input files for product ccc.
*MSIFMT1	Exit prior to processing of any input files.
*MSIFMT6	Exit after the processing of all input files.

7.3.3 Define Reports (DYfdaEXC, fdaEVA, and DYfdaMBO)

CA MICS supports exception and management objective reporting in its batch job reporting processes. In this section, we discuss coding conditions that should be reported as well as coding the reports themselves.

```
These topics are discussed in detail in the following
subsections:

    1 - Define Exception Reporting
    2 - Define Management Objective Reporting
```

7.3.3.1 Define Exception Reporting

```
Exception processing is handled using the following members:

  o sharedprefix.MICS.SOURCE(fdaEVA) is the exception value
    analysis routine, which helps you choose values to code
    in the exception test routine

  o prefix.MICS.CNTL(fdaEVA) is the JCL to execute the
    exception value analysis routine

  o sharedprefix.MICS.SOURCE(DYfdaEXC) is the exception test
    routine

If you plan to use CA MICS exception processing, you must
define the shells for the exceptions in the DYfdaEXC members
in sharedprefix.MICS.SOURCE and
sharedprefix.MICS.HOLD.USER.SOURCE.

The HOLD.USER.SOURCE version of DYfdaEXC should contain:

  /* COMPONENT: fda                          */
  /* EXCEPTION PROCESSOR MODULE              */
  /* GENERATED: mm/dd/yy BY userid           */
  %INCLUDE SOURCE(DYfdaEXC);

where you fill in your FDA's ID, the date, and your user
identification.

The CA MICS.SOURCE version of DYfdaEXC should contain:

  /* COMPONENT: fda                          */
  /* EXCEPTION PROCESSOR MODULE              */
  /* GENERATED: mm/dd/yy BY userid           */

where you fill in your FDA's ID, the date and your user
identification.

There are two styles of exception processor used in CA MICS:

  o the normal processor, like that used by the CA MICS
    Analyzer for TSO (DYTSOEXC), scans newly created file
    cycles looking for thresholds that have been exceeded.

  o an alternative processor, like that used by the CICS
    Analyzer (DYCICEXC), is useful if your FDA does not
    create DETAIL timespan files.  The DYCICEXC module
    illustrates a technique that examines the values of key
    indicators as the detailed data is read.  If the DYCICFMT
```

```
    routine notes an exception condition, it writes a record
    to the CICS Incident File (CICCIN).  The CICCIN file is
    read by the DYCICEXC routine to report exceptions.

Code the exception processor logic in
sharedprefix.MICS.SOURCE(DYfdaEXC) so that the same version
of the exception processor code can be used by all database
units in which the FDA is installed.  (You can tailor
exception processors for individual database units later.)

The general form of an exception processor is:

  o One SAS DATA step per file examined in the process.  We
    call this data step an analysis DATA step for ease of
    discussion.

  o One final DATA step to merge exceptions found for this
    FDA with exceptions found in previous processors.

Each analysis DATA step examines observations from one of the
FDA's CA MICS database files or one other source of
exception possibilities.  Each analysis DATA step contains an
intermediate file in the DATA statement, the SET statement
(or INFILE statement with input code), observation selection
logic (if needed), followed by a HIT routine, followed by a
TESTS section.

The HIT routine is the single place where observations are
written to the working exception file, and is linked to by
specific test routines.

The TESTS section contains one or more exception tests.  Each
exception test is in SAS DO group form:  if the exception
condition is present, the exception data elements are set and
the HIT routine is linked.  Code your exception tests in the
format outlined below.

The fdaEVA routine in sharedprefix.MICS.SOURCE helps you
choose values to code in the TESTS section.  The fdaEVA
routine produces a report that shows statistical measures for
the data elements used in DYfdaEXC.  To produce the report,
submit the job in prefix.MICS.CNTL(fdaEVA).
$SCOL
The sharedprefix.MICS.SOURCE(DYfdaEXC) code follows this
format:

/* ****************************************** */
/* EXCEPTION ANALYSIS OF THE CURRENT DATA     */
/* (product and file IDs here)                */
```

```
/* ***************************************** */

DATA ADMfffDT (KEEP=%EXCKEEP(TS=DAYS));
/* ***************************************** */
/* if the timespan of analyzed data had been     */
/* DAYS instead of DETAIL, the intermediate file */
/* name above would be of the form ADMfffDY.     */
/* ***************************************** */
%EXCLEN(TS=DAYS);
%EXCFMT(TS=DAYS);
SET &iiit..iiifff01 END=EOFSW;
     /* iii=info. area, t=X for DETAIL, D for DAYS */
RETAIN INFOAREA 'iii'
       TIMESPAN 'DETAIL'     /* or DAYS */
       DBFILE   'iiifff01'
      EXCCOUNT 1;
   IF _N_ = 1  THEN PUT INFOAREA= TIMESPAN= DBFILE= ;
   OBSERVNO = _N_;
   GOTO TESTS;
HIT:
   OUTPUT;
   EXCCODE  = '00000';
   SEVERITY = ' ';
   MGMTAREA = ' ';
   EXCDESC1 = ' ';
   EXCDESC2 = ' ';
   RETURN;
TESTS:
 /* ***************************************************** */
 /* EX NR: 9nnnn                                          */
 /* TITLE: exception condition identification text        */
 /* FILE:  timespan.iiifff01                              */
 /* ***************************************************** */
IF (exception condition)
 THEN DO;
  EXCCODE='9nnnn';
  SEVERITY='C';
  MGMTAREA='PERFORMANCE';
  EXCDESC1='exception condition identification text';
  EXCDESC2='additional condition text, or blank';
  LINK HIT;
END;
 /* ***************************************************** */
 /* EX NR: 9nnnn                                          */
 /* TITLE: exception condition identification text        */
 /* FILE:  timespan.iiifff01                              */
 /* ***************************************************** */
IF (exception condition)
  ...
```

```
END;
RUN;
************************************************************;
DATA &ADMD..ADMEXC00 (KEEP=%EXCKEEP(TS=DAYS));
%EXCLEN(TS=DAYS);
%EXCFMT(TS=DAYS);
%EXCLBL;
SET  &ADMD..ADMEXC00 WORK.ADMfffDT ;

* BYPASS ADMEXC OUTPUT IF FILE DEFINITION IS NOT ACTIVE ;
IF &EXCSFD THEN OUTPUT;
RUN;

PROC DATASETS NOLIST NOWARN DDNAME=WORK;
DELETE  ADMfffDT ;
RUN;
QUIT;
```

The final DATA step interleaves the new exceptions with the existing master exception file contained in the DAYS timespan as file ADMEXC00 (aged to ADMEXC01).

7.3.3.2 Define Management Objective Reporting

CA MICS supports reporting management objectives for the DAYS, WEEKS, and MONTHS timespans. Code that identifies management objectives resides in sharedprefix.MICS.SOURCE members named:

- DYfdaMBO for daily reports
- WKfdaMBO for weekly reports
- MNfdaMBO for monthly reports

Guidelines for coding management objective reports are:

- Do not update the CA MICS database in a management objective report module.
- Tailor your objectives to the timespan being reported.

7.4 Testing, Documenting and Deploying the FDA

At this point, you have written and validated your field-developed application code. In this section, you will

o verify that the code is valid

o write documentation to include in the CA MICS documentation set

o deploy the FDA in production

To perform a final test, consider running the CA MICS operational jobs against data that you create to test its functions, including the duplicate data checks and exception and management objective reports.

To document your field-developed application in standard CA MICS guides, follow these guidelines:

Add documentation to the Database Structure and Content Guide.

The members you edit or create in sharedprefix.MICS.DOC.TEXT are:

IZ - add a section reference to the section you are adding.

For example, IZ might contain:

```
+-----------------------------------------------+
|$HEDR Chapter 35. USER COMPONENTS              |
|                                               |
| XYZ Corporation has written and installed     |
| the following user components on its CA MICS  |
| system:                                       |
|                                               |
| Product                    Information Area   |
| ------------------------------------------- |
| UNP - User NPA Component   UNP - 3705 Activity |
|                            UNI - Incidents    |
|                            UCA - Lines, etc.  |
| ULO - Logon Tracker        ULO - Logon Info   |
| USM - HSM Activity         USM - HSM Activity |
|                                               |
| For more information, refer to the following  |
| sections:                                     |
|                                               |
|   1 - UNP Information Area                    |
|   2 - UNI Information Area                    |
|   3 - UCA Information Area                    |
|   4 - ULO Information Area                    |
|   5 - USM Information Area                    |
|                                               |
+-----------------------------------------------+
```

IZn - write an overview of the FDA's information area.

For example, IZ1 might contain:

```
+-----------------------------------------------+
|$HEDR 35.1 - UNP Information Area              |
|                                               |
| This section identifies each file in the UNP  |
| Information Area of the UNA Product.          |
|                                               |
| This information area provides ....           |
|                                               |
| The files in this information area are described|
| in the following sections:                    |
|                                               |
|   1 - UNC File - 3705 Activity                |
|   2 - UN_ File - ...                          |
|   3 - UC_ File - ...                          |
+-----------------------------------------------+
```

IZnf - write an overview of the information area's file.

```
     For example, IZ11 might contain:

     +-----------------------------------------------+
     |$HEDR 35.1.1 - UNC File (3705 Activity)        |
     |                                               |
     | The 3705 Activity File (UNPUNCxx) contains data |
     | elements that describe ...                    |
     |                                               |
     | The following sections describe:              |
     |                                               |
     |   1 - File Organization                       |
     |   2 - Data Elements List                      |
     |   3 - Usage Considerations                    |
     +-----------------------------------------------+

IZnf1 - call in the file's generated organization list

     For example, IZ111 might contain:

     +-----------------------------------------------+
     |$HEDR 35.1.1.1 - File Organization             |
     |                                               |
     | $SMEM #INS000S                                |
     | $SMEM #INSUNCS                                |
     | $FIG Figure 35-x. UNPUNC File's Organization  |
     +-----------------------------------------------+

IZnf2 - call in the file's generated element list

     For example, IZ112 might contain:

     +-----------------------------------------------+
     |$HEDR 35.1.1.2 - Data Elements List            |
     |                                               |
     |$SMEM #INS000E                                 |
     |$SMEM #INSUNCE                                 |
     |$FIG Figure 35-x. UNPUNC File's Data Elements  |
     +-----------------------------------------------+

IZnf3 - write notes about how to use the file
```

For example, IZ113 might contain:

```
+-----------------------------------------------+
|$HEDR 35.1.1.3 - Usage Considerations          |
|                                               |
|  When using the UNPUNC File, ... (special     |
|  considerations)                              |
|                                               |
|  Do not use this file to ...                  |
|                                               |
|  To retrieve information from this file, ...  |
+-----------------------------------------------+
```

PICIZ - add a call to your new section

For example, PICIZ might contain:

```
+-----------------------------------------------+
|$PAGE                                          |
|$LMEM ITITLE0Z Title Page                      |
|$LMEM ITITLE0Z Title Page                      |
|$$TOC          Table of Contents Generator     |
|$SPAG                                          |
|$PAGE  35 001                                  |
|$SMEM IZ       USER COMPONENTS                 |
|$SMEM IZ1      UNP Information Area            |
|$SMEM IZ11     UNC File                        |
|$SMEM IZ111    File Organization               |
|$SMEM IZ112    Data Elements List              |
|$HEDR IZ113    Usage Considerations            |
|$SMEM IZ12     UN_ File                        |
|$SMEM IZ121    File Organization               |
|$SMEM IZ122    Data Elements List              |
|$HEDR IZ123    Usage Considerations            |
|$SMEM IZ2      UNI Information Area            |
|$SMEM IZ21     UN_ File                        |
|$SMEM IZ211    File Organization               |
|$SMEM IZ212    Data Elements List              |
|$SMEM IZ213    Usage Considerations            |
| ...                                           |
+-----------------------------------------------+
```

PICIXD - add a Data Dictionary for your FDA by adding
calls to data elements existing in or added by
you to sharedprefix.MICS.DIC.TEXT

For example, PICIXD might contain:

```
+-----------------------------------------------+
|$PAGE                                          |
|$LMEM ITITLEXD Database Structure and Content Gui|
|$LMEM ITITLEXD Database Structure and Content Gui|
|$SPAG                                          |
|$$TOC                                          |
|$PAGE   D 001                                  |
|$SMEM IXD      Data Dictionary                 |
|$SMEM IXD1     The Structure of the Data Dictiona|
|$SMEM IXD2     Common Data Elements            |
|$DMEM ASID     Address Space Identification    |
|$DMEM CHANNEL  Physical Channel Address        |
|$DMEM COMMAND  Command/Transaction Name        |
|$DMEM DAY      Day of Month                    |
| ...                                           |
|$SMEM IXD34    The ADMCYC (Cycles Log) File    |
|$DMEM IHLCYCLE MICS Cycle                      |
|$DMEM IHLLOWTS Input Data Low Time Stamp       |
|$DMEM IHLHITS  Input Data High Time Stamp      |
|$SPAG                                          |
|$SMEM IXDZ     Field Developed Applications    |
|$DMEM UNPABNDC Abend Code                      |
|$DMEM UNPABNDN Abend Message Number            |
|$DMEM UNPADSOA ADS/Online Application Name     |
|$DMEM UNPAVDCT Average Conversational Respons Tim|
|$DMEM UNPCOST  Processing Charges              |
|$DMEM UNPCRSTM Conv. Transaction Response Time |
| ...                                           |
|$SPAG                                          |
+-----------------------------------------------+
```

To deploy the FDA in production, follow these steps:

o Redefine the FDA in sharedprefix.MICS.GENLIB(DBTABLE)

If you placed the FDA in its own test database unit,
follow the guidelines in section 2.3.3.3 of the PIOM for
changing the FDA's unit identifier from T (test unit) to
either P (primary unit) or U (standard database unit).

o Rebuild the database unit jobs to include your FDA

  - Submit the job in prefix.MICS.CNTL(JCLGENB) to generate installation JCL members.

  - Submit the job in prefix.MICS.CNTL(COPYLIBU) to move CA MICS parameter library members that relate to your FDA.

  - Submit the job in prefix.MICS.CNTL(CYCLEGEN) to generate database unit definitions.

  - Submit the job in prefix.MICS.CNTL(BACKUP) to back up the database unit definitions.

  - Submit the job in prefix.MICS.CNTL(RESTORE) to restore the new specifications to the new database unit.

7.5 MSI Definition Notes

The following sections contain additional information for using the CA MICS-SAS Interface.

1 - Default Data Types
2 - Resolving Data Element Type Conflicts
3 - Formats Assigned to Input SAS Variables
4 - Account Codes
5 - File Sort Sequences
6 - @@FIRST and @@LAST
7 - KEEP and DROP Processing
8 - CA MICS Chargeback and Accounting Interface
9 - Data Dictionary Aliases

After reviewing this section, refer to Appendix A for a checklist on installing an MSI generated FDA.

7.5.1 Default Data Types

Numeric elements are defined as Accumulated. Character variables or data elements defined as part of a sort sequence are always defined as RETAINed. These definitions can be overridden using the MIN, MAX, RETAIN, COMPUTED, AVERAGE, and PERCENT MSI statements.

7.5.2 Resolving Data Element Type Conflicts

As MSI reads the variable definitions from the model SAS data set, it verifies the information against known common data elements. If a match is found, MSI will verify that the type and length of the SAS variable matches the common data element definition. If a type mismatch occurs, MSI will make the data element non-common and use the definition from the SAS data set.

While a type conflict does not trigger an error (warning messages are issued), this condition should be avoided since it can prevent the merging of information.

To properly resolve these conflicts, it is recommended that the SAS variable be renamed (using the MSI RENAME statement).

7.5.3 Formats assigned to Input SAS Variables

During the SAS Input Data Set evaluation phase of MSI processing, the output format of the SAS variable is assigned to the CA MICS data element. If a SAS variable has a user defined SAS format associated with it, daily processing will terminate with a FORMAT NOT FOUND error. To prevent this error, two methods are recommended. First: include the SAS options NOFMTERR in sharedprefix.PARMS(MATSO/MABATCH). Note that this may allow errors previously trapped to be allowed. Second: use the USEROBJLIB option in JCLDEF/JCLDEFC to make the user defined formats available to CA MICS processing.

MSI assigns the length of the input SAS variables to the data elements. Currently, there is no facility to change lengths of data elements. If the data element length is too small to contain all the data, the length of the input variable must be increased.

7.5.4 Account Codes

All MSI components support user defined account codes. To add account codes to an MSI file, a NAME or NAMX statement must first be added for @@ACCT:

```
NAME @@ACCT 00 0 0 0 0 0
```

After this, the special MCG variable @@ACCT can be used on the MSI SEQUENCE or KEEP statements. The sharedprefix.MICS.PARMS members fdaACCT and fdaACRT contain the account field definitions and assignment code. Note that these must be completed. If you do not have a SEQUENCE statement no user defined sequence elements will be generated regardless of the NAME @@ACCT statement.

If NOACCT was specified during the GENSHELL process, the sharedprefix.MICS.PARMS fdaACCT and fdaACRT may not have been generated and will need to be created.

7.5.5 File Sort Sequences

In order to simplify the specification of the file sort sequences, MSI breaks the sort sequence into two pieces:the User and the Timespan Sequences. The User Sequence is defined by the SEQUENCE, and APPENDSEQUENCE statements.

The Timespan Sequence determines how a file is sorted with regard to the timespan where it is active (detail,days, ...). It is automatically generated by MSI and cannot be directly modified. The ADD statement allows for limited changes in the Timespan Sequence for specific timespans. AFTERSEQUENCE allows data elements to be added at the end of the Timespan Sequence.

To insert elements before the Timespan sequence, you have to add after the User Sequence using the APPENDSEQUENCE.

7.5.6 @@FIRST and @@LAST

Special CA MICS data elements @@FIRST and @@LAST are used to define specialized code to be executed before and after computed data elements are calculated. MSI provides support through the standard CA MICS Component Generator facilities (refer to section 4.2.4.4.3 in this guide).

7.5.7 KEEP and DROP Processing

The FDA definition using MSI is simplified through the use of new statements.

The KEEP and DROP statements control whether or not a data element is active.

The following rules apply to KEEP/DROP processing:

- If no KEEP/DROP statement is specified, then the default action is to KEEP.
- Once a KEEP statement is encountered, the default action is to DROP.
- Sequence Elements cannot be dropped.
- KEEP and DROP specified for individual timespans override Global Definitions.
- Dropping a data element in Detail also drops in Days, Weeks, Months, and Years.

7.5.8 CA MICS Chargeback and Accounting Interface

MSI fully supports CA MICS Chargeback and Accounting by providing for all MSI files the required exits and interfaces. In order to activate Accounting and Chargeback for MSI FDA file journal files, charging elements and rate tables must be defined. Please refer to the CA MICS Chargeback and Accounting Guide, section 4.13, for details.

7.5.9 Data Dictionary Aliases

When defining a standard FDA, all non-common data elements must start with the file id (or the parallel file id). For FDAs that use the CA MICS-SAS Interface (MSI), this rule has been lifted since the input SAS data set can contain variables of any name. This can potentially lead to the Data Dictionary Entry for an existing CA MICS data element to be overlaid with the information for a 'new' data element. For example, an MSI FDA that contains a data element called CPUBUSY would conflict with the existing CA MICS data element CPUBUSY from the HARCPU file.

To avoid this situation, the CA MICS Component Generator (MCG) will automatically generate Data Dictionary Aliases for any file that uses the CA MICS-SAS Interface. These alias names have the form fffnnnnn, where fff is the file id, and nnnnn is a unique number generated by the MCG. This generated alias is used except when the data element already exists in the CA MICS File Definition Table (FDT). In this case, the alias entry in the FDT is used.

The MCG Alias statement can be used to reassign the alias name for ANY data element, and will override any automatically generated names. Note that the ALIAS statement has an alternate form, that can only be used with MSI (refer to section 4.2.4.3.6.5 in this guide).

7.6 MSI Unit Level Parameters

FDAs created using MSI will introduce three members in prefix.MICS.PARMS: fdaOPS, INPUTfda, and fdaSYSID.These members are described in the following sections:

1 - FDA Processing Options (fdaOPS)
2 - Input Data set Specification (INPUTfda)
3 - Original System Id Assignment (fdaSYSID)

7.6.1 FDA Processing Options (fdaOPS)

The fdaOPS member of prefix.MICS.PARMS contains statements for the processing options of the MSI FDA.

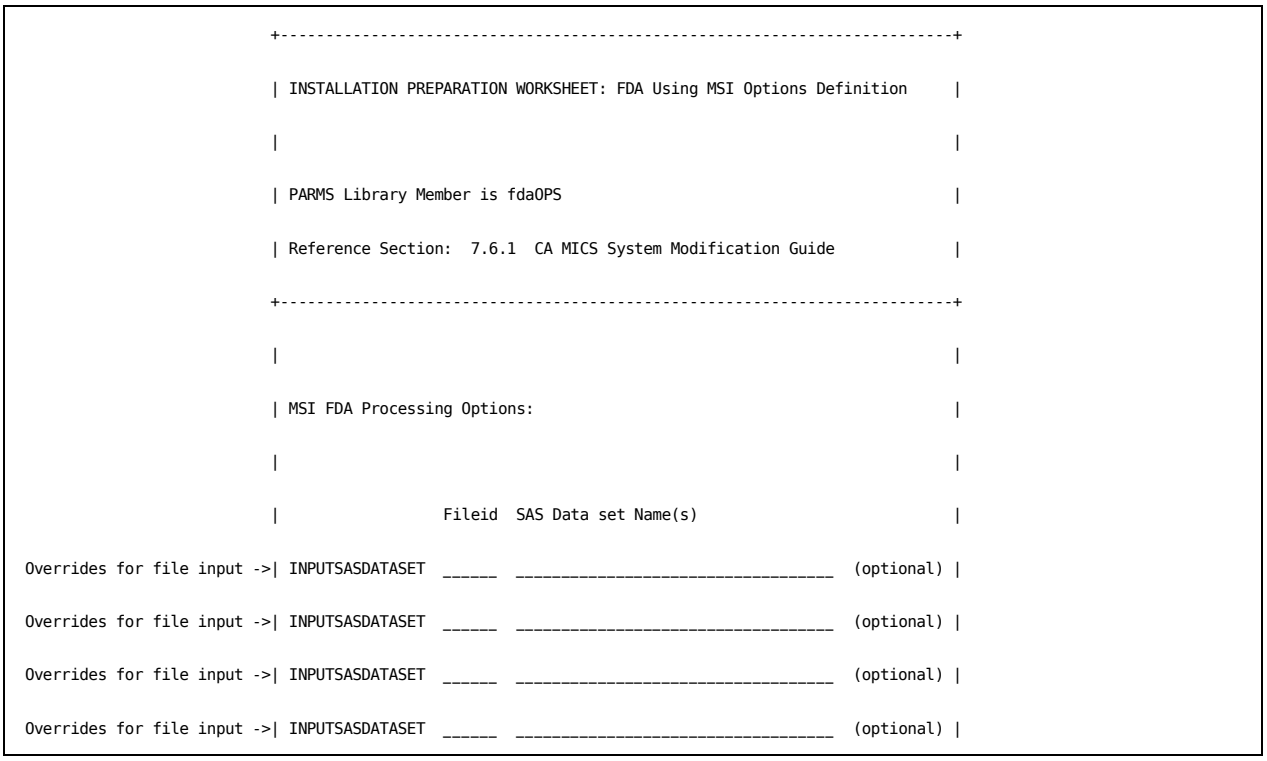
fdaOPS contains the following optional statements:

- o INPUTSASDATASET Statement(optional)
- o CHECKPOINTMODE Statement (optional)
- o Work File definitions (optional)

Each FDA using MSI has its own set of parameters defined in prefix.MICS.PARMS(fdaOPS). These parameters control the specification of work files, checkpoint processing, and input SAS data sets.

Figure 7-3 shows the Processing Options Definition Worksheet for an FDA using MSI. Its statements are described after the worksheet in the various subsections:

1 - Input Data set Specification (INPUTSASDATASET)
2 - Check Point Processing Options (CHECKPOINTMODE)
3 - Work File Parameters



```
                          +-------------------------------------------------------------------------+
                          | INSTALLATION PREPARATION WORKSHEET: FDA Using MSI Options Definition     |
                          |                                                                         |
                          | PARMS Library Member is fdaOPS                                          |
                          | Reference Section:  7.6.1  CA MICS System Modification Guide            |
                          +-------------------------------------------------------------------------+
                          |                                                                         |
                          | MSI FDA Processing Options:                                             |
                          |                                                                         |
                          |                 Fileid  SAS Data set Name(s)                            |
Overrides for file input ->| INPUTSASDATASET  ______  ______________________________________  (optional) |
Overrides for file input ->| INPUTSASDATASET  ______  ______________________________________  (optional) |
Overrides for file input ->| INPUTSASDATASET  ______  ______________________________________  (optional) |
Overrides for file input ->| INPUTSASDATASET  ______  ______________________________________  (optional) |
```

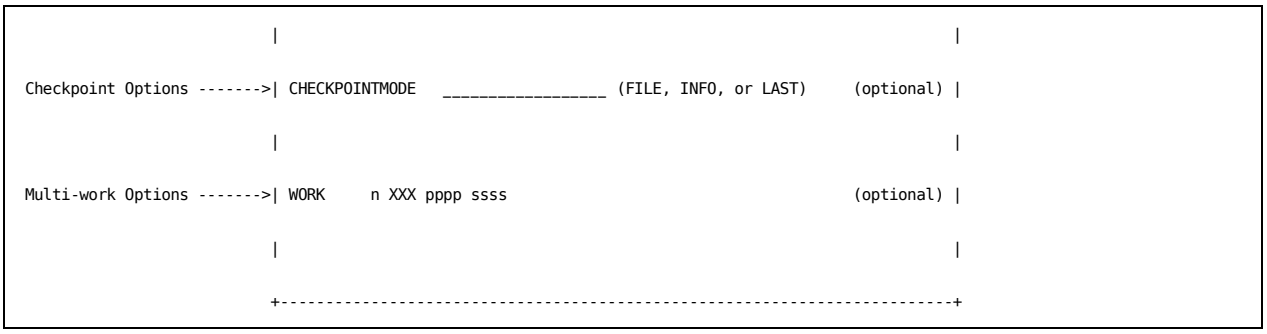


Figure 7-3. MSI FDA Processing Options Definition Worksheet

7.6.1.1 Input Data set Specification (INPUTSASDATASET)

In the fdaGENIN member, the INPUTSAS statement specifies the default name SAS data set used for input in the DAYnnn step. MSI provides the INPUTSASDATASET option to override the default input data set as well as add additional input data sets.

Multiple data sets can be specified per statement, and multiple INPUTSASDATASET statements can be specified per file. Duplicate data set names are ignored. Do not specify SAS data set options on SAS data set names. Any libnames specified on INPUTSASDATASET statements must be defined in the INPUTfda.

The order of the INPUTSASDATASET statements determine the order the input SAS data set will be read.

The INPUTSASDATASET statement format is:

```
 INPUTSASDATASET fileid datasetname ...
```

where:

fileid = three character fileid

datasetname = the name of the SAS data set to be used for input. Note that the data set name should be of the format libname.membername. Any libnames specified should appear in the INPUTfda member.

Example:

```
INPUTSASDATASET ZZZ FIRST.TYPEXX SECOND.TYPEXX
INPUTSASDATASET ZZZ THIRD.TYPEXX
INPUTSASDATASET ZZT FIRST.ZORRO
INPUTSASDATASET ZZZ THIRD.TYPEXX
```

Note that the above statements define input SAS data sets for two files ZZZ and ZZT. Note that for file ZZZ, the second THIRD.TYPEXX will be ignored.

7.6.1.2 Check Point Processing Options (CHECKPOINTMODE)

Normally, CA MICS Checkpoint processing will cause a U300 ABEND if there is less than ten new records input or selected during daily processing of an FDA. Since FDAs using MSI can support multiple input files, MSI provides options to control when checkpoint processing occurs.

The syntax of the CHECKPOINTMODE statement is

```
  CHECKPOINTMODE option
```

where option is LAST, INFO, or FILE. The default is FILE.

CHECKPOINTMODE options are

LAST - at least 10 observations for all files in the MSI FDA.

INFO - all files within an Information area must have at least ten observations.

FILE - each file must have at least 10 observations.

7.6.1.3 Work File Parameters

FDAs using MSI can utilize standard CA MICS multiple work file support to spread FDA work files across multiple WORKnn data sets. Use the WORK parameter described below to activate multiple work file support for your MSI FDA.

Note: Before using the WORK parameter, see information on the use of the MULTWORK parameter in Section 4.2.4.3.1, MSI Options Statement (MSIOPTS).

```
WORK n XXX pppp ssss
--------------------

This statement is optional.  It enables sites experiencing
either SAS WORK space allocation problems or out of work
space conditions during DAYnnn, where nnn is the job step
step number, daily update processing to allocate multiple
WORK files.

You can allocate up to nine SAS WORK files for use during the
daily update step.  These additional work files are used in
conjunction with the single work data set allocated by
default using the WORKSPACE parameter.

Because the individual space allocation requirement for each
WORK file is typically much smaller, it is more likely to be
satisfied.

To take advantage of multiple WORK files support, edit
prefix.MICS.PARMS(cccOPS), where (ccc) is the component
identifier, and insert a WORK statement as shown below:

       WORK n XXX pppp ssss

where: n (0-9) is the number of WORK data sets
       XXX  is TRK or CYL
       pppp is the primary allocation
       ssss is the secondary allocation

NOTE:  When allocating any number of SAS WORK data sets, be
aware that one additional SAS WORK data set will
automatically appear in the DAYnnn JCL to facilitate sorting.
For example, if you allocate six SAS WORK data sets, you will
get seven in actuality.

Example 1:  WORK 6 TRK 800 100

In this example, six WORK data sets are specified.  Each data
set has an allocation of 800 tracks primary and 100 tracks
secondary.
```

```
Example 2:  WORK 4 CYL 100 20

In this example, four WORK data sets are specified.  Each
data set has an allocation of 100 cylinders primary and 20
cylinders secondary.

NOTE:  Since there is some performance impact when using
multiple work files, you should specify the minimum number of
work data sets to meet your work space requirements.  As a
start, try specifying 2 or 3 work data sets.

How Much Space Should You Allocate?
-----------------------------------

Review prefix.MICS.PARMS(JCLDEF) for EACH unit containing
this product.  Find the WORKSPACE parameter.  It will
resemble this sample statement:

      WORKSPACE      TRK 500 250

The value shows the current SAS WORK space allocation for the
unit as a single data set.  It also serves as the default
value used in the units DAYnnn daily update step unless you
provide a WORK parameter.

To achieve the equivalent work space allocation of
WORKSPACE TRK 500 250 using multiple WORK data sets that will
collectively share the work space requirements of the daily
update step, you could code either:

      WORK 2 TRK 250 125 or WORK 5 TRK 100 50

To determine the total work space, multiply the number of
work files (n) by the primary (pppp) and secondary (ssss)
values specified.

SASWORK
-------

This statement is optional.
```

```
The WORK DD statement in the CA MICS procedures allocates
a temporary data set where SAS keeps its temporary data
files and other items that SAS uses during processing of
the current job.

By default, the space allocated is defined in the member
prefix.MICS.PARMS(JCLDEF) with the WORKSPACE and WORKUNIT
parameters, then generated into all the JCL procedures for
a given unit.

With the SASWORK statement you have the option to override
this unit-wide definition to specify the space allocation
individually for the current step.

The format of the SASWORK statement is:

SASWORK data_set_allocation_parameters

where data_set_allocation_parameters is one or more data set
allocation parameters (for example, STORCLAS or SPACE)
separated by spaces.

You can also specify the SASWORK parameter as the following:

SASWORK XXX pppp ssss

where:

  XXX  is TRK or CYL
  pppp is the primary allocation
  ssss is the secondary allocation

If you omit the data_set_allocation_parameters or the SASWORK
statement, the WORK data set is allocated according to the
values you specified for the WORKUNIT and WORKSPACE
parameters in prefix.MICS.PARMS(JCLDEF).  Use the
data_set_allocation_parameters to override this default,
either to alter the space allocation or to use System Managed
Storage (SMS) parameters to control data set placement and
characteristics.

Specify data set allocation parameters, separated by blanks,
according to SAS LIBNAME statement syntax.  If you need
multiple lines, repeat the SASWORK keyword on the
continuation line.
```

```
Example:

SASWORK STORCLAS=MICSTEMP SPACE=(XXX,(pppp,ssss))

where:

  STORCLAS - specifies a storage class for a new data set.
             The name can have up to 8 characters.
  SPACE    - specifies how much disk space to provide for
             a new data set being allocated.
  XXX      - is TRK or CYL.
  pppp     - is the primary allocation.
  ssss     - is the secondary allocation.


Note:  If you change the SASWORK parameter, you must:

o  Run cccPGEN
o  Run JCLGENU for DAILY (to regenerate DAILY)
```

7.6.2 Input Data set Specification (INPUTfda)

The JCL statements defining the libnames specified on either the INPUTSASDATASET statement in fdaOPS or the INPUTSAS in fdaGENIN are specified here. These JCL statements will be included in the daily processing JCL.

7.6.3 Original System Id Assignment (fdaSYSID)

The data element ORGSYSID is used by CA MICS to identify the originating system of the data. The fdaSYSID exit is used for specialized assignment of ORGSYSID. For example, many data sources contain ORGSYSIDs up to 256 characters. Since CA MICS only supports four-byte ORGSYSIDs, this exit can be used by the installation to recode extra-long ORGSYSIDs to four bytes. Another use of fdaSYSID is to assign an ORGSYSID when the input data does not provide one.

fdaSYSID is invoked after COMMONEXIT statements.

Chapter 8: CA MICS NSM Integration (MTI)

The integration of NSM data to CA MICS is made possible by a special group of features designed to bring NSM performance, chargeback, and asset management data into CA MICS. This bundle of features, called MTI (CA MICS NSM Integration), comes with base CA MICS and, fully exploited, can add powerful new perspectives for resource management at your site.

This chapter describes these features and utilities and provides checklists for enabling and working with them.

This section contains the following topics:

8.1 NSM Systems Performance (see page 441)
8.2 CA MICS NSM Integration (MTI) (see page 442)
8.3 Features (see page 444)
8.4 MTI Reporting (see page 446)
8.5 Integration Checklists (see page 454)
8.6 Considerations (see page 469)
8.7 MTI Unit Level Parameters (see page 472)
8.8 MTI Operational Jobs (see page 486)

8.1 NSM Systems Performance

NSM is the management solution that enables businesses to get the most out of their technology infrastructure. By providing a universal method for monitoring and managing all your mission-critical resources, this powerful solution enables you to gain and maintain control of your existing environment while confidently exploiting new technologies and new business opportunities.

NSM dynamically models your environment. Its powerful EnterpriseDiscovery (TM) automatically discovers your entire spectrum of IT assets and populates the object repository with the information it has identified about the various devices, their relationships, connections, and topology.

As part of its base functionality, NSM enterprise management includes comprehensive performance management capabilities. Utilizing intelligent agents, thresholds, and correlation, NSM continually monitors performance, collects usage data for long-term planning and trend analysis, and manages unexpected performance problems before they impact business operations.

NSM lets administrators view their entire environment at once with functions that analyze and present data in a meaningful and consistent form. With this global view, administrators can better pinpoint bottlenecks, identify components that are consuming excessive resources, and evaluate how changing system parameters will affect performance.

NSM samples and stores both real-time and historical data with its base agent technology. These agents have small footprints and can save performance data locally, placing minimal demands on networked systems. The type of performance information collected is easily selected from a wide range of predefined choices.

NSM enables administrators to configure performance agents with threshold policies. When the performance of any component falls outside the policy bounds, an event is escalated to the NSM Domain Manager. The Domain Manager references its own policy to determine how best to respond to the situation. Domain Managers are thus able to resolve more complex or wide-ranging performance issues.

For more information about NSM, including how to contact CA with questions or requests, please visit the CA web site: www.ca.com.

8.2 CA MICS NSM Integration (MTI)

CA MICS processes data from many different z/OS data sources and distributed computing platforms such as UNIX (via HP's Measureware), Tandem, DOS/VSE, and DEC. In response to client requests to support systems such as AIX, Sun, and Microsoft NT servers, CA MICS provides support for these platforms using NSM Systems Performance.

MTI provides the base technology to read the NSM Systems Performance data and extends standard DIA support by providing the tools for installations to create and customize their own components for integrating NSM Systems Performance data into the CA MICS architecture.

MTI technology includes a new approach for delivering metrics. The standard approach is that DIAs define metrics that may prove useful. The DIAs activate those metrics that are expected to be useful to most clients in most environments. With distributed systems, the quantity of metrics is so great that the majority are actually delivered turned off. The client tailors distributed definitions by activating inactive metrics or de-activating metrics not applicable to specific client requirements.

While the standard approach of using pre-delivered metrics with user-tailoring works for existing DIAs, this approach has proven impractical for NSM Performance Management data. NSM Systems Performance allows different machines to selectively monitor different items (called resource subtypes). The standard, static DIA model would include everything NSM Systems Performance could monitor. The cost to manage a component built this way would be excessive given the possibility of monitoring thousands of systems, each with thousands of populated (or empty) metrics.

A unique paradigm is needed for processing NSM Systems Performance data. With this new model, the installation could create one or more components to process the NSM Systems Performance data and the components would be customized to the contents of the data itself. This approach allows the installation to configure MTI to suit the client's needs by creating components tailored to the input data. In other words, you tailor NSM to collect what you need and then MTI automates bringing that data into the CA MICS database structure.

This paradigm also resulted in the construction of technologies to provide for rapid implementation and deployment of MTI components. MTI provides the ability to create the CA MICS Component Generator statements. Using MTI, a site's component developer does not write the component statements but rather uses MTI capabilities.

As a whole, the CA MICS NSM Integration component, MTI, provides all the tools to perform resource management using CA MICS across the many distributed systems monitored with NSM Systems Performance.

8.3 Features

This section describes the key features of the MTI package which enable the interface between NSM and CA MICS:

1 - MTI Dictionary
2 - MTI Component Creator
3 - CA MICS NSM Data Transfer (MICSXFR)

Note that these features provide a highly automated yet very powerful creative tool for customizing your own components. Using MTI requires no new knowledge for the CA MICS Administrator. Without requiring you to specify multiple parameters, MTI exploits the unique capabilities of NSM to create, in a single step, full components, ready for integration into CA MICS. Yet, you retain the option to tailor these new components as you see fit.

8.3.1 MTI Dictionary

The MTI Dictionary forms the bridge to join the NSM Systems Performance and CA MICS architectures.

Mapping between Systems Performance and CA MICS (MTI Dictionary)

NSM Systems Performance uses machines, resources, and resource subtypes, all of which can have long names. CA MICS, on the other hand, uses SYSIDs, files, and elements with names no more than eight- characters long (with associated 40-character descriptions). By generating and assigning eight-character CA MICS names to map to the machines, resources, and resource subtypes in the NSM Performance Trend data, the MTI Dictionary provides the bridge that allows the Systems Performance data to be integrated into CA MICS. Section 8.6 addresses in greater detail the differences between Systems Performance entities and CA MICS entities and how MTI handles them.

Automatic Documentation (CA MICS Data Dictionary)

As part of the MTI Dictionary load process, CA MICS Data Dictionary builds entries to document every resource subtype encountered. During component generation, MTI assigns an alias linking previous dictionary entries to CA MICS database elements. Entries for each resource subtype contain information regarding the source, originating machine type, and CA MICS computation.

8.3.2 MTI Component Creator

Once the Data Dictionary is generated, MTI is able to create the definitions for an MTI Component based on the data that will be input. This feature makes it possible to create operational components for processing Performance Trend data without having any specialized training or experience.

Input data, inevitably, will change. As changes occur, the component generator can update the component definitions by adding and deleting files based on the input data.

MTI will integrate any installation customization to an MTI Component (e.g., file tailoring) into the changes made by the component generator. This feature of MTI eliminates the need to keep reapplying customization every time the component is updated.

One-Step MTI Component Creation

The technologies introduced with MTI allow the creation of components with a single job. No knowledge of CA MICS component definition syntax is necessary to create an MTI component. Based on the transfer facility (see Section 8.3.3) and the unique capabilities of NSM, CA MICS can read NSM Systems Performance output and, in a single step, create a full MTI component, ready for integration into CA MICS. Since it is not necessary to specify multiple parameters in this automated process, implementation time is dramatically reduced.

Yet, MTI components are highly configurable and you may opt to tailor the MTI components to suit your site's unique requirements.

CA MICS Accounting and Chargeback Support

All MTI Components automatically include the necessary exits to support CA MICS Accounting and Chargeback.

8.3.3 CA MICS NSM Data Transfer (MICSXFR)

MICSXFR is a CA-supplied utility that transfers NSM Performance Cube, Asset Management or NT Resource Accounting data to a data set on the host where CA MICS resides. MICSXFR allows you to specify which NSM extract data to supply to CA MICS for further processing. The utility converts the data to CSV format (if required), consolidates the data, and transfers it to the CA MICS host using FTP.

Section 4.7 of the Planning, Installation, Operation, and Maintenance Guide provides an installation checklist as well as some additional information about the utility. Once installed, the CA MICS Data Transfer online help provides information on how to use MICSXFR.

8.4 MTI Reporting

MTI includes several MICF inquiries that can provide detailed information about MTI components and the status of the MTI Dictionary.

Note: If any of the inquiries fails due to insufficient disk space, you may need to increase the space allocation for temporary SAS data sets. To do this, select MICF Options, Data Set Allocation Parameters, and increase the space allocation for the TEMPSAS and WORK entries.

These inquiries produce a number of reports which are covered in the following sections:

1 - MTILD1 - Defined Resource Types
2 - MTILD2 - Defined Resource Subtypes
3 - MTILD3 - Machine to ORGSYSID Table
4 - MTILD4 - MTI Defined Machine Types
5 - MTILD5 - MTI Defined Components
6 - MTILD6 - MTI Components Detail
7 - MTILD7 - MTI Component Element Detail

8.4.1 MTILD1 - Defined Resource Types

The MTILD1 report lists the NSM Performance Management resource types defined to MTI.

```
                              CA MICS NSM Interface:  Defined Resource Types                                  1

                                                    CA MICS


INQUIRY:  MTILD1                                                                                RUN DATE:  10JUN99


                MTI File                                      INSTANCE
                  ID       Resource Type                      FLAG      Key Lookup Name


                  _0A      NWLink SPX                          1        NWLINK_SPX
                  _0B      PhysicalDisk                        1        PHYSICALDISK
                  _0C      Processor                           1        PROCESSOR
                  _0D      Redirector                          0        REDIRECTOR
                  _0E      SQLServer                           0        SQLSERVER
                  _0F      SQLServer User Defined Counters     0        SQLSERVER_USER_DEFINED_COUNTERS
                  _0G      SQLServer-Licensing                 0        SQLSERVER_LICENSING
                  _0H      SQLServer-Locks                     0        SQLSERVER_LOCKS
                  _0I      SQLServer-Log                       1        SQLSERVER_LOG
                  _0J      SQLServer-Procedure Cache           0        SQLSERVER_PROCEDURE_CACHE
                  _0K      SQLServer-Users                     1        SQLSERVER_USERS
                  _0L      Server                              0        SERVER
                  _0M      Server Work Queues                  1        SERVER_WORK_QUEUES
                  _0N      NBT Connection                      1        NBT_CONNECTION
                  _0O      HpeNgent                            0        HPENGENT
                  _01      LogicalDisk                         1        LOGICALDISK
                  _02      Memory                              0        MEMORY
                  _03      Objects                             0        OBJECTS
                  _04      Paging File                         1        PAGING_FILE
                  _05      System                              0        SYSTEM
                  _06      Browser                             0        BROWSER
```

```
                    _07       Cache                          0        CACHE

                    _08       NWLink IPX                     1        NWLINK_IPX

                    _09       NWLink NetBIOS                 1        NWLINK_NETBIOS
```

Figure 8-1. MTI Defined Resource Types Report

8.4.2 MTILD2 - Defined Resource Subtypes

This report lists the NSM Systems Performance resource subtypes defined to MTI.

```
                                CA MICS NSM Interface: Resource Information                                  1

                                                    CA MICS

 INQUIRY:  MTILD2                                                                          RUN DATE:  10JUN99


---------------------------------------------------------- MTI File ID=_0A ------------------------------------------------------------


  CA MICS                                          CA MICS       Root                                  SAS         SAS
  ELEMENT                                          Element     Element for                   SAS       Variable    Variable
     ID       Resource Subtype Name                 Type       Computation    SAS format    Informat     length       Type


  _0ARAFRA    Frames/sec                             RAT        _0ARTFRA                                 8          N
  _0ARADAT    Datagrams/sec                          RAT        _0ARTDAT                                 8          N
  _0ACNRET    Connections No Retries                 SUM                                                 8          N
  _0ARAFBY    Frame Bytes/sec                        RAT        _0ARTFBY                                 8          N
  _0ACSTIM    Connection Session Timeouts            SUM                                                 8          N
  _0AERESP    Expirations Response                   SUM                                                 8          N
  _0ARADRE    Datagrams Received/sec                 RAT        _0ARTDRE                                 8          N
  _0ARAFRS    Frames Re-Sent/sec                     RAT        _0ARTFRS                                 8          N
  _0AEACK     Expirations Ack                        SUM                                                 8          N
  _0ADREMO    Disconnects Remote                     SUM                                                 8          N
```

_0AFNLIS	Failures No Listen	SUM		8	N
_0ARAFRE	Frames Received/sec	RAT	_0ARTFRE	8	N
_0AFADAP	Failures Adapter	SUM		8	N
_0AFFOUN	Failures Not Found	SUM		8	N
_0ARAPRE	Packets Received/sec	RAT	_0ARTPRE	8	N
_0ARADBY	Datagram Bytes/sec	RAT	_0ARTDBY	8	N
_0ARADSE	Datagrams Sent/sec	RAT	_0ARTDSE	8	N
_0ARAFB1	Frame Bytes Received/sec	RAT	_0ARTFB1	8	N
_0AWSAVE	Window Send Average	SUM		8	N
_0AFLINK	Failures Link	SUM		8	N
_0ARAFBS	Frame Bytes Sent/sec	RAT	_0ARTFBS	8	N
_0ACOPEN	Connections Open	SUM		8	N
_0ACCANC	Connections Canceled	SUM		8	N
_0ARAFBR	Frame Bytes Re-Sent/sec	RAT	_0ARTFBR	8	N
_0ARAFB2	Frame Bytes Rejected/sec	RAT	_0ARTFB2	8	N
_0ARADBS	Datagram Bytes Sent/sec	RAT	_0ARTDBS	8	N
_0APATIM	Piggyback Ack Timeouts	SUM		8	N
_0AFRLOC	Failures Resource Local	SUM		8	N
_0ARAPSE	Packets Sent/sec	RAT	_0ARTPSE	8	N
_0ARAPAQ	Piggyback Ack Queued/sec	RAT	_0ARTPAQ	8	N
_0ARAFR3	Frames Rejected/sec	RAT	_0ARTFR3	8	N
_0ARAFSE	Frames Sent/sec	RAT	_0ARTFSE	8	N
_0AFRREM	Failures Resource Remote	SUM		8	N
_0ARADBR	Datagram Bytes Received/sec	RAT	_0ARTDBR	8	N
_0ADLOCA	Disconnects Local	SUM		8	N
_0ACWRET	Connections With Retries	SUM		8	N
_0ARABTO	Bytes Total/sec	RAT	_0ARTBTO	8	N
_0ARAPAC	Packets/sec	RAT	_0ARTPAC	8	N

Figure 8-2. MTI Defined Resource Subtypes Report

8.4.3 MTILD3 - Machine to ORGSYSID Table

As part of the MTI Dictionary load process, the MTI Long-to-Short SYSID file (MTIL2S) is created. Each new machine encountered during the MTI dictionary load process is added to MTIL2S, then assigned an arbitrary unique ORGSYSID.

Currently, this table is for documentation purposes only and is not used by any MTI process. However, the installation is free to use this information to assist in assignment of the ORGSYSID in the cccSYSID exit (refer to Section 8.7.3).

This report shows all previously encountered Machines and the internally assigned ORGSYSID values as shown in the sample below.

```
           CA MICS NSM Interface: Machine to ORGSYSID                                    1

                                                CA MICS


INQUIRY:  MTILD3                                                                  RUN DATE:  10JUN99


       Original                 Originating
       Machine                     System       Machine
       Name                    Identification    Type     Machine Type(long)                    Datetime Added


       NTWIS01                     $000         NTW400I  Microsoft NT Workstation 4.00 Intel   10JUN99:15:40:58.17
       HPMSI01                     $001         HPX100   HP-UX 10.x                            10JUN99:15:40:58.17
       NTWIS02                     $002         NTW400I  Microsoft NT Workstation 4.00 Intel   10JUN99:15:40:58.17
       NTSIS01                     $003         NTS400I  Microsoft NT Server 4.00 Intel        10JUN99:15:40:58.17
       NTSIS02                     $004         NTS400I  Microsoft NT Server 4.00 Intel        10JUN99:15:40:58.17
       NTWIS03                     $005         NTW400I  Microsoft NT Workstation 4.00 Intel   10JUN99:15:40:58.17
       NTSIS03                     $006         NTS400I  Microsoft NT Server 4.00 Intel        10JUN99:15:40:58.17
       XAD1                        $007         MICS     MICS                                  10JUN99:15:40:58.17
       SUNIS01                     $008         SOL250S  SUN Solaris 2.5 Sparc                 10JUN99:15:40:58.17
       MICSUS                      $009         NTS400I  Microsoft NT Server 4.00 Intel        10JUN99:15:40:58.17
```

```
NTSIS04                    $010        NTS400I    Microsoft NT Server 4.00 Intel        10JUN99:15:40:58.17
```

Figure 8-3. MTI Machine to ORGSYSID Table

8.4.4 MTILD4 - MTI Defined Machine Types

The MTI Dictionary load process creates the Machine type table which contains the short and long descriptions of machine types.

MTILD4 produces a report showing each machine type encountered with the corresponding long, descriptive name.

```
                                  CA MICS NSM Interface: Machine Types                                    1

                                                CA MICS


INQUIRY:  MTILD4                                                                        RUN DATE:  11JUN99


                          Machine    Machine                                    Datetime
                           Type      Type(long)                                   Added


                          NTW400I    Microsoft NT Workstation 4.00 Intel    10JUN99:15:40:58.33
                          HPX100     HP-UX 10.x                             10JUN99:15:40:58.33
                          NTS400I    Microsoft NT Server 4.00 Intel         10JUN99:15:40:58.33
                          MICS       MICS                                   10JUN99:15:40:58.33
```

Figure 8-4. MTI Defined Machine Types Report

8.4.5 MTILD5 - Defined MTI Components

This report lists the MTI components you have created.

```
                                        Defined MTI Components                                            1

                                                CA MICS
```

```
INQUIRY:  MTILD5                                                                RUN DATE:  11JUN99

                                    CA MICS       CA MICS        CA MICS
                                   Component     Component     Information
                                     Id        Step Number       Area        Label

                                     UXX           150           UXX         MTI
```

Figure 8-5. Defined MTI Components Report

8.4.6 MTILD6 - MTI Components Detail

This report lists the MTI components and the files they populate.

```
                                        Defined MTI Components                                  1
                                              CA MICS

INQUIRY:  MTILD6                                                                RUN DATE:  11JUN99

                                  CA MICS       CA MICS      CA MICS
                                 Component     Information     File
                                   Id            Area          Id       Label

                                   UXX           UXX           UXB      MEMORY
                                                               UXC      PAGING
                                                               UXD      SYSTEM
                                                               UXA      LOGICALDISK
```

Figure 8-6. MTI Components Detail Report

8.4.7 MTILD7 - MTI Component Element Detail

For each MTI Component, MTILD7 lists the information areas, files, and data elements.

MTI Component: Element Detail 1
CA MICS

INQUIRY: MTILD7 RUN DATE: 11JUN99

CA MICS Component Id	CA MICS Information Area	CA MICS File Id	CA MICS Element Name	CA MICS Data Dictionary Alias	Label	CA MICS Data Element Type	CA MICS Variable Type
UXX	UXX	UXA	UXAFMEGA	_01FMEGA	Free Megabytes	NUMERIC	RETAINED
			MONTH	.	Month of Year	NUMERIC	RETAINED
			DAYNAME	.	Name Of Day Of Week	ALPHA	RETAINED
			DAY	.	Day of Month	NUMERIC	RETAINED
			STARTTS	.	Start Time Stamp	DATETIME	MINIMUM
			ENDTS	.	End Time Stamp	DATETIME	MAXIMUM
			ORGSYSID	.	Originating System Identification	ALPHA	RETAINED
			ZONE	.	Time Zone	ALPHA	RETAINED
			INTERVLS	.	Number of Recording Intervals	NUMERIC	ACCUMULATED
			INSTANCE	INSTANCE	Instance of Subtype	ALPHA	RETAINED
			MACHINE	.	Machine Name	ALPHA	RETAINED
			SYSID	.	System Identifier	ALPHA	RETAINED
			DURATION	.	Recording Interval Time	TIME	ACCUMULATED
			YEAR	.	Year of Century	NUMERIC	RETAINED
			HOUR	.	Hour Of Day	NUMERIC	RETAINED
UXX	UXX	UXB	UXBRAPAG	_02RAPAG	Pages/sec	NUMERIC	COMPUTED
			UXBRAPFA	_02RAPFA	Page Faults/sec	NUMERIC	COMPUTED
			DAY	.	Day of Month	NUMERIC	RETAINED
			UXBRAPWR	_02RAPWR	Page Writes/sec	NUMERIC	COMPUTED
			SYSID	.	System Identifier	ALPHA	RETAINED
			MACHINE	.	Machine Name	ALPHA	RETAINED
			MONTH	.	Month of Year	NUMERIC	RETAINED
			ORGSYSID	.	Originating System Identification	ALPHA	RETAINED
			STARTTS	.	Start Time Stamp	DATETIME	MINIMUM
			UXBRTPAG	_02RAPAG	Root of Pages/sec	NUMERIC	ACCUMULATED
			HOUR	.	Hour Of Day	NUMERIC	RETAINED
			UXBRTPFA	_02RAPFA	Root of Page Faults/sec	NUMERIC	ACCUMULATED
			DURATION	.	Recording Interval Time	TIME	ACCUMULATED
			UXBRTPRE	_02RAPRE	Root of Page Reads/sec	NUMERIC	ACCUMULATED
			UXBRTPIN	_02RAPIN	Root of Pages Input/sec	NUMERIC	ACCUMULATED
			ENDTS	.	End Time Stamp	DATETIME	MAXIMUM
			UXBRTWCO	_02RAWCO	Root of Write Copies/sec	NUMERIC	ACCUMULATED
			INTERVLS	.	Number of Recording Intervals	NUMERIC	ACCUMULATED
			UXBRTTFA	_02RATFA	Root of Transition Faults/sec	NUMERIC	ACCUMULATED
			ZONE	.	Time Zone	ALPHA	RETAINED
			UXBRAWCO	_02RAWCO	Write Copies/sec	NUMERIC	COMPUTED
			UXBRATFA	_02RATFA	Transition Faults/sec	NUMERIC	COMPUTED
			YEAR	.	Year of Century	NUMERIC	RETAINED
			UXBRTPOU	_02RAPOU	Root of Pages Output/sec	NUMERIC	ACCUMULATED
			UXBRAPRE	_02RAPRE	Page Reads/sec	NUMERIC	COMPUTED

Figure 8-7. MTI Component Element Detail Report

8.5 Integration Checklists

This section provides the checklists you will need to activate MTI, create and update MTI components, and perform MTI utility functions:

1 - Activating MTI in the CA MICS Complex
2 - Creating an MTI Component
3 - Updating an existing MTI Component
4 - Recovering the MTI Dictionary

The jobs used in these checklists are documented in greater detail in Section 8.8 of this guide. Appendix B contains a discussion of utility control statement syntax used in some of these jobs.

8.5.1 Activating MTI in the CA MICS Complex

This checklist is only used once, to activate MTI in your complex. Follow these steps if CA MICS is already in operation at your site and you are going to be integrating NSM data into CA MICS for the first time.

If, at this time, you are in the process of installing CA MICS at your site, the steps for activating MTI are included in the CA MICS installation checklists.

Once you have activated MTI, you will use the checklist in Section 8.5.2 to create MTI components.

___1. Edit sharedprefix.MICS.PARMS(JCLGENUC).

Delete all lines, then copy sharedprefix.MICS.PARMS(JCL$0MTI) into this member. Save the member and exit.

___2. Edit sharedprefix.MICS.CNTL(JCLGENUC).

Submit the job in sharedprefix.MICS.CNTL(JCLGENUC).

Ensure that there are no error messages in MICSLOG or SYSTSPRT, that the MICSLOG contains the normal termination message, BAS10999, and that the job completes with a condition code of zero.

```
___3. Run MTICGEN.

      Generate the product by submitting the job in
      sharedprefix.MICS.CNTL(MTICGEN).

      Ensure that there are no error messages in ISPLOG and
      MICSLOG; that MICSLOG contains the normal termination
      messages, BAS20999I; and that the job completes with a
      condition code of zero.

___4. Initialize the Dictionary.

      Submit the job in sharedprefix.MICS.CNTL(DICINIT).

      Ensure that there are no error messages in the MICSLOG;
      that the MICSLOG contains the normal termination
      message, MTI02399I; and that the job completes with a
      condition code of zero.

  ************************************************************
  *  NOTE:  In order to complete MTI installation it is     *
  *         necessary to add MTI to the primary unit.  This *
  *         will provide backup and restore capability for  *
  *         the MTI Dictionary.                             *
  *                                                         *
  *         Perform steps 5 through 8 in the Primary        *
  *         database only.                                  *
  ************************************************************

___5. Edit prefix.MICS.PARMS(JCLDEF).

      Add your MTI component's three letter identifier to the
      COMPONENTS statement.  Save the changes.

___6. Edit prefix.MICS.PARMS(JCLGENU) so that it contains
      a single line that reads:

      COPYLIBU
```

___7. Submit the job in prefix.MICS.CNTL(JCLGENU).

Ensure that there are no error messages in MICSLOG or SYSTSPRT, that the MICSLOG contains the normal termination message, BAS10999, and that the job completes with a condition code of zero.

___8. Submit the job in prefix.MICS.CNTL(COPYLIBU).

Ensure that there are no error messages in MICSLOG and that the job completes with a condition code of zero.

___9. See section 4.7 of the PIOM for instructions for installing the CA MICS Data Transfer utility on your workstation.

8.5.2 Creating an MTI Component

This checklist is used to create an MTI component from NSM Systems Performance data. This data is transferred to the z/OS system using the CA MICS Data Transfer program (see PIOM Section 4.7).

Before creating an MTI component, the input data should be configured to contain the resource subtypes that will be collected by MTI.

Use this checklist one time for each MTI component.

```
*********************************************************
*                                                       *
*    ! ! !  W A R N I N G  ! ! !                        *
*                                                       *
*  IF YOU ARE GENERATING MULTIPLE MTI COMPONENTS, YOU   *
*  MUST COMPLETE THIS CHECKLIST FOR EACH MTI COMPONENT  *
*  BEFORE GENERATING A NEW COMPONENT.                   *
*                                                       *
*  The MTICREAT process does not add file definitions   *
*  to CA MICS.  This is only done AFTER the cccCGEN is  *
*  completed.                                           *
*                                                       *
*  To avoid unpredictable results when generating       *
*  multiple MTI components, the cccCGEN for the first   *
*  component MUST be run before running MTICREAT for    *
*  the next component.                                  *
*                                                       *
*********************************************************
```

___1. Select a name for the new MTI component. Component names are three characters in length, start with a 'U' or 'Z', and cannot be an existing component, information area, or file name.

___2. Select the input that will be used by the new component. Record the data set names. This information will be used in the next step.

NOTE: The INPUTCSV DD does NOT support concatenated datasets.

NSM data used by MTI is supplied via MICSXFR (CA MICS Data Transfer). Section 4.7 of the PIOM provides instructions for installing MICSXFR. Once the utility is installed, refer to its online help system for information on using the MICSXFR.

___3. Edit sharedprefix.MICS.CNTL(MTICREAT).

```
Specify the name of the new MTI component (selected in
Step 1) in the EXEC statement SYSPARM= parameter.

Optionally, you can specify the step number for the new
MTI component by adding it after the component name
(e.g., SYSPARM='UZZ 151').  The step number can be any
number between 150 through 190 that has not already
been used.  If there is a concern about eventually
using most or all of the numbers in this range, one
should define multiple information areas within a user
component wherever it is possible.

Update the INPUTCSV DD with the input data set
information from Step 2.

NOTE:  The INPUTCSV DD does NOT support concatenated
       datasets.

NOTE:  The MTICREAT job can be very resource intensive.
       Make sure that MTICREAT runs in a jobclass that
       will allow one or more minutes of CPU time.

Submit the job and make sure that all steps complete
with a condition code of zero and there are no error
messages in any of the MICSLOGs.

*********************************************************
*                                                       *
*  If the MTICREAT job fails in the JCLGEN step, it     *
*  cannot be restarted because of CA MICS internal      *
*  locking mechanisms.  If this situation occurs, a     *
*  JCLGENUC must be manually run to complete the        *
*  MTICREAT process.                                    *
*                                                       *
*  Edit sharedprefix.MICS.PARMS(JCLGENUC) and remove    *
*  all lines.  Copy sharedprefix.MICS.PARMS(JCL$0ccc)   *
*  where ccc is the component name you selected into    *
*  this member.  Save the member and exit.              *
*                                                       *
*  Edit sharedprefix.MICS.CNTL(JCLGENUC).  Submit       *
*  the job and ensure it completes with a return        *
*  code of zero.                                        *
*                                                       *
*********************************************************
```

```
___4. (Optional) Run Shared MICF Inquiry MTILD3.
      This inquiry provides a list of all machine names
      encountered to date.  This information is useful
      for creating entries in prefix.MICS.PARMS(SYSID) and
      the optional prefix.MICS.PARMS(cccSYSID) exit.

___5. Proceed with the checklist in PIOM Section 3.8.2 for
      installing the new component into an existing database
      unit, or the checklist in PIOM Section 3.8.3 for
      installing the new component into a new database unit.
```

8.5.3 Updating an existing MTI Component

One of the design objectives of MTI was to allow the component to be customized to the input data. The definition of an MTI component may need to be changed if the contents of the input data changes.

MTI provides a utility, called MTIGENER, to update the definition of an MTI component (stored in sharedprefix.MICS.GENLIB(cccGENIN)). MTIGENER will scan a component's input data, create CA MICS file definitions for each new resource encountered, and then update sharedprefix.MICS.GENLIB(cccGENIN).

Note: MTIGENER preserves any changes you have made to an MTI component (e.g., file activation/deactivation) in sharedprefix.MICS.GENLIB(cccGENIN). However, sequence numbers are not retained and are removed.

During component generation (step 2), the INPUTCSV DD should contain the input data of the component. If INPUTCSV is not in the JCL, MTI will create files with all resource subtypes of a given resource (based upon the MTI Dictionary). This may result in files with much more data than you really want. The INPUTCSV file is used to create a subset based upon the resource subtypes in the input data.

```
The default INPUTCSV definitions are stored in
sharedprefix.MICS.PARMS(cccINPUT) which was created during
the MTICREAT process.  This member is initialized with data
set names from the INPUTCSV DD at the time MTICREAT was run.


___1. (Optional) If you need to modify the INPUTCSV data set
      names or concatenate additional input files to the
      INPUTCSV DD, then

      Edit sharedprefix.MICS.PARMS(cccINPUT)

      and specify the necessary data set names on the
      INPUTCSV DD statement.  Save the member.

___2. (Optional) If you modified the
      sharedprefix.MICS.PARMS(cccINPUT) member, then

      Edit sharedprefix.MICS.PARMS(JCLGENUC)

      and copy the contents of
      sharedprefix.MICS.PARMS(JCL$0ccc) into this member
      (JCLGENUC).

      Submit the job in sharedprefix.MICS.CNTL(JCLGENUC).

      Ensure that there are no error messages in MICSLOG or
      SYSTSPRT, that the MICSLOG contains the normal
      termination message, BAS10999I, and that the job
      completes with a condition code of zero.

___3. (Optional) Edit sharedprefix.MICS.CNTL(cccDSCAN)

      where ccc is the MTI component name.

      This job will compare the input file(s) with the MTI
      Dictionary and report on any resource types and
      subtypes that are not defined in the current MTI
      Dictionary.
```

```
      Submit sharedprefix.MICS.CNTL(cccDSCAN).

      Ensure that there are no error messages in the MICSLOG;
      that the MICSLOG contains the normal termination
      message, MTI02399I; and that the job completes with a
      condition code of zero.

      NOTE:  It is possible there may be no changes to the
             configuration.  In this case, proceed directly
             to Step 5.

************************************************************
*                                                          *
*  If new resources have been added, you must edit and     *
*  submit the cccDLOAD job in Step 4.  If you do not       *
*  perform this step and new resources have been added,    *
*  the cccUPDAT job in Step 7 will abend.                  *
*                                                          *
************************************************************

___4. Edit sharedprefix.MICS.CNTL(cccDLOAD)

      where ccc is the MTI component name.

      This job updates the MTI dictionary with any new
      resources as identified in the cccDSCAN job in the
      previous step.

      Executing this job will not only eliminate the
      MTI02108W and/or MTI02110W message in your MTI DAILY
      job step, but also make these resources and resource
      subtypes available to process in your MTI DAILY
      jobstep, if desired.  Even if you do not wish to
      process these resources in your MTI DAILY, you may
      still execute this job to eliminate the warning
      messages with no detrimental effects.
```

```
      NOTE:  These new resources will NOT be processed in
             your MTI DAILY jobstep until you complete the
             remainder of this checklist.

      Ensure the INPUTCSV DD points to the correct input
      data sets for this component.

      If you wish to suppress the MTI02108W and/or
      MTI02110W message in your MTI DAILY job step AND
      enable these resources to be processed in your MTI
      DAILY jobstep, then

      Submit the job in sharedprefix.MICS.CNTL(cccDLOAD).

      Ensure that there are no error messages in the MICSLOG;
      that the MICSLOG contains the normal termination
      message, MTI02399I; and that the job completes with a
      condition code of zero.

___5. Create a backup of sharedprefix.MICS.GENLIB(cccGENIN).

      cccUPDAT (step 7) does NOT make a backup of
      sharedprefix.MICS.GENLIB(cccGENIN).  It is highly
      recommended that you make a copy of this member for
      recovery purposes.  For example, cccGENIN could be
      copied to member cccGENBK.

      WARNING:  WITHOUT A BACKUP COPY THERE IS NO WAY TO
                RECOVER IF cccGENIN IS DAMAGED.

___6. (Optional) Edit sharedprefix.MICS.CNTL(cccUPSCN)

      where ccc is the MTI component name.

      This job produces a report of the updates that will
      take place should you run the cccUPDAT job in the
      subsequent step.  This allows you to review the
      changes before actually implementing them.

      Ensure the INPUTCSV DD points to the correct input
      data sets for this component.

      Submit the job in sharedprefix.MICS.CNTL(cccUPSCN).

      Ensure that there are no error messages in the MICSLOG;
      that the MICSLOG contains the normal termination
      message, MTI02399I; and that the job completes with a
      condition code of zero.
```

```
___7. Edit sharedprefix.MICS.CNTL(cccUPDAT)

      where ccc is the MTI component name.

      This job customizes your
      sharedprefix.MICS.GENLIB(cccGENIN) with the new
      resources identified in the cccUPSCN job in step 6
      above.

      Ensure the INPUTCSV DD points to the correct input
      data sets for this component.

      Submit the job in sharedprefix.MICS.CNTL(cccUPDAT).

      Ensure that there are no error messages in the MICSLOG;
      that the MICSLOG contains the normal termination
      message, MTI02399I; and that the job completes with a
      condition code of zero.

      ******************************************************
      *  It is possible for there to be no changes to the  *
      *  configuration.  In this case, cccUPDAT will       *
      *  terminate with a U998 ABEND and a message stating *
      *  that there are no new resources to add to the     *
      *  component.                                        *
      *                                                    *
      *  However, it is still necessary to execute the     *
      *  cccCGEN in step 9 as new elements may have been   *
      *  added to existing files.                          *
      *                                                    *
      *  Additionally, if the cccDLOAD job was not run to  *
      *  define these new resources to the MTI Dictionary, *
      *  the cccUPDAT job will abend and issue messages    *
      *  stating that the cccDLOAD job must be run prior   *
      *  to executing the cccUPDAT job.                    *
      ******************************************************

___8. (Optional) Make manual changes to cccGENIN

      Edit sharedprefix.MICS.GENLIB(cccGENIN) and make
      any changes as necessary.

      For example, if you do not wish to use multiple work
      file support with this MTI component, then insert the

      MSIOPTS NOMULTWORK
```

```
      statement before the first FILE statement in your
      MTIGENIN.  This may also be done on a FILE by FILE
      basis rather than globally.  Refer to section 4.2.4.3.1
      in this guide for more information on the MSIOPTS
      statement.

___9. Run cccCGEN.

      Edit sharedprefix.MICS.CNTL(cccCGEN).
      Optionally, update the INPUTCSV DD with the correct
      input data sets for this component.

      Generate the product by submitting the job in
      sharedprefix.MICS.CNTL(cccCGEN).

      Ensure that there are no error messages in ISPLOG and
      MICSLOG; that MICSLOG contains the normal termination
      messages, BAS20999I; and that the job completes with a
      condition code of zero.

      ********************************************************
      *  It is possible for there to be no changes to the    *
      *  configuration.  In this case, cccUPDAT will         *
      *  terminate with a U998 ABEND and a message stating   *
      *  that there are no new resources to add to the       *
      *  component.                                          *
      *                                                      *
      *  Since there may be new data elements added to       *
      *  existing files, the cccCGEN is still required.      *
      *                                                      *
      *  If there were no changes to file activation, Audit  *
      *  Archive, History Weekly, or History Monthly         *
      *  status, the remainder of this checklist can be      *
      *  skipped.                                            *
      ********************************************************

  ************************************************************
  *  Repeat step(s) 10-18 for each unit database that        *
  *  contains the ccc MTI component.                         *
  ************************************************************

__10. Edit prefix.MICS.PARMS(JCLGENU) so that it contains a
      single line that reads:

      COPYLIBU GDGSGEN

      Submit the job in prefix.MICS.CNTL(JCLGENU).
```

```
      Ensure that there are no error messages in MICSLOG or
      SYSTSPRT; that the MICSLOG contains the normal
      termination message, BAS10999I; and that the job
      completes with a condition code of zero.

__11. Submit the job in prefix.MICS.CNTL(COPYLIBU).

      Ensure that there are no error messages in MICSLOG and
      that the job completes with a condition code of zero.

__12. Edit prefix.MICS.PARMS(cccOPS).

      Refer to section 8.7 of the System Modification Guide
      on coding parameters for cccOPS.

__13. Submit the job in prefix.MICS.CNTL(cccPGEN).

      Ensure that there are no error messages in MICSLOG and
      that the job completes with a condition code of zero.

__14. Edit prefix.MICS.PARMS(DBMODEL).  Find the FILE
      statement(s) that identify the file(s) added or changed
      by this maintenance.

      Modify the number of cycles to be retained in each FILE
      statement to suit your installation's standards.

__15. Submit the job in prefix.MICS.CNTL(CYCLEGEN) to
      generate file cycle retention limits.

      Ensure that there are no error messages in MICSLOG,
      that MICSLOG contains the normal termination message,
      BAS11999I, and that the job completes with a condition
      code of zero.

__16. If audit or history files are used in this unit
      database, submit the job in prefix.MICS.CNTL(GDGSGEN).
      If allocated, ensure that there are no error messages
      in MICSLOG.

      The last IDCAMS step may complete with a non-zero
      return code (either an 8 or a 12) if
      audit/history/backup files currently exist.  Therefore,
      you must review the SYSPRINT SYSOUT DD and make sure
      that the GDG entries for newly added files have been
      created successfully.
```

```
__17. Edit prefix.MICS.PARMS(INPUTccc).

      Verify that the names of all data sets are specified
      on the INPUTccc DD.

__18. Regenerate operational JCL.

      Submit the job in prefix.MICS.CNTL(JCLGEND).

      Ensure that there are no error messages in MICSLOG or
      SYSTSPRT, that the MICSLOG contains the normal
      termination message, BAS10999, and that the job
      completes with a condition code of zero.

  ************************************************************
  * Repeat the above step(s) for each unit database that    *
  * contains ccc.                                           *
  ************************************************************
```

8.5.4 Recovering the MTI Dictionary

This section addresses recovery in the unlikely event of a catastrophic failure resulting in the total destruction of the MTI Dictionary through the use of the MTIDUTIL utility (discussed in detail in Section 8.8). Although chances of such a loss occurring are slim, the MTI Dictionary was constructed so that it can be rebuilt with minimal impact to existing MTI components.

Note that the MTI Dictionary is stored in the sharedprefix.MICS.TABLES data set and is backed up with the Primary CA MICS database unit. This checklist should be used ONLY if the Primary database backup is unavailable.

```
___1. Reinitialize the MTI Dictionary.

      Edit sharedprefix.MICS.CNTL(MTIDUTIL) and delete all
      lines following the MTISYSIN DD.  Add the following
      after the MTISYSIN DD:

      %DICINIT(FORCE=YES)
```

```
*******************************************************
*                                                     *
*     ! ! !  C A U T I O N  ! ! !                     *
*                                                     *
*  Running a DICINIT with the FORCE=YES option will   *
*  completely destroy the existing MTI Dictionary!    *
*  All existing MTI configuration information will    *
*  be lost!                                           *
*                                                     *
*******************************************************
```

Submit the job in prefix.MICS.CNTL(MTIDUTIL).

Ensure that there are no error messages in the MICSLOG; that the MICSLOG contains the normal termination message, MTI02399I; and that the job completes with a condition code of zero.

___2. Reload the METRICS / MTI Dictionary.

Edit sharedprefix.MICS.CNTL(DICLOAD).

CA MICS shipped an HFS CSV file representing certain pre-packaged Metrics data that were created for certain machine types. This file is called the "Standardized Metrics Update File".

All Standardized Metrics Update File data previously loaded by way of the INPUTMTX DD will also have to be reloaded. The Metrics file will need to be specified as a data set name on the INPUTMTX DD.

On the INPUTMTX DD, add the data set name of the Standardized Metrics Update file.

On the INPUTCSV DD, add the data set names of all the input data sets for ALL MTI components. Concatenated data sets are allowed. You should manually check the daily jobs, cccCGEN, cccUPDAT, DICLOAD, and DICSCAN in order to locate all data sets.

In order to properly rebuild the MTI Dictionary, all data sets previously used to build the dictionary must be reloaded.

```
      Failure to do so can give unpredictable results.

      Submit the job in sharedprefix.MICS.CNTL(DICLOAD).

      Ensure that there are no error messages in the MICSLOG;
      that the MICSLOG contains the normal termination
      message, MTI02399I; and that the job completes with a
      condition code of zero.

___4. Delete all existing MTI CA MICS Data Dictionary
      entries.

      Using the ISPF Library utility (Options 3.1),
      delete all members starting with an underscore(_) in
      sharedprefix.MICS.DIC.TEXT.

___5. Compress sharedprefix.MICS.DIC.TEXT.

      In order to make room for the new MTI Data Dictionary
      members, you should compress
      sharedprefix.MICS.DIC.TEXT.  You can use ISPF Library
      Utilities (option 3.1), or IEBCOPY.

___6. Rebuild the MTI CA MICS Data Dictionary entries.

      Edit sharedprefix.MICS.CNTL(DICDSHEL).  Make sure the
      COMMIT=YES parameter is specified on the DICSHELL
      statement.

      Submit the job in sharedprefix.MICS.CNTL(DICDSHEL).

      Ensure that there are no error messages in the MICSLOG;
      that the MICSLOG contains the normal termination
      message, MTI02399I; and that the job completes with a
      condition code of zero.
```

```
___7. Regenerate each MTI Component.

      ******************************************************
      *                                                    *
      *  Repeat this step for each MTI component. To       *
      *  identify MTI components, run the MICF Shared      *
      *  Inquiry MTILD5 (Defined MTI Components).          *
      *                                                    *
      ******************************************************

      Edit sharedprefix.MICS.CNTL(cccCGEN).  Make sure that
      the INPUTCSV DD contains the proper data sets.  Verify
      with the daily jobs in all units containing MTI
      component ccc.

      Submit the job in sharedprefix.MICS.CNTL(cccCGEN).

      Ensure that there are no error messages in ISPLOG and
      MICSLOG; that MICSLOG contains the normal termination
      messages, BAS20999I; and that the job completes with a
      condition code of zero.
```

8.6 Considerations

The CA MICS Data Transfer utility, MTI Dictionary, and the Component Generator make the CA MICS NSM Integration interface possible despite the significant differences between NSM and CA MICS environments, data types, and data packaging.

8.6.1 SYSIDs vs. Machine IDs

One area of variation between CA MICS and NSM Systems Performance is the difference in length between the original system identifier (ORGSYSID) and the machine ID. CA MICS limits ORGSYSIDs to four characters, whereas allows machines to have names greater than four characters.

The ORGSYSID is significant because it, in combination with the resource type, uniquely identifies the high and low timestamp of the input data for CA MICS checkpoint processing, which prevents the introduction of duplicate data into the CA MICS database.

Part of the MTI Dictionary load process updates the MTI Long-to-Short SYSID file (MTIL2S). This file contains all machine names encountered and an internally generated unique ORGSYSID. Note that this file is for documentation purposes only and is not used by MTI.

By default, MTI assigns the first four characters of the machine name to ORGSYSID. In the event that ORGSYSID does not contain sufficient uniqueness (i.e. several machines start with the same four characters) MTI provides the cccSYSID exit for you to reassign the long machine name to the four character ORGSYSID.

Refer to Section 8.7.3 for details on how to implement cccSYSID exit.

While it is best to assign a unique ORGSYSID to each machine, the cost to manage a large number of systems can be overwhelming. The CA MICS Checkpoint contains one entry per resource type, per ORGSYSID. In very large configurations, this combination may exhaust the maximum of 999 checkpoint entries for a unit database. To reduce demands on CA MICS Checkpoint, the cccSYSID exit can be used to cluster machines into a single ORGSYSID.

This configuration allows multiple machines to share the same CA MICS Checkpoint information when CA MICS database update encounters no data for MACHINE_A and data for MACHINE_B.

Note, however, that this configuration might result in unexpected data loss. Consider the following example: MACHINE_A and MACHINE_B are assigned the same ORGSYSID MACH. On a given day the CA MICS database update encounters data from MACHINE_B but not from MACHINE_A. CA MICS Checkpoint is updated to show the high timestamp for ORGSYSID MACH to be yesterday. MACH is updated because cccSYSID assigns its machine name (MACHINE_B) to MACH. The next day, the input data sets for MACHINE_A contain data from the past two days whereas the input data sets for MACHINE_B contain data just for the previous day. The oldest data for MACHINE_A (two days ago) will be dropped because the checkpoint indicates that this data has already been processed based upon ORGSYSID of MACH, assigned to both MACHINE_A and MACHINE_B.

Finally, MTI has been designed so that the machine is inserted as part of the sort sequence ahead of the logical SYSID, guaranteeing that the original machine is always available for reporting purposes regardless of the ORGSYSID.

8.6.2 Data Types

During the MTI Dictionary load process (DICLOAD), each resource subtype from the input data is scanned to determine its data type. This scanning process identifies a resource subtype as a Minimum, Maximum, Accumulated, Rate (quantity over time), Average, Percentage, or Unknown and saves this information in the MTI Dictionary. When the CA MICS Component Generator is run, it uses this resource subtype information to set the data type of the assigned CA MICS data element.

Resource-subtypes with a data type of Accumulated, Minimum and Maximum are considered the same as the CA MICS data types. Resource sub-types with a data type of Rate, Average and Percent types are unique in that it is implied that some formula must be used to derive the associated CA MICS data elements. In most cases, the elements required to compute this value are not available and to simply assign a CA MICS data type of retained or accumulated would not allow the assigned CA MICS elements to be properly computed. This can result in severe inaccuracies in the data.

To facilitate recomputation of CA MICS data elements with a data type of Rate, Average, and Percent data types the CA MICS Component Generator performs the following special actions:

- CA MICS Component Generator will first create a root element. A root element is an accumulated element that is used in the computation of a CA MICS element.

- Next, the component generator will create the initialization code for the root element that will 'de-compute' the CA MICS data element to obtain the original quantity. For example, rate data elements are multiplied by DURATION.

- Finally, the component generator internally generates code to perform the actual computation.

Whenever MTI cannot determine what type a particular resource-subtype is, it is given a type of Unknown. The CA MICS Component Generator will treat Unknown types as Accumulated.

For Metrics load carried out through the use of the INPUTMTX DD, the data type is taken verbatim from the data type found in the record for the input data set specified on the INPUTMTX DD. This data set will be the latest Standardized Metrics Update file.

8.7 MTI Unit Level Parameters

Components created using MTI will add three members to prefix.MICS.PARMS. They are described in the following sections:

1 - Component Processing Options (cccOPS)
2 - INPUTccc
3 – cccSYSID

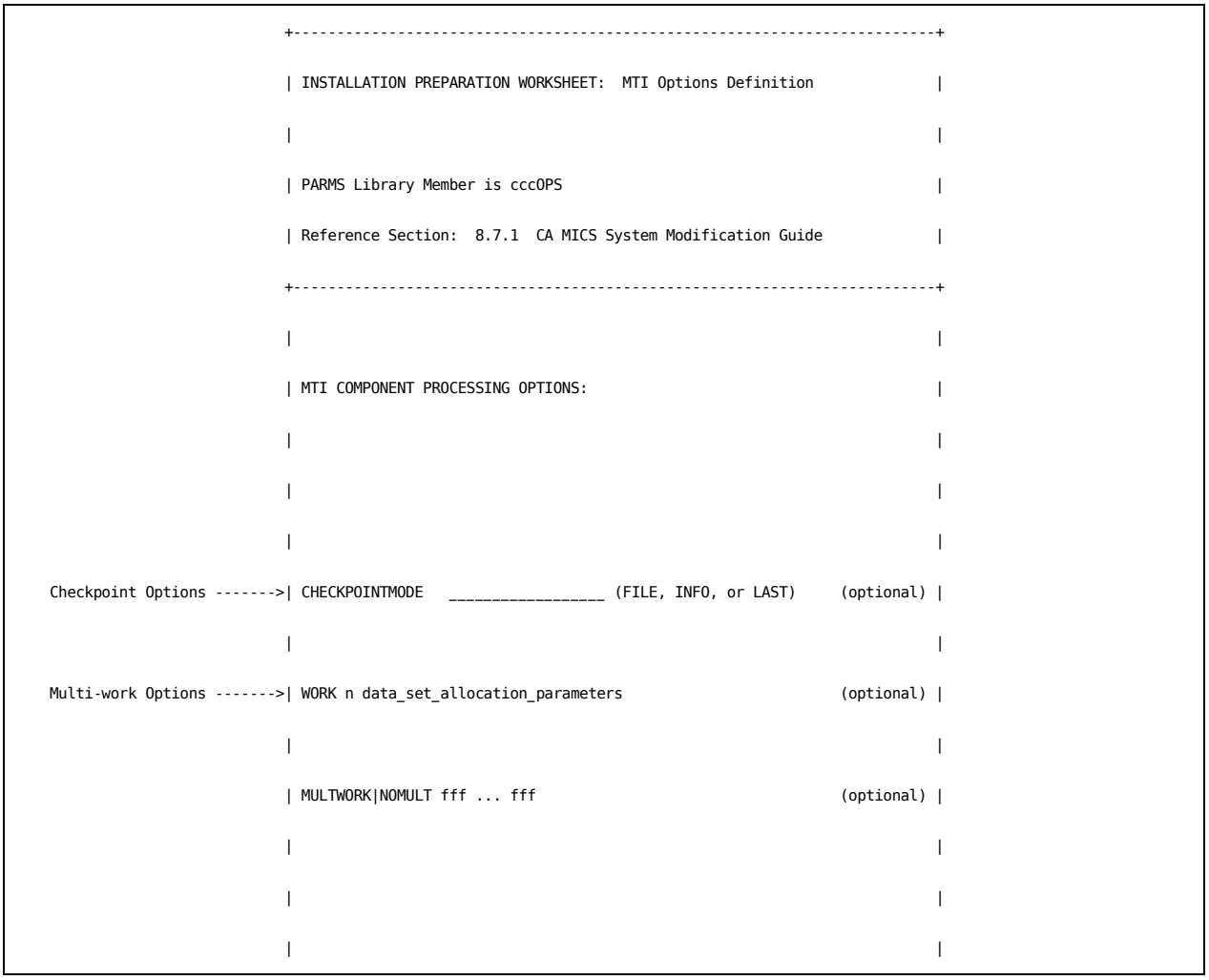
8.7.1 Component Processing Options (cccOPS)

Each component using MTI has its own set of parameters defined in prefix.MICS.PARMS(cccOPS). These parameters control the specification of work files and checkpoint processing.

cccOPS contains the following optional statements:

- CHECKPOINTMODE Statement (optional)
- Work File Parameters (optional)

Figure 8-8 shows the Processing Options Definition Worksheet for an component using MTI. Its statements are described after the worksheet.



```
                                +-------------------------------------------------------------------+
```

Figure 8-8. MTI Component Processing Options Definition Worksheet

```
                        Check Point Processing Options (CHECKPOINTMODE)

                          Normally, CA MICS Checkpoint processing will cause a U300
                          ABEND if there are less than ten new records input during
                          daily processing of a component.  Since components using
                          MTI can support multiple input files, MTI provides options
                          to control when checkpoint processing occurs.


                          STATEMENT SYNTAX

                          CHECKPOINTMODE option
                            where options are LAST, INFO, or FILE.
                            (The default is FILE.)

                          STATEMENT OPTION DEFINITIONS

                          LAST - At least 10 observations must exist across all files
                                 in the MTI component (total of 10 observations).

                          INFO - Each Information area must have at least ten
                                 observations across all files in the information
                                 area.  (total of 10 observations).

                          FILE - Each file must have at least 10 observations.

                          The CHECKPOINTMODE option has no effect if SYSPARM=NODATA
                          is specified on the DAILY job DAYnnn step EXEC statement.
```

```
Work File Parameters

  All MTI components fully support the standardized multiple
  workfile option.

  Note:  MTI components do not support incremental update or
         internal step restart.


WORK
----

This statement is optional.  It enables sites experiencing
either SAS WORK space allocation problems or out of work
space conditions during DAYnnn or INCRnnn (where nnn is the
job step number), daily or incremental update processing, to
allocate multiple WORK files.

You can allocate multiple WORK files for use during the daily
and/or incremental update job step.  The maximum number of
WORK files you can allocate varies by product.  These
additional work files are used in conjunction with the single
work data set allocated by default using the JCLDEF
parameters WORKUNIT and WORKSPACE.

Because the individual space allocation requirement for each
WORK file is typically much smaller, it is more likely to be
satisfied.

To take advantage of multiple WORK files support, edit
prefix.MICS.PARMS(cccOPS) and insert a WORK statement as
shown below:

WORK n data_set_allocation_parameters

where n is the number of WORK data sets
```

```
data_set_allocation_parameters is one or more data
        set allocation parameters (for example, STORCLAS or
        SPACE) separated by spaces.

You can also specify the WORK parameter as the following:

WORK n XXX pppp ssss

where:

  n    is the number of WORK data sets
  XXX  is TRK or CYL
  pppp is the primary allocation
  ssss is the secondary allocation


Note:  When allocating any number of SAS WORK data sets, be
aware that one additional SAS WORK data set is automatically
allocated to facilitate sorting.  For example, if you
allocate six SAS WORK data sets, you will actually get seven.

If you omit the data_set_allocation_parameters or the WORK
parameter, the work data sets are allocated according to the
values you specified for the WORKUNIT and WORKSPACE
parameters in prefix.MICS.PARMS(JCLDEF).  Use the
data_set_allocation_parameters to override this default,
either to alter the space allocation or to use System Managed
Storage (SMS) parameters to control data set placement and
characteristics.
```

Note: If you allocate insufficient space for the WORK data sets, DAYnnn and/or INCRnnn processing will fail and can only be restarted from the beginning.

Note: If internal step restart is active, you can override the WORK data set allocation parameters at execution-time using the //PARMOVRD facility. For more information about execution-time override of dynamic data set allocation parameters, see the PIOM, section 2.3.6.

Specify data set allocation parameters, separated by blanks, according to SAS LIBNAME statement syntax. If you need multiple lines, repeat the WORK keyword on the continuation line.

WORK accepts the engine/host options documented in the SAS Companion for the z/OS environment, including STORCLAS, UNIT, SPACE, BLKSIZE, DATACLAS, MGMTCLAS, and VOLSER.

Important! Do not specify the DISP parameter.

Example 1:

```
WORK n STORCLAS=MICSTEMP SPACE=(XXX,(pppp,ssss),RLSE)
```

where:

```
  n        - is the number of WORK data sets.
  STORCLAS - specifies a storage class for a new data set.
             The name can have up to 8 characters.
  SPACE    - specifies how much disk space to provide for
             a new data set being allocated.
  XXX      - is TRK or CYL.
  pppp     - is the primary allocation.
  ssss     - is the secondary allocation.
  RLSE     - specifies that free-space should be released
             when the data set is closed.
```

```
Example 2:

WORK n XXX pppp ssss

where:

  n        - is the number of WORK data sets.
  XXX      - is TRK or CYL.
  pppp     - is the primary allocation.
  ssss     - is the secondary allocation.

Example 3 (multiple lines):

WORK n STORCLAS=MICSTEMP UNIT=SYSDA
WORK   SPACE=(xxxx,(pppp,ssss),,,ROUND))

where:

  n        - is the number of WORK data sets.
  STORCLAS - specifies a storage class for a new data set.
             The name can have up to eight characters.
  UNIT     - specifies the generic unit for a new data set.
             The name can have up to eight characters.
  SPACE    - specifies how much disk space to provide for
             a new data set being allocated.
  XXX      - is TRK or CYL.
  pppp     - is the primary allocation.
  ssss     - is the secondary allocation.

Note:  Since there is some performance impact when using
multiple WORK files, you should specify the minimum number of
WORK data sets to meet your work space requirements.  As a
start, try incrementing the number gradually beginning from
the default.
```

```
WORK Considerations
-------------------

How Much Space Should You Allocate?

o  First Time Implementation of Multiple Work Files

   If this is the first time you are implementing multiple
   work files for this product in this unit, review
   prefix.MICS.PARMS(JCLDEF) and find the WORKSPACE
   parameter.  It will resemble this sample statement:

   WORKSPACE      TRK 500 250

   The value shows the current SAS WORK space allocation for
   the unit as a single data set.  It also serves as the
   default value used in the unit's DAYnnn daily update
   (and/or INCRnnn incremental update) step unless you
   provide a WORK parameter.

   To achieve the equivalent work space allocation of
   WORKSPACE TRK 500 250 using multiple WORK data sets that
   will collectively share the work space requirements of
   the daily and/or incremental update step, you could code
   either one of these:

   WORK 2 SPACE=(TRK,(250,125))

   WORK 5 SPACE=(TRK,(100,50))

   To determine the total work space, multiply the number of
   WORK files (n) by the primary (pppp) and secondary (ssss)
   values specified.

   Note:  To simplify the example, only the SPACE parameter
   is shown above.  You can follow either with data set
   allocation parameters like UNIT or STORCLAS as required
   for your site.

o  Adjusting Allocation for Existing Multiple WORK Files

   If you have previously implemented multiple WORK file
   support for this product in this unit, and you want to
   change either the number of WORK files or the space
   allocations, examine prefix.MICS.PARMS(cccOPS) and find
   the existing WORK statement.
```

- If the existing WORK statement only specifies the number of WORK files but does not contain space allocation information as shown below:

  WORK 5

  Then each of the multiple WORK files is allocated using the values from the WORKSPACE parameter of prefix.MICS.PARMS(JCLDEF), as described earlier under First Time Implementation of Multiple Work Files.

  To increase workspace, you can increase the number of WORK files (for example, change WORK 5 to WORK 6,7,8, or 9), increase the space allocation in the WORKSPACE parameter, or do both.

  To decrease workspace, you can decrease the number of WORK files (for example, change WORK 5 to WORK 4,3,2, or 1), decrease the space allocation in the WORKSPACE parameter, or do both.

  You can also elect to explicitly specify the multiple WORK file space allocation by adding the space allocation values directly to the WORK statement. This will remove the link to the prefix.MICS.PARMS(JCLDEF) WORKSPACE parameter for multiple WORK file space allocation. This is recommended as it serves to clearly document, in one place, how multiple WORK files are allocated.

- If the existing WORK statement does include space allocation as shown in the examples below:

  WORK 5 TRK 200 100

  or

  WORK 5 SPACE=(TRK,(200,100)) STORCLAS=MICSTEMP

  Simply change the values to meet your needs.

  If you need more work space, you can increase the number of WORK files (for example, change WORK 5 to WORK 6,7,8, or 9), increase the space allocation (for example, change TRK 200 100 to TRK 250 120), or do both.

  To decrease work space, you can decrease the number of

```
      WORK files (for example, change WORK 5 to WORK 4,3,2,
      or 1), decrease the space allocation (for example,
      change TRK 200 100 to TRK 150 80), or do both.

Note:  If internal step restart is NOT active (RESTART NO)
and you change the WORK parameter, you must:

o  Run cccPGEN
o  Run JCLGENU for DAILY (to regenerate DAILY) and, if
   incremental update is enabled, INCRccc

When internal step restart is active, (RESTART YES), then,
when you change WORK and run cccPGEN, changes take effect
immediately.  There is no need to run JCLGENU.


SASWORK
-------

This statement is optional.

The WORK DD statement in the CA MICS procedures allocates
a temporary data set where SAS keeps its temporary data
files and other items that SAS uses during processing of
the current job.

By default, the space allocated is defined in the member
prefix.MICS.PARMS(JCLDEF) with the WORKSPACE and WORKUNIT
parameters, then generated into all the JCL procedures for
a given unit.

With the SASWORK statement you have the option to override
this unit-wide definition to specify the space allocation
individually for the current step.

The format of the SASWORK statement is:

SASWORK data_set_allocation_parameters

where data_set_allocation_parameters is one or more data set
allocation parameters (for example, STORCLAS or SPACE)
separated by spaces.

You can also specify the SASWORK parameter as the following:

SASWORK XXX pppp ssss

where:
```

```
  XXX  is TRK or CYL
  pppp is the primary allocation
  ssss is the secondary allocation

If you omit the data_set_allocation_parameters or the SASWORK
statement, the WORK data set is allocated according to the
values you specified for the WORKUNIT and WORKSPACE
parameters in prefix.MICS.PARMS(JCLDEF).  Use the
data_set_allocation_parameters to override this default,
either to alter the space allocation or to use System Managed
Storage (SMS) parameters to control data set placement and
characteristics.

Specify data set allocation parameters, separated by blanks,
according to SAS LIBNAME statement syntax.  If you need
multiple lines, repeat the SASWORK keyword on the
continuation line.

Example:

SASWORK STORCLAS=MICSTEMP SPACE=(XXX,(pppp,ssss))

where:

  STORCLAS - specifies a storage class for a new data set.
             The name can have up to 8 characters.
  SPACE    - specifies how much disk space to provide for
             a new data set being allocated.
  XXX      - is TRK or CYL.
  pppp     - is the primary allocation.
  ssss     - is the secondary allocation.


Note:  If you change the SASWORK parameter, you must:

o  Run cccPGEN
o  Run JCLGENU for DAILY (to regenerate DAILY) and, if
   incremental update is enabled, INCRccc
```

```
MULTWORK|NOMULT fff fff ... fff
-------------------------------

Since multiple work files usage impacts performance, this
product provides these optional parameters so you can
restrict multiple work files usage to only those files having
excessive space requirements.

Note:  You can only use one of these optional parameters with
the WORK statement, NOT both.

The MULTWORK parameter restricts the use of multiple WORK
files to ONLY those listed after the MULTWORK keyword.

MULTWORK fff fff ... fff

where fff is the unique three character identifier

If you need multiple lines, repeat the MULTWORK on the
continuation line.

The NOMULT parameter forces the use of multiple WORK files
for all files EXCEPT those specified after the NOMULT
keyword.

NOMULT fff fff ... fff

where fff is the unique three character identifier

If you need multiple lines, repeat the NOMULT on the
continuation line.
```

8.7.2 Input Data set Specification (INPUTccc)

```
The JCL statements defining the input data sets are
specified.  These JCL statements will be included in the
daily processing JCL.  The DD name for the input must be
INPUTccc.
```

8.7.3 Original System Id Assignment (cccSYSID)

The data element ORGSYSID is used by CA MICS to identify the originating system of the data. Additionally, the ORGSYSID is used by CA MICS Checkpoint processing to prevent inclusion of duplicate data into the CA MICS database.

As discussed in section 8.6.1, MACHINE (the long name of the originating system) is converted to the four byte ORGSYSID.

By default, MTI assigns the first four bytes of MACHINE to ORGSYSID. If the default ORGSYSIDs do not provide acceptable uniqueness to distinguish between various machines (i.e. multiple machines start with the same four characters), cccSYSID can be used to manually assign ORGSYSIDs. cccSYSID is invoked after COMMONEXIT statements.

WARNING: Each ORGSYSID must be defined in prefix.MICS.PARMS(SYSID).

The shared MICF Inquiry MTILD3 can be used to produce a list of all encountered machine names (reference section 8.4.3).

Below is a sample cccSYSID exit using the information extracted from the MTILD3 report:

```
    IF MACHINE =: 'NTW' THEN ORGSYSID = 'NTWK';
       ELSE IF MACHINE = : 'NTS' THEN ORGSYSID = 'NTSV' ;
       ELSE IF MACHINE = : 'SOL' THEN ORGSYSID = 'SUN';
```

If you have numerous machines it will be very inefficient to use IF, THEN, ELSE logic. Alternatively, you can use the machines from the MTILD3 report to create a format of machine IDs and their corresponding ORGSYSID assignments. For example:

```
  ORGSYSID = PUT(MACHINE,$MTIORGS.);
  IF ORGSYSID = 'UNKN' THEN DO;

  /* 'UNKN' IS THE OTHER CLAUSE IN THE USER WRITTEN  */
  /* $MTIORGS FORMAT.  SEE SMG 4.5 FOR DETAIL ON     */
  /* CREATING AND ACCESSING USER WRITTEN FORMATS     */

   INLINE  = 'MACHINE '|| MACHINE || ' NOT FOUND IN FORMAT' ;
   WARNX   = 'DEFAULTING TO ORGSYSID UNKN';
   CALL MICSLOG('BAS00195') ;

  END;
```

This additional logic tests for the OTHER value which indicates that no match was found for this machine name within the format and issues a BAS00195W message, containing the machine name, to the MICSLOG. This approach may assist in tracking new machine names. You can also abort the job when an unidentified machine is encountered by inserting an:

```
ABORT ABEND nnn;
```

prior to the END statement above. Substitute any number for nnn other than 998 or 999 which are used by CA MICS and SAS.

Note that during the DAILY update step the machine names are already noted on the MICSLOG. Additionally, if the resulting ORGSYSID is not already defined in SYSID (either the complex or unit level) the job will terminate abnormally if UNDEFINEDORGSYSID ABORT has been specified.

Another option is to use WHEN/SELECT processing such as:

```
SELECT(MACHINE);
  WHEN ('unixabcd')
    ORGSYSID = 'UNIX' ;
  WHEN ('WINNT40DEV')
    ORGSYSID = 'NT' ;
  WHEN .....
  OTHERWISE ORGSYSID = 'UNKN' ;
```

Remember that SELECT/WHEN processing performs the action, in this case the assignment of ORGSYSID, indicated in the "FIRST" true WHEN statement. Therefore the order of your statements may be critical to their success. Additionally, you will need to test on the actual machine name (unlike the IF, THEN, ELSE example above) which tested that the machine name simply 'began' with those characters.

8.8 MTI Operational Jobs

The MTI Operational jobs are used to maintain and update the MTI Dictionary as well as MTI components (see the checklists in Section 8.5). The jobs are generated as part of the MTI installation and the creation of an MTI component.

For more detailed description of utility control statement syntax, refer to Appendix B of this guide.

8.8.1 Initialize the Dictionary (DICINIT)

The DICINIT job is used to initialize the MTI Dictionary and create the necessary files. This job should only be run once.

Section 8.5.1 contains the checklist which uses the DICINIT job to initialize the MTI Dictionary.

8.8.2 Load the MTI Dictionary (DICLOAD/cccLOAD)

In order for MTI components to process NSM Systems Performance data, it is necessary to load the MTI Dictionary with the resources and resource subtypes in the input data. The MTI Dictionary load process is used to scan the input data, and update the MTI Dictionary with this information.

The INPUTCSV DD is used to specify the input data sets for the dictionary load process.

The INPUTMTX DD (optional) is used to specify the input data set to be used to load the shipped Standardized Metrics Update File (SMUF). The data used here is delivered with the CA MICS product and is in CSV/HFS format . The resource information in these files are related to certain types of machines and will aid in more accurate mapping of resource and resource subtypes. If the INPUTMTX DD is not specified, the INPUTCSV DD data set is the only one that will be used to load resource and resource subtypes. To update and "overlay" existing entries in the dictionary with the more accurate metrics information found in the SMUF file, the INPUTMTX DD will need to point to the shipped file and "OVERLAY" will need to be specified on the MTISYSIN DD.

```
ie.   //MTISYSIN DD *
       %DICLOAD(LOAD,OVERLAY=YES)
```

The cccDLOAD is a component specific version of DICLOAD using the INPUTCSV definition from sharedprefix.MICS.PARMS(cccINPUT) created during the MTI component creation job (MTICREAT). If changes are made to cccINPUT, then cccDLOAD and cccDSCAN should be regenerated using JCLGEN0 or JCLGENUC.

MTI also generates the DICSCAN and cccDSCAN jobs. These jobs perform the same functions as DICLOAD/cccDLOAD but do not update the MTI Dictionary.

Section 8.5.3 contains the checklist which uses cccDLOAD (and cccDSCAN) to update existing MTI components with resources and resource subtypes when the input data set has changed. Section 8.5.4 contains the checklist which uses DICLOAD (and DICSCAN) recover the existing MTI Dictionary in the event that it is lost or damaged.

8.8.3 Create an MTI Component (MTICREAT)

MTI is specially designed to allow for the creation of multiple components, rather than just one, each of which is customized to the NSM input data. In this way, the installation can control how much data will be retained in the CA MICS database.

The MTICREAT job requires that you:

o Select a component name. The component name must be three characters long, must start with 'U' or 'Z', and cannot be an already existing component name.

  The component name is specified by the SYSPARM= parameter. For example, the following statement creates an MTI component called 'U10':

```
//MTICREAT EXEC MICSCOM,SYSPARM='U10'
```

  MTICREAT allows the option to specify a component step number explicitly. If not specified, MTICREAT selects the next available step number.

  The step number is specified after the component name in the SYSPARM= parameter. For example, the following is used to create an MTI component called U10 with a step number of 153:

```
//MTICREAT EXEC MICSCOM,SYSPARM='U10 153'
```

o Select and specify the input data set for the new component.

  This input data set is specified by the INPUTCSV DD in the MTICREAT step. Note that MTICREAT does NOT support concatenated datasets. The resource types in this data set will be converted by MTICREAT into CA MICS file definitions.

Once the component name and input data are specified, the MTICREAT job scans the input data, creates sharedprefix.MICS.GENLIB(cccGENIN) (where ccc is the component name specified on the SYSPARM=), generates all code to execute the component, and creates supporting jobs.

Section 8.5.2 contains the checklist which uses the MTICREAT job to create new MTI components.

8.8.4 Create the CA MICS Data Dictionary Entries(DICDSHEL)

The DICDSHEL job creates or recreates CA MICS Data Dictionary entries. DICDSHEL creates data dictionary entries for each defined NSM Systems Performance resource subtype as well as internally generated MTI data elements used for computation.

Note that a DICDSHEL is automatically executed during the DICLOAD process, adding a Data Dictionary entry for each NEW resource subtype encountered.

8.8.5 Repair the MTI Dictionary (DICREPAR)

Occasionally, updates to the MTI Dictionary may fail due to corrupt indices or other problems. In most cases, the MTI dictionary repair job can be run to correct these problems. DICREPAR performs a SAS repair against all MTI Dictionary files, then rebuilds the indices.

8.8.6 Update an MTI Component (cccUPSCN/cccUPDAT)

From time to time, it may be necessary to update an MTI component if the contents of the input data changes.

One the major design features of MTI is that components are customized to the input data. If the data changes, changes may need to be made to an MTI component. The cccUPDAT is used to synchronize the sharedprefix.MICS.GENLIB(cccGENIN) with the input data specified on the INPUTCSV DD. cccUPSCN produces a report of the updates that will take place should you run the cccUPDAT job. Running this job before running cccUPDAT gives you the opportunity to review any changes prior to implementing them. cccUPSCN does not update sharedprefix.MICS.GENLIB(cccGENIN) or the MTI Dictionary. Both jobs are used in checklist 8.5.3 for updating an existing MTI component.

sharedprefix.MICS.PARMS(cccINPUT) is used to specify the input data set for some MTI component jobs. If changes are made to this member, then cccUPDAT and cccUPSCN should be regenerated using JCLGEN0 or JCLGENUC.

Appendix A: MSI FDA Checklist

This checklist can help you design, create, and implement Field Developed Applications that use the CA MICS-SAS Interface.

Note: Read section 4.2 and all of Chapter 7 before creating any FDAs. These sections contain information describing the FDA generation process and the syntax of all CA MICS Component Generator (MCG) statements.

___1. Name and Number the FDA (ccc and nnn)

Select the three-character name and unique identifying number for your MSI FDA.

The FDA name CANNOT be MSI and should not conflict with existing CA MICS or user FDA names. Use U or Z as the first character to avoid conflicts with existing and future CA MICS Data Integration Applications.

The FDA number can be any value from 150 through 190, but must not be the same number used by a previously generated FDA. If there is a concern about eventually using most or all of the numbers in this range, one should define multiple information areas within a user component wherever it is possible.

___2. Identify the SAS Input data sets

Identify the SAS data sets that will feed your MSI FDA. These must be specified on INPUTSAS or INPUTMODEL statements in sharedprefix.MICS.GENLIB(cccGENIN). Run a SAS PROC CONTENTS to get a listing of variables in the SAS data sets. From this listing the names of common data elements, sort sequences, and data element types can be determined.

```
___3. Create Initial sharedprefix.MICS.GENLIB(cccGENIN)

      ___a. Copy sharedprefix.MICS.GENLIB(MSIGENIN) to
            sharedprefix.MICS.GENLIB(cccGENIN).  This member
            will serve as a template to assist in the
            definition of the new MSI FDA.

      ___b. You do not need to change or modify any cccGENIN
            contents at this time.  Just make sure the name
            (cccGENIN) matches the name you selected for your
            MSI FDA.  For example, if you selected ZTQ as
            your MSI FDA name in Step 1, the
            sharedprefix.MICS.GENLIB member must be named
            ZTQGENIN.

___4. Create sharedprefix.MICS.CNTL(cccCGEN)

      ___a. Edit sharedprefix.MICS.PARMS(JCLGENUC) so that it
            contains the following:

            cccCGEN

            where ccc is the MSI FDA name you selected in
            Step 1.

            Submit the job in
            sharedprefix.MICS.CNTL(JCLGENUC).  Ensure that
            there are no error messages in MICSLOG or
            SYSTSPRT, that the MICSLOG contains the normal
            termination message, BAS10999I, and that the job
            completes with a condition code of zero.

      ___b. JCLGENUC creates a new cccCGEN member in
            sharedprefix.MICS.CNTL.  Edit cccCGEN and add a
            JCL statement referencing each SAS library
            containing SAS data sets used as input for this
            MSI FDA.

            Example:  //libname1 DD DSN=sas.source.lib1,
                      //            DISP=SHR
                      //libname2 DD DSN=sas.source.lib2,
                      //            DISP=SHR
```

Note: In Step 5 you will define the files for your MSI FDA. Each file will identify the source SAS data sets used as input with the MSI INPUTSAS or INPUTMODEL statements. You will need to remember and use the same libnames that you use for the JCL statements you add to cccCGEN in this step.

___5. Update sharedprefix.MICS.GENLIB(cccGENIN)

In this step you update the cccGENIN member you created in Step 3 with information that defines your MSI FDA.

___a. Edit sharedprefix.MICS.GENLIB(cccGENIN)

___b. Edit the COMP statement to specify the name (ccc) and number (nnn) of the MSI FDA. You determined the values for ccc and nnn in Step 1 above.

___c. Edit the AREA statement to specify the information area name (iii) for the first set of files. It is recommended that you begin the information area name with U or Z to avoid conflicts with existing CA MICS information area names.

Update the AREA statement label to reflect the contents of this information area.

___d. Edit the FILE statement to specify the file name name (fff) for your first MSI FDA file.

Update the FILE statement label to reflect the contents of this file.

```
___e. EDIT the INPUTSAS statement and update
      libname.member to identify the SAS data set used
      to create this file.  Use the same libname that
      you used when adding the JCL statement for this
      SAS data library to cccCGEN in Step 4 above.

___f. Use MSI statements as described in Chapter 4 of
      this guide to complete the file definition.
      Standard data element definition statements can
      also be used. See Section 4.2.4.3, MSI Definition
      Statements and Section 4.2.4.4, Standard Data
      Element Definition Statements.

___g. Repeat Steps d-f to define other files for this
      information area.  If an additional information
      area is required, add a new AREA statement after
      the last file definition, then code file
      definition statements as described in Steps d-f.

      ************************************************
      *                                              *
      * NOTE: It is not necessary to define all      *
      * information areas and files at this time.    *
      * Additional information areas and files can be *
      * defined during the GENFILE process.          *
      *                                              *
      ************************************************
```

___6. Run MCG SYNTAX Check

The syntax checking mode of the CA MICS Component Generator (MCG) provides a quick method of testing for syntax errors in cccGENIN. The entire syntax of the cccGENIN is tested including sequence definitions and dependencies, but the FDA is not integrated into the CA MICS database nor is any code generated. Running a GEN SYNTAX can help find errors in FDA definition before any code is generated. A GEN SYNTAX can be run any time a change is made to cccGENIN.

___a. Edit sharedprefix.MICS.GENLIB(cccGENIN).

___b. Change the ACCT option on the COMP statements to NOACCT. For GEN SYNTAX processing, NOACCT must be specified. Since the FDA has not been previously defined, specifying ACCT will cause the MSICGEN to fail. This option can be changed to ACCT during the GENSHELL and GENFILES MCG processes.

___c. Change the GEN statement to read GEN SYNTAX.

___d. Run the syntax check by submitting the job in sharedprefix.MICS.CNTL(cccCGEN). Ensure that there are no error messages in ISPLOG and MICSLOG, that MICSLOG contains the normal termination messages, BAS20999I, and that the Job completes with a condition code of zero.

___e. If you encounter syntax checking errors, correct sharedprefix.MICS.GENLIB(cccGENIN) and resubmit the cccCGEN job.

___f. This process can be repeated as many times as necessary.

```
___7. Run the MCG GENSHELL Process

      The GENSHELL process creates all the modules necessary
      to support a MSI FDA and defines the FDA to CA MICS.
      Since the GENSHELL can only be run once, the FDA name
      and number must be finalized at this time and cannot be
      changed.

      ___a. Edit sharedprefix.MICS.GENLIB(cccGENIN) and
            change GEN SYNTAX to GEN GENSHELL.

      ___b. Modify the COMP statement to specify the
            ACCT/NOACCT option.  It is recommended that the
            ACCT option be specified on the COMP statement
            whether or not User Defined Accounts are actually
            used in the MSI FDA.  This will make it easier to
            add User Defined Account Codes if they are
            required at a later date.  The inclusion of
            account codes is controlled by adding a

            NAME @@ACCT  00   0 0 0 0 0

            statement to the file definition and then
            specifying @@ACCT in a SEQUENCE statement.

            ************************************************
            *                                              *
            * NOTE: It is not necessary to define all      *
            * information areas and files at this time.    *
            * Additional information areas and files can be *
            * defined during the GENFILE process. However, *
            * data dictionary shells are only created for  *
            * data elements defined in the GENSHELL        *
            * process.                                     *
            *                                              *
            ************************************************
```

```
      ___c. Do not create sharedprefix.MICS.PARMS(cccACCT) at
            this time.  Default
            sharedprefix.MICS.PARMS(cccACCT) and (cccACRT)
            members will be generated by the GENSHELL
            process.

      ___d. Run the GENSHELL by submitting the job in
            sharedprefix.MICS.CNTL(cccCGEN).  Ensure that
            there are no error messages in ISPLOG and
            MICSLOG, that MICSLOG contains the normal
            termination messages, BAS20999I, and that the job
            completes with a condition code of zero.

___8. Run the GENFILES process.

      At this point, the CA MICS Component Generator (MCG)
      has created all the necessary component definition
      modules for the new FDA.  The GENFILES process is used
      to activate the FDA and generate the modules required
      to execute it.

      ___a. Edit sharedprefix.MICS.GENLIB(cccGENIN)

      ___b. If you do not need to make additional file or
            information area updates at this time, go to
            Step 8.c now.

            To make file or information area updates now,
            edit sharedprefix.MICS.GENLIB(cccGENIN) and code
            the updates.  Change the GEN statement to GEN
            SYNTAX to check your modifications.

            Run the GEN SYNTAX check by submitting the job in
            sharedprefix.MICS.CNTL(cccCGEN).  Ensure that
            there are no error messages in ISPLOG and
            MICSLOG, that MICSLOG contains the normal
            termination messages, BAS20999I, and that the job
            completes with a condition code of zero.
```

```
___c. Update sharedprefix.MICS.PARMS(cccACCT).  You
      must define at least one account field if you
      specified ACCT on the COMP statement in step
      7.b above.

___d. Edit sharedprefix.MICS.GENLIB(cccGENIN) and
      change the GEN statement to GEN GENFILES.

      Run the GEN GENFILES process by submitting the
      job in sharedprefix.MICS.CNTL(cccCGEN).  Ensure
      that there are no error messages in ISPLOG and
      MICSLOG, that MICSLOG contains the normal
      termination messages, BAS20999I, and that the job
      completes with a condition code of zero.

___e. To complete the installation of MSI FDA, refer to
      section 3.8.2 (Adding a Product to an Existing
      Database), or to section 3.8.3 (Adding a New
      Product to a New Database Unit) in the PIOM.

      For instructions concerning the coding of unit-
      level parameters for the FDA, refer to Section
      7.6 of this guide.
```

Appendix B: MTI Control Statement Reference

This Appendix documents in greater detail control statements that appear in the MTI jobs (Section 8.8). Most of these statements are related to the MTI Dictionary Utility. MTIDUTIL is the utility provided for performing maintenance to the MTI Dictionary. Maintenance is performed initially, when creating the dictionary, as well as any time you alter the NSM configuration and thus the input data for your MTI components.

During installation of MTI, the MTIDUTIL job is created by manually executing the MTI Dictionary Utility.

The following is an example of the MTIDUTIL JCL:

```
//   . JOB .
//* START OF USER DEFINED JCL STATEMENTS
//* END OF USER DEFINED JCL STATEMENTS
//* MTIDUTIL CREATED 18FEB99 17:04 BY JCLGENU
//MTIDUTIL EXEC  MICSCOM
//DICTNARY DD DISP=OLD, DSN=sharedprefix.MICS.TABLES
//SYSIN    DD DISP=SHR,
              DSN=sharedprefix.MICS.INCLLIB(MTIDUTIL)
//INPUTCSV DD DISP=SHR,
              DSN=NSM.performance.mgmt.data
//MTISYSIN DD *
```

All MTI Dictionary Utility statements are entered in the MTISYSIN DDname.

This section contains the following topics:

B.1 Initialize the Dictionary (DICINIT) (see page 500)
B.2 Load the MTI Dictionary (DICLOAD) (see page 500)
B.3 Create the CA MICS Data Dictionary Entries(DICDSHEL) (see page 501)
B.4 Repair the MTI Dictionary (DICREPAR) (see page 502)
B.5 Create or update an MTI Component (MTIGENER) (see page 502)

B.1 Initialize the Dictionary (DICINIT)

```
STATEMENT SYNTAX

   %DICINIT({,FORCE=NO | YES})

   where:  Items in brackets are optional
           Items in uppercase are entered exactly as shown
           Items in lowercase (e.g. ddname) are user supplied
STATEMENT OPTION DEFINITIONS

FORCE=NO | YES

    determines whether or not existing MTI Dictionary files
    are to be reinitialized.  Default is NO.

    **********************  WARNING  ***********************
    *                                                      *
    *  Since FORCE=YES may adversely affect all MTI        *
    *  components, it should only be executed under        *
    *  direction of CA MICS Product Support.               *
    *                                                      *
    **********************  WARNING  ***********************
```

B.2 Load the MTI Dictionary (DICLOAD)

```
In order for MTI components to process NSM Systems
Performance data, it is necessary to load the MTI Dictionary
with the resources and resource subtypes in the input data.

The MTI Dictionary load process (DICLOAD) is used to scan
the input data, and update the MTI Dictionary with
this information.

STATEMENT SYNTAX

   %DICLOAD(verb,
            {,INPUTCSV=ddname})

   where:  Items in brackets are optional
           Items in uppercase are entered exactly as shown
           Items in lowercase (e.g. ddname) are user supplied
```

```
STATEMENT OPTION DEFINITIONS

verb

    sets the mode of the Dictionary load process function and
    can be either LOAD or SCAN.

    SCAN   scans the input data (as defined by the INPUTCSV=
           keyword) and report on what will be added to the
           Dictionary.

    LOAD   also scans the input data, but will apply updates
           to the DICTIONARY.

INPUTCSV=ddname

    Defines the DDname that contains the input data.  By
    default, INPUTCSV is defined as INPUTCSV.

REPORT=YES | NO

    determines whether reports showing items to be added
    to the MTI Dictionary will be produced.  Default is YES.

OVERLAY=YES | NO

    determines whether the Standardized Metrics Update File
    entries should be used to overlay the existing duplicate
    data found in the current data dictionary. Default is NO.
```

B.3 Create the CA MICS Data Dictionary Entries(DICDSHEL)

The DICDSHEL statement creates or recreates CA MICS Data Dictionary entries. DICDSHEL creates data dictionary entries for each defined NSM Systems Performance resource subtype as well as internally generated MTI data elements used for computation. A DICDSHEL is automatically executed during the DICLOAD process adding a Data Dictionary entry for each NEW resource subtype encountered.

```
STATEMENT SYNTAX

%DICSHELL({PRINT=NO | YES}
          {,COMMIT=NO | YES})

   where:  Items in brackets are optional
           Items in uppercase are entered exactly as shown
           Items in lowercase (e.g. ddname) are user supplied
STATEMENT OPTION DEFINITIONS

PRINT=NO | YES

    optionally prints the contents of the
    Data Dictionary entries that will
    be added to sharedprefix.MICS.DIC.TEXT.  Default is NO.

COMMIT=NO | YES

    controls whether or not the generated data dictionary
    elements will be written to sharedprefix.MICS.DIC.TEXT.
    Default is NO.
```

B.4 Repair the MTI Dictionary (DICREPAR)

```
If the MTI Dictionary is damaged in some way, DICREPAR
attempts to correct the problem by running a SAS Repair
against Dictionary data sets and rebuilding indexes.
STATEMENT SYNTAX

   %DICREPAR
STATEMENT OPTION DEFINITIONS

(none)
```

B.5 Create or update an MTI Component (MTIGENER)

```
There are two ways to create and update an MTI component.
The first is to manually create the component definitions
in sharedprefix.MICS.GENLIB(cccGENIN) from scratch.
```

```
The second way is to let MTI create and maintain the MTI
component.  MTIGENER is used by the MTICREAT and cccUPDAT
jobs to create and update MTI component definitions based
upon the input data.  In this way, the MTI component is
customized to only those resources and resource subtypes
actually in the data.

Note that the purpose of MTIGENER is to create and update
MTI component definitions.  It does not execute the CA MICS
Component Generator.

STATEMENT SYNTAX

%MTIGENER(ccc
          {,INPUTCSV=ddname}
          {,LABEL=label_name}
          {,STEPNUM=step_number}
          {,FILEPREF=file_prefix}
          {,COMMIT=NO | YES})

   where:  Items in brackets are optional
           Items in uppercase are entered exactly as shown
           Items in lowercase (e.g. ddname) are user supplied
STATEMENT OPTION DEFINITIONS

ccc (required)

    is the three character name of the component to be
    generated and must start with U or Z.  If the component
    has been previously defined, MTIGENER will add any new
    resource found on the input data as new files.

The following are optional:

INPUTCSV=ddname

    specifies the DDNAME of the input data.  INPUTCSV is the
    default DDNAME.

LABEL=label_name

    sets the name of the component.  Label_name can be up to
    forty characters.  If LABEL= is not specified, MTIGENER
    creates a name for you.
```

```
STEPNUM=step_number

    specifies the step number for the DAILY, WEEKLY, MONTHLY,
    and YEARLY jobs.  It should have a unique value from 150
    through 190. If STEPNUM is not specified, MTIGENER will
    assign the first available step number.  If there is a
    concern that eventually all of the step numbers will be
    used, one should define a new information area within an
    existing user component.

FILEPREF=file_prefix

    sets the first characters of generated file names.  If
    not specified, MTIGENER will use the first two characters
    of the component name.

COMMIT=NO|YES

    controls whether not the definitions will be stored in
    sharedprefix.MICS.GENLIB(cccGENIN).  By default,
    MTIGENER will not update component definitions
    (COMMIT=NO).
```