MICRO

A Relational Database Management System

Institute for Labor and Industrial Relations

The University of Michigan

Contents

Introduction . . . 1
How to Use This Manual . . . 1
Other Information on Micro . . . 2

I Terms, Concepts, and Procedures 3

Micro Overview 5

Introduction . . . 5
Result Set Concept . . . 6
Basic Operations . . . 8
Sorting and Weeding . . . 9
Crosstabulated Datasets . . . 9
Data Relations . . . 9
Dataset Management . . . 11
Dataset Documentation . . . 14
Data Entry . . . 15
Micro Programming Facility . . . 15
Miscellaneous Commands . . . 16

Dataset Structure 19

Introduction . . . 19
Directory File . . . 21
Dictionary File . . . 22
Security . . . 25
Documentation for Datasets . . . 25

Dataset Definition 27

Menu Mode . . . 27

Command Mode 30
MTS Macro Mode 32

Data Entry and Updating **35**
Interactive and Free Format 35
Fixed Format 39
Custom Programming Method 48

Visual Data Entry and Editing **49**
Visual Program Function Key Definition 49
Screen Definition 53
Other Parameters 55

MTS **57**

II Micro Tutorial **59**

Introduction **61**

Command Language I **63**
Running Micro 63
Help Command 63
Making Datasets Available 64
List Command 64
Get Command 64
Print Command: Printing the Contents of a Dataset 65
Print Command: Printing Only the Number of Records 66
Unscrambling (Decrypting) Data 66
Correcting Mistyped Commands 66
Context-sensitive Help 66
Listing Possible Replacements 66
Document Command: Documenting Dataset Structure 67
Find Command: Data Retrieval and the Result Dataset 68
Permanent and Temporary Datasets 69
Name Command: Retaining the Result Dataset 70
Abbreviations and Synonyms 70

Print Command: Horizontal and Vertical Layouts 70
Find Command: The **AND** Connective 71
Print Command: Specifying Fields 72
Find Command: The **OR** Connective 73
Change Command: Changing Data 73
Release Command: Releasing Accumulated Datasets 74
Sort Command: Sorting a Dataset 74
Select Command: Selecting Fields from a Dataset 75
Weeding Duplicate Records 75
Key Command: Generating a Unique Field 76
Crosstabulate Command: Aggregating Data 77
Print Command: Stratified Layout 77
Crosstabulate Command: Descriptive Statistics 78
Write Command: Exporting Data 78
Command Sequence to Calculate Ages 79
    Calculate Command: Creating a New Field 79
    Calculate Command: Calculating Ages from Birthdates 79
Settings Command: Micro Settings 80
Save Command: Saving a Temporary Dataset 81
Stop Command: Ending the Session 81

Command Language II 83
Find Command: More Search Methods 83
External Data 85
Key Fields 85
Restrict Command: Retrieval Using Links to Another Dataset 86
Splitting and Combining Datasets 87
Release Command: Specifying which Datasets to Keep 87
Restrict Command: The Not-Restriction 87
RAJ: Joining Datasets on Matching Links 88
Validating Data 89
Replace Command: Making Permanent Changes to a Dataset 90
Join Command: Including Records with Non-Matching Links 90
Destroy Command: Destroying a Dataset 92
Keys and Links: Relationships between Datasets 93

Database Definition 97

Running MicDef 97

Getting Help 98

Creating a Dataset in Menu Mode 99

Creating a Field 99

Creating a Category 101

Adding a Category to an Existing Field 102

Getting Help From a Prompt 103

Adding Categories from a File 103

Showing the Field Structure 104

Inserting a New Field 104

Showing Dataset Structure 106

Editing a Field 107

Adding a Default Category 107

Moving Between Menus 108

Saving a Dataset Definition 109

Using Command Mode 110

Overriding Default Attributes 111

Edit Command 111

Creating a Link Field 112

Append Command: Appending Categories 112

Insert Command: Inserting Categories 113

Insert Command: Inserting a Field 114

Show Command: Showing the Field Structure 115

Redo Command: Redoing a Field 115

Field Scaling 116

Patterns for Names 117

Change Command: Changing Field Items 118

Menu Mode 118

Transferring a Dataset From One *userid* to Another 119

Data Entry 123

Interactive 123

Free Format 126

Enterdata Command 128

Fixed Format . . . 130
Custom Programming . . . 135
Visual Data Entry and Editing . . . 136

Miscellaneous Topics **137**
Updating . . . 137
Interactive . . . 137
Free format . . . 138
Update Command . . . 138
Runfiles . . . 139
Command Files . . . 140
Micro Programming Facility . . . 141
Micro Interface Functions . . . 145
Parameter Passing . . . 150
Generating and Executing Micro and MPF Micro commands . . . 151
A Simple MPF Handler . . . 154
Using the Demo MPF Micro command . . . 155

III Command Reference **157**

ILIR:Micro **159**
ILIR:Micro Parameters . . . 161
Example of Command Reference . . . 163
CALCULATE Micro Command . . . 167
CHANGE Micro Command . . . 177
COMBINE Micro Command . . . 181
COMMENT Micro Command . . . 183
CROSSTABULATE Micro Command . . . 185
DATESTAMP Micro Command . . . 191
DESCRIBE Micro Command . . . 193
DESTROY Micro Command . . . 197
DISPLAY Micro Command . . . 199
DOCUMENT Micro Command . . . 201
ENTERDATA Micro Command . . . 205
EXPORT Micro Command . . . 207

FIND Micro Command . . . 219
FULLDOC Micro Command . . . 225
GET Micro Command . . . 229
HELP Micro Command . . . 231
JOIN Micro Command . . . 235
KEY Micro Command . . . 241
LIST Micro Command . . . 243
LOCK Micro Command . . . 245
MESSAGE Micro Command . . . 249
MTS Micro Command . . . 251
NAME Micro Command . . . 253
PRINT Micro Command . . . 257
PURGE Micro Command . . . 277
READ Micro Command . . . 279
RELEASE Micro Command . . . 281
REMOVE Micro Command . . . 283
REPLACE Micro Command . . . 287
RESETTINGS Micro Command . . . 289
RESTRICT Micro Command . . . 293
RESTRICTANDJOIN Micro Command . . . 297
SAVE Micro Command . . . 303
SELECT Micro Command . . . 305
SETTINGS Micro Command . . . 307
SORT Micro Command . . . 313
STOP Micro Command . . . 315
TECHDOC Micro Command . . . 317
UPDATE Micro Command . . . 321
WRITE Micro Command . . . 325

ILIR:MicDef **329**

ALTER MicDef Command . . . 331
APPEND MicDef Command . . . 333
COPY MicDef Command . . . 335
DELETE MicDef Command . . . 337
DESCRIBE MicDef Command . . . 339

DESTROY MicDef Command 341
DOC MicDef Command 343
EDIT MicDef Command 345
FDOC MicDef Command 347
GET MicDef Command 349
HELP MicDef Command 351
INSERT MicDef Command 353
MENU MicDef Command 355
MOVE MicDef Command 357
PERMIT MicDef Command 359
REDO MicDef Command 361
STOP MicDef Command 363
TDOC MicDef Command 365
WRITE MicDef Command 367

ILIR:Micro.Form **369**
CLOSE Micro.Form Command 371
DELETE Micro.Form Command 373
DOCUMENT Micro.Form Command 375
RECODE Micro.Form Command 377
REDO Micro.Form Command 379
STOP Micro.Form Command 381

IV Micro Programming Facility **383**
Introduction 385
MPF Micro Commands 385
Reserved Words 387
Generating Micro Commands 388
Micro Interface Functions 390
Front End Processor 395
Exceptional Conditions 396
Running the Preprocessor 404
Target Code 405

Appendix: Deprecated Features 411

ILIR:Micro Parameters 411
CALL Micro Command 413
EXTCALL Micro Command 417
MICALL Micro Command 421
USE Micro Command 423
Macro Facility 425
Deprecated Settings and Modifiers 428

Index 431

List of Tables

Example of Micro Dataset 5

Micro Commands That Produce a Result Set 7

Micro Dataset Management Commands 12

Micro Dataset Documentation Commands 15

Micro Data Entry Commands 15

Micro Miscellaneous Commands 16

Dataset Structure 20

Field Types in Micro 23

Minimum and Maximum Values by Field Size and Type 23

Micro and MicDef Documentation Commands 26

MicDef Dataset Editing Commands 31

MicDef Dataset Management Commands 31

MicDef Dataset Documentation Commands 32

MicDef Miscellaneous Commands 32

Dataset Definition Notation 33

Methods of Data Entry and Updating 35

Record Boundaries for Fixed-Format Data Entry 42

Input Data Types 44

Recode Types 45

Micro.Form Commands 46

Visual Program Function Key Definitions 49

MicDef Menus 97

Syntax Generic Terms 163

Calculate Setting 169

Calculate Command Logarithmic Functions 172

Calculate Command Trigonometric Functions 172

Calculate Command Arithmetic Functions 172

Calculate Command Date Functions 173
Calculate Command Time Functions 175
Calculate Command Miscellaneous Functions 176
Crosstabulate Settings 186
Descriptive Statistics From Crosstabulate Command 190
Join Setting 236
Dataset Locking by Micro Commands 247
Print Command Modifiers 263
RestrictAndJoin Setting 298
MPF Dataset Attribute Functions 391
MPF Field Attribute Functions 391
Enumerations for type FIELD_TYPE 393
Enumerations for type FIELD_SCFUNC 393
MPF Record Access Functions 393
MPF Category Attribute Functions 394
Return Codes in MPF Execution 397
C Signals Received by an MPF Handler 398

Introduction

The Micro Relational Database Management System is a family of computer programs that runs on MTS, the academic mainframe operating system at the University of Michigan. The principal program is **ILIR:Micro**, an interactive command language interpreter that performs data retrieval, subsetting, combination, aggregation, data entry and editing, and other functions. Some of these functions are available as stand-alone programs.

ILIR:MicDef defines datasets, the number and types of fields and categories. **ILIR:MicEdit** provides full-screen visual data editing and entry. **ILIR:Micro.Add** and **ILIR:Micro.Up** add new records to datasets, and update existing records, respectively. **ILIR:Micro.Cnvrt** creates a Micro dataset from a fixed format source. The format is specified with **ILIR:Micro.Form**.

ILIR is the MTS userid of the Institute for Labor and Industrial Relations, on whose MTS account the programs reside. The Micro Project at the Institute maintains the system. This manual documents the Micro System.

How to Use This Manual

There are several pathways into this manual. As the **Table of Contents** shows, it consists of four Parts. The first, **Terms, Concepts and Procedures**, discusses the Micro programs. The first section of this part, the **Micro Overview**, surveys ILIR:Micro, and is a good introduction for a new user. The next three sections of Part I cover **Dataset Structure, Dataset Definition, Data Entry and Updating**, and **Visual Data Entry and Editing**. This discussion is fairly detailed, and a new user need not read it all. Much of it is summarized in tables, as noted in the **List of Tables** in the front matter. A brief fifth section describes how to obtain MTS documentation and an MTS account.

The second Part is the **Micro Tutorial**, containing five sessions with the Micro programs, actual computer input and output interspersed with commentary. The first two are **Command Language I and II**, the third is **Dataset Definition**, the fourth **Data Entry**, and the fifth **Miscellaneous Topics**. The Micro programs and most of the necessary concepts can be learned from these tutorials, without reference to the rest of the manual. The Table of Contents entries for the tutorials indicate what each example covers, and can be used to find an example which accomplishes a particular task, or is close enough to be adapted. The five-session Micro course given at the University of Michigan each fall and winter term is based on these tutorials.

The third Part is the **Command Reference**. The commands are described in a common format, including a complete explanation of command syntax and a precise description of behavior. This reference may be consulted when some point about a command is in question. Commands for ILIR:Micro, ILIR:MicDef and ILIR:Micro.Form are in the Command Reference. The data entry and updating programs (Micro.Add and Micro.Up, and the common behavior of their corresponding Micro commands, Enterdata and Update) are discussed in the Data Entry and Updating section of Part I. The Visual Data Entry and Editing section in Part I discusses ILIR:MicEdit. The ILIR:Micro reference material is available on-line, during Micro sessions, through the Micro Help System.

The fourth Part describes the Micro Programming Facility, which is of interest to programmers. It provides an interface to the C language from the Micro command prompt, a C interface to Micro's run-time internal data structures, and to the Micro command interpreter itself.

An Appendix describes "deprecated" features, which have been superseded by better ways of doing things, but are documented for the benefit of those still using them. They may be removed from the programs at some point, though not without warning.

Other Information on Micro

As noted above, the tutorials are used in the Micro course given at the University of Michigan each fall and winter term. There are five two-hour sessions, given on the central campus in Ann Arbor. No fee is charged. The schedule is in the file ILIR:MICRO/COURSE. For further information, contact the Micro Project at the address below.

There is a computer conference devoted to Micro on MTS, utilizing the Confer II™ software. Issue the MTS command '`$Source ILIR:Forum`' and follow the prompts to register. On-line help in using the program is available. The conference discusses changes, enhancements, bugs and other matters related to the Micro programs. Participation is encouraged.

The Micro Project provides consulting in the use of the Micro System. For such services, or any questions on Micro, send e-mail to the Internet address '`micro.dbms@um.cc.umich.edu`' ('`$mess send to micro.dbms`' on MTS), the Bitnet address '`userILIR@umichum`', or write or call:

Micro Project
Institute for Labor and Industrial Relations
1111 East Catherine Street
University of Michigan
Ann Arbor, MI 48109–2054

313-763-3118
313-763-0913 (fax)

Part I

Terms, Concepts, and Procedures

Micro Overview

Introduction

Micro is a relational database system that operates on collections of information called *datasets.* A dataset is like a table, with rows and columns. A row is called a *record*, and a column is called a *field.* Every record has an entry for each field; each field is represented in every record. Different fields may hold different types of data, and a record may need several fields to describe adequately the data object it represents. Some fields may be character strings; others may be numeric; others may be *categorical*, with numeric or character values corresponding to categories represented by the field.

A university might maintain a dataset of information on students. One field might be Student-ID-num for identification, another might be the Name of the student, another Sex, and so on. The identification number is a numeric field, the name a character field. The Sex field is categorical, with two possible values, Female and Male (and perhaps Unknown, for missing data). The Sex field could be numeric or character. Conceptually, such a dataset resembles the following table.

Example of Micro Dataset

Field Name	Student-ID-num	Name	Sex	...
Record	133680736	Lee Sue Jean	female	...
Record	122045525	Grace Kenneth R	male	...
Record	139904799	Bailey Edith	female	...
⋮	⋮	⋮	⋮	⋮

The normal sequence when starting to use Micro is to define the structure of each dataset, enter the data into those sets, and perform operations on them using ILIR:Micro. The first task is described in the sections on Dataset Structure and Dataset Definition, the second in Data Entry and Editing. Whether that sequence of tasks should be undertaken may depend on the data manipulation capabilities of Micro, so they are discussed first.

Result Set Concept

All data operations in Micro use datasets as operands, and create a *result dataset*, named Result. For example, the Find command creates a dataset of all the records from the operand dataset containing the field values given on the command. The Find command might be used to retrieve all the male students (records where '`sex=male`') in a Students dataset like that described above; the Result set would contain the records of all the male students. However, the Students dataset itself would not be changed. Commands that create a Result leave the operand sets unchanged.

The Result set is a full-fledged dataset, and may be used immediately, by its names of Result, or It, as the operand for a subsequent command. If that command also creates a Result set, the new Result replaces the existing one; there is only one Result set at a time. The following table lists the Micro commands that create a Result set; following the table, each command is discussed.

Micro Commands That Create a Result Set

Command[a]	Example[b]	Result Set
Find	`find in students where sex = male`	records in dataset students where field Sex is male

REMove	`rem from students where -`
	`sex is unknown`
		records in dataset Students where Sex is known (male or female)

Change	`ch in students where -`
	`SIN = 360984224 so that -`
	`class-year = 79`
		dataset Students with record where field SIN=360984424 has field Class-year changed to 79

CALCulate	`calc in students age = int((date() -`
	`- from_yymmdd(Birthdate)) -`
	`/ 365.25)`
		dataset Students with new field Age set to value of expression

SOrt	`sort in students by Birthdate`	dataset Students sorted by field Birthdate in ascending order; records appear in order of decreasing age of student
REMove	`rem withaid from students2`	dataset Students2 without records in dataset Withaid[c]
COmbine	`co withaid and withoutaid`	records in datasets Withaid and Withoutaid[c]
DAtestamp	`da it`	old Result set, with new field Datestamp containing integer date in YYMMDD form
Key	`key it`	old Result set, with new field Key containing unique integer (1...#records in set);
Select	`sel in it key, sex, race`	Key, Sex and Race fields from old Result set
Xtab (CRoss-tabulate)	`xtab in students sex, race`	fields Sex and Race, along with fields Count and Percent showing how many, and what percentage of, records in dataset Students had those values for Sex and Race; all records in Students with a distinct combination of values for Sex and Race (i.e., black and female) are represented by one record in the Result set

REStrict	`res in seta where -`
	`linka is linkb in setb`
		records from dataset Seta whose value for field Linka is also a value for field Linkb on a record in dataset Setb[d,e]

Join	`join seta by linka with setb by linkb`	Linka is the link field in Seta, Linkb the link field in Setb; the records of the Result set have Linka, followed by non-link fields from Seta, followed by the non-link fields from Setb; these new records include all records from Seta, and all records from Setb; any values of Linka in Seta but not in Setb have filler values inserted for Setb's fields on the new record; any values of Linkb in Setb but not in Seta have filler values inserted for Seta's fields on the new record[d,e]
RAJ (RESTRICT -AndJoin)	`raj seta by linka with setb by linkb`	Linka is the link field in Seta, Linkb the link field in Setb; the records of the Result set have Linka, followed by non-link fields from Seta, followed by any non-link fields from Setb; only records from Seta and Setb that have common values for the link fields are included; there are no fields with filler data[d,e]

[a]The minimum acceptable abbreviation is capitalized.
[b]The dash (-) tells Micro that the command is continued to another line.
[c]The two datasets must have identical structure, i.e., the same number and types of fields.
[d]The two datasets need not, and typically don't, have identical structure.
[e]The link fields must have identical size and type.

Basic Operations

The Find command is the data retrieval operation in Micro. It produces a Result set containing all records from the operand dataset that satisfy the search criteria given on the command.

The Remove command is the inverse of the Find command; it produces a Result set containing all records that don't satisfy those criteria. The same criteria on the Find and Remove commands partition the dataset into mutually exclusive Results:

```
find in students where sex = female
remove from students where sex = female
```

The first command produces a Result set containing all records where sex is female, the second a Result where sex is not female (male, or missing data).

The Change command produces a Result set that is the dataset named on the command, with the selected record(s) changed as specified. In the above table, the record '`where SIN = ...`' has the field Class-year set to '`79`'.

The Calculate command in the above table produces a Result set with a new field, Age, set to the value:

```
int((date() - from_yymmdd(Birthdate)) / 365.25)
```

The `date` function provides the current date in an internal form. The `from_yymmdd` function converts an integer date in the form YYMMDD to an internal date. The difference of these two items, divided by the number of days in a year (365.25), the whole quantity converted to integer by the `int` function, is the Age.

The Sort command re-orders the records in the dataset, according to the fields named on the command. These become the first fields in the record. Micro datasets are kept sorted at all times, as noted below.

The second form of the Remove command given above works on datasets, not on fields. It removes an entire dataset from another. The datasets must have identical structure, the same number and types of fields.

The Combine command combines datasets of identical structure. Sorting and weeding of duplicate records are performed, as noted below.

The Datestamp command creates a Result set with a new field, named Datestamp by default. The field holds the current date as an integer in the form YYMMDD.

Sorting and Weeding

The records of a dataset are kept *sorted* (ordered) at all times, in ascending order. The key used in the sort is the entire record; if a record begins with numeric fields, the dataset is in ascending numeric order, if with character fields, in alphabetical order. Therefore, fields by which a dataset is sorted are first in the record in the Result set, like the Birthdate field in the table example. Identical records would normally be adjacent, but in Micro datasets, all identical records are reduced to just one occurence, a process called *weeding.*

If duplicate records are to be retained, each record must be made unique. A unique key may be appended to each record to ensure that no two records are identical, with the Key command. In the examples in the table, the Key command was used to ensure uniqueness among the fields used in the subsequent Select command, which created a Result set containing the fields Key, Sex and Race. In the absence of the Key field, the Select command would have included only one record for each distinct combination of Sex and Race.

Crosstabulated Datasets

A crosstabulated dataset, created with the Xtab command, summarizes or aggregates data. In addition to the count and percent information illustrated in the above table, the Xtab command can produce descriptive statistics—total, mean, median, standard deviation, minimum and maximum—of any numeric field. The Result set from the command

```
xtab in it race,sex by mean age
```

has a field Ave.age, containing the average age of students for each combination of Race and Sex in the old Result set.

Data Relations

The Restrict, Join, and RestrictAndJoin commands produce Result sets that express relations between data elements from different datasets. These relations are accomplished with *link fields,* having the same size and type, in the different sets. For example, a Students dataset might have a field identifying students by Student-ID-Number (SIN), as in some of the above examples. A Fin-Aid dataset of financial aid awards might identify each award by the same field. The Restrict command could create a Result consisting of all records from Students whose SIN value matches an SIN value in Fin-Aid:

```
restr in students where sin is in fin-aid
```

SIN is used as a link field. The Result set contains records of all students who had received financial aid awards.

Each student is represented only once in Students; a given value for SIN is present only once. SIN is a *key field* in Students, identifying a record uniquely. However, a student may have more than one financial aid award. SIN is not a key field in Fin-Aid, and the relationship between Students and Fin-Aid is *one-to-many*.

SIN would be a key field in a Students dataset from a previous term. The relationship between the current Students and a previous one, using SIN as a link field, would be *one-to-one*. For each SIN in Students there would be at most one record with that SIN value in the older datset (though perhaps none). The Result from the command

```
restr in students where sin is in students.last
```

would contain one record for each student in both datasets, those enrolled in both terms.

Datasets can be joined on link fields, as well as restricted, with the Join command. The records of the Result set from the command

```
join students by sin with fin-aid
```

contains the link field from Students, followed by the non-link fields from Students, followed by the non-link fields from Fin-Aid. Each record in Students is represented, even though not all Students have financial aid. Result records for students without aid have have filler values (blanks for character fields, zeroes for numeric, by default) for Fin-Aid fields. Each record in Fin-Aid is represented, even though the SIN occurs only once in Students. The first part of the Result record (from Students) is repeated as necessary to make new records with the fields from Fin-Aid.

The RestrictAndJoin command combines the effects of the previous commands. The Result from the command

```
raj in students where sin is in fin-aid
```

has the same structure as the Result from the Join command in the previous example, but only the Students records that have matching SINs in Fin-Aid are present. No records are created with filler data values. Each Fin-Aid record is represented.

If the link fields are keys in neither dataset (as with SIN in current and old versions of Fin-Aid) the relationship is *many-to-many*. Each record in one set could match each record in the other. The Result could contain each record from the first, combined with each record from the second, though this relation is usually of less interest.

Whether data have a one-to-one, one-to-many or many-to-many relationship is a fundamental question in the design of a database. It also has practical consequences in Micro; fields used as links in set-creation operations must have the same size and type; and the size of the sets resulting from those operations must be kept in mind when the operands are large datasets.

The "link field" specified for each dataset on Restrict, Join and RAJ can be an expression containing more than one field. The link expression for each dataset must have the same number of fields.

```
res in seta where link1a and link2a ... are link1b and link2b ... in setb
join seta by link1a by link2a ... with setb by link1b by link2b ...
raj seta by link1a by link2a ... with setb by link1b by link2b ...
```

A match exists when the ith link fields have the same value in both datasets. If the fields have the same name in both datasets, the link expression for the second dataset can be omitted. Links and keys are explained further in the second Micro tutorial, in Part 2. Examples of datasets with the different relations—one-to-one, one-to-many, and many-to-many—are used to show the Result sets produced by the Join command.

The above is the default behavior for Join and RestrictandJoin; Join and RAJ can produce the same Result sets, regardless of data relationships, when certain features of the syntax are used, as explained in the command reference section.

Dataset Management

The table below lists Micro commands which allow datasets to be locked to avoid file deadlock with other users, activated for analysis, saved permanently or temporarily, printed, destroyed, replaced, exported, etc.

Micro Dataset Management Commands

Command	Example	Effect
LOCK	`lock students`	locks Students data file to modify, to prevent deadlock, before issuing commands that will require modifications
Name	`name it students2`	renames the Result set as temporary dataset students2
SAve	`save students2`	saves Students2 as a permanent dataset
Get	`get ilir`	reads the file ILIR:Directory and makes the datasets described in it available for use
PUrge	`purge`	releases all datasets and reinitalizes available sets to contents of user's directory file
RELease	`rel students`	releases students; `'release all'` differs from purge in that directory information is still available; storage associated with the sets is released
REPlace	`replace students with it`	replaces dataset Students with the current Result set
DESTroy	`destroy students2`	destroys the dataset Students2
Print	`print students`	prints Students dataset in tabular format, with field names as headings, records (field values) as rows
Print	`print for rtf students on -s`	prints Students dataset in RTF format, for impo into Microsoft Word[1] as a table
EXPOrt	`export for spssx in students f1,f2,f3 on -stu`	writes first three fields of Students dataset on -STU.D and commands to import it into SPSS-X on -STU.C

Dataset Locking

The Lock command locks a dataset's dictionary or data files to prevent file deadlock with other users. The commands that operate on datasets lock their files in various ways. All the commands that create a Result set lock data and dictionary files for reading. The Enterdata and Update commands lock the dictionary file for reading and the data file for modification. If another user has the data file locked for reading, modification cannot take place. The Lock command allows a file to be locked explicitly, so it will be available for later modification.

The command sequence

```
Change in Students ...
Change in It ...
```

[1]Microsoft Word and Microsoft Excel are registered trademarks of Microsoft Corporation. SPSS-X and SPSS/PC+ are registered trademarks of SPSS, Inc. SAS is a registered trademark of the SAS Institute, Inc. Lotus 1-2-3 is a registered trademark of Lotus Development Corporation.

```
Change in It ...
...
Replace Students with It
```

first locks the data and dictionary files to read, during which time other users may also issue commands that lock to read. However, the Replace command needs to lock the data file to modify, not possible if others have the data file locked to read. Using '`Lock Students`' before the first Change solves the problem. There is no Unlock command; commands that lock to modify unlock the dataset automatically. Datasets that have been locked, implicitly or explicitly, may be unlocked with the Release command (which also releases storage). A table in the Lock command section of the Command Reference summarizes the locking effects of all Micro commands.

Permanent and Temporary Datasets

Operations on datasets typically require several intermediate Result sets before the final Result is obtained. Since the Result set is replaced by each command that produces a Result set, it must be possible to save intermediate Results until they can be used as operands. The Result set can be saved as a *temporary* dataset, via the Name command, as shown in the table.

Temporary datasets do not exist from one Micro session to the next; permanent disk storage is not associated with temporary datasets. A temporary data set can be saved as a permanent dataset, as in the example of the Save command in the table.

Permanent datasets exist from one Micro session until the next in permanent MTS disk files or on magnetic tape. Permanent datasets on disk can be destroyed or replaced if the permission was granted when the dataset was defined.

Dataset Management Commands

The *directory file* is maintained on a userid if datasets have been created on that id. This directory is read automatically at the start of a Micro session, making the datasets listed in it available for use. Otherwise, the Get command is used to read the directory on another id, to make datasets stored there available.

The Replace command replaces an entire dataset with another set. The sets need not have identical structure, i.e., the same number and types of fields. The replacement set can be the Result set, as in the above example, or a temporary or permanent dataset. The dataset must be *replaceable*, an attribute set when the dataset is created, as explained in the next chapter on Dataset Structure.

The Destroy command destroys the named dataset. The dataset must be *destroyable*, an attribute set when it is created, like the replaceable attribute.

The Release command releases the storage associated with datasets and unlocks the set's files. The directory denoting the dataset's name, description and files is still available. Releasing a temporary dataset is the same as destroying it, in which case directory information is no longer available.

The Purge command releases all datasets, permanent and temporary, acquired during the session, and re-reads the directory on the userid used to signon, as is done at the start of a Micro session. The Purge command "re-initializes" the Micro session, in effect.

Printing Datasets as Tables for Wordprocessing Programs

The second form of the Print command prints the Students dataset on a file for downloading to a microcomputer, and importing as a table into a word processing program. In addition to RTF (Rich Text Format), supported by Microsoft products, tables can be produced in IBM's DCA (Document Content Architecture) format, used by IBM products, two formats for $\TeX$, the typesetting program used in academic circles, and STAT:Textedit, the text processing program that runs on MTS.

Exporting Datasets to Other Programs

The Export command can generate Micro datasets in forms importable by microcomputer spreadsheet programs and mainframe and microcomputer statistical analysis software. In addition to SPSS-X, formats are available for SPSS/PC+, the microcomputer version; for all versions of SAS; for OSIRIS IV, the statistical analysis program maintained by the Institute for Social Research at the University of Michigan; for WK1, the format for Lotus 1-2-3 and many other spreadsheets; and ASCII format, all values in ASCII characters, separated by commas, with non-numeric (character) values in quotes.

Dataset Documentation

Documentation is provided in three formats, *descriptive*, *technical*, and *full*. The descriptive format includes information necessary to work with the dataset in Micro, such as the dataset description, and field and category names, descriptions and abbreviations. The technical format excludes field and category descriptions, but includes additional information such as field length, type, displacement, scaling factor and function. The full format includes user and technical information, and dataset information as well: replaceable, destroyable and encrypted status, and file names.

These formats distinguish most of the commands in the Micro system that produce documentation, with the exception of the Describe command, which produces dataset, field and category descriptions. All of these commands can document either an entire dataset or a subset of fields or categories.

Micro Dataset Documentation Commands

Command	Example	Effect
DEscribe	`desc in students f1`	gives description of field F1 in dataset Students
DOcument	`doc in students fields`	gives description of fields in dataset Students
FUlldoc	`fu students`	full (descriptive and technical) documentation for dataset Students
TEchdoc	`te students`	technical documentation for dataset Students

Data Entry

Data may be entered in Micro in free format or fixed format, interactively or machine-read, by a Micro command or by stand-alone programs. The following table briefly indicates the data entry capabilities of Micro commands. Data entry is discussed at length elsewhere.

Micro Data Entry Commands

Command	Example	Effect
Enterdata	`ent in students from -newstu with errors on -err`	enters data from file -NEWSTU
UPdate	`up students ids=sin ups=class-year from -fixstu`	for records in Students having field SIN in records in -FIXSTU, updates field Class-year

Note that the Enterdata command enters data into all fields of a dataset; the Update command enters data into selected fields. Both of these commands are available as stand-alone programs, ILIR:Micro.Add and ILIR:Micro.Up.

Micro Programming Facility

The Micro Programming Facility allows functions written in an augmented version of the C programming language to be called and receive parameters from the Micro command prompt. The augmented syntax allows declaring such functions, known as MPF commands, and allows generating and executing Micro and MPF commands. The source code is processed by a preprocessor, which passes through standard C and generates C source code for the special syntax. The resulting code is then compiled with the MTS C compiler. The preprocessor also creates a library of MPF commands, which may be activated during a Micro session.

The functions in an MPF library may call Micro Interface Functions to reference Micro datasets, fields, categories and individual records. An interface to the Micro command interpreter is also available. An MPF program may be a complete front end to Micro, handling all interaction with the user.

Miscellaneous Commands

The last group of Micro commands performs miscellaneous tasks.

Miscellaneous Commands

Command	Example	Effect
COMMent	`comm RAJ performed`	none; appears with other commands on output
Display	`dis cost`	displays cost of Micro session
Help	`help`	enters Micro Help System, which displays topics for selection
MEssage	`mess "Results of RAJ"`	prints message on terminal; entire command appears on output
MTs cmd	`mt`	returns temporarily to MTS command mode, allowing re-entry with $Restart
SET	`set cmd:  output -out`	writes Micro output on file -OUT
RESEt	`reset cmd:  output`	restores default value for output destination (terminal)

The Comment command is useful in command files, to explain the action of the commands.

The cost printed by the Display command includes four figures, the amount spent in the command so far, the amount spent in Micro, the surcharge portion of the Micro cost, and the cost of the MTS session. The Display command can also show the time, date, and amount of virtual memory in use.

The Help command enters the menu-driven Micro Help System, where topics can be selected by item number. A null response returns to the previous menu, eventually to command level. The tutorials illustrate use of the Help System. The help text is the Micro Command Reference for ILIR:Micro, in Part 3 of this manual.

The Message command prints a message on the terminal; the command itself appears with others in the Micro output.

The MTS command returns to MTS command mode. $Restart can be used to re-enter Micro. MTS commands can also be issued directly from Micro, when prefixed with a dollar sign.

The Settings command sets parameters which affect the value of various commands, or of Micro in general. The syntax shown in the example, '`cmd: ...`', denotes the functional area affected, in this case general command behavior, and the aspect affected, '`output`', to be sent to file '`-out`' until further notice.

The Resettings command restores a Micro parameter to its default value.

The Settings and Resettings commands in the Command Reference shows all the functional areas, and all the aspects in those areas, which may be set. Settings for specific commands are described in the Command Reference for those commands. For instance,

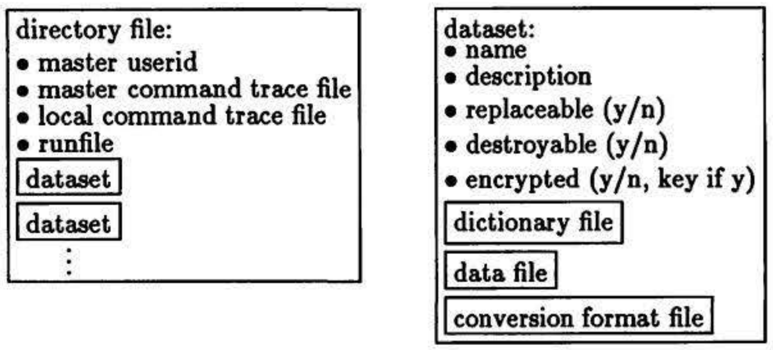
```
   set join: gf 5
```

sets the growth factor, which limits the size of Result sets created by the Join and RestrictAndJoin commands.

Dataset Structure

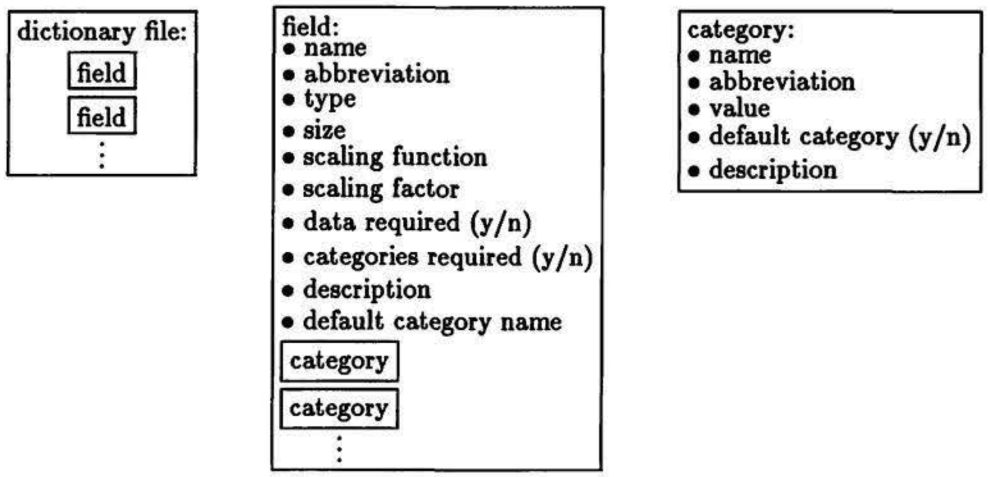
Introduction

The following diagrams summarize the structure of Micro datasets.

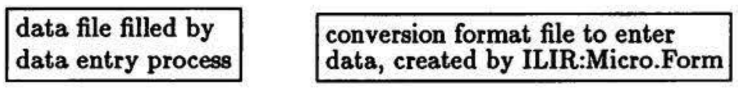


A *directory file*, named DIRECTORY, contains Micro run-time parameters, and a list of datasets available for analysis. The entry for each dataset consists of several attributes, including the name of the *dictionary file*.

The dictionary file describes the structure of the dataset, which is made up of fields. Each field has certain attributes, including one or more categories if necessary. Data is categorical when it represents a group or stratum or other classification (like gender) rather than an analytical value. A category in turn has its attributes.



The *data file* is filled by one of several possible methods of data entry. The optional *conversion format file*, created by ILIR:Micro.Form, describes the format, in terms of record size, starting position, and length of fields, of data in fixed format to be entered into Micro.


data file filled by data entry process
conversion format file to enter data, created by ILIR:Micro.Form

Directory File

Run-time Parameters

Micro run-time information, and information on datasets, is stored in a *directory file* (named DIRECTORY), on the userid where the dataset is created. This information is created and maintained by ILIR:Micro and ILIR:MicDef. The user's directory file is read automatically, and the datasets described made available, when Micro is run. Directories on other userids can be read with the Micro Get command.

The run-time information consists of the *master userid*, and the names of up to three files. The master userid defaults to the current userid during a Micro session, if not explicitly set with the Set command in Micro. It can also be changed with ILIR:MicDef.

Two of the three files are used by the Micro command trace facility. The first time this facility is activated, with the Set command, master and local command trace files, named MASTERCMT and LOCALCMT by default, are created. The master file is created on the master userid, the local file on the current userid. When command tracing begins, if the local command trace file is not empty, it is copied to the end of a master command trace file, and then emptied. For each Micro command executed, the date, time, userid, command name and command text are written to the local file. If the session ends normally, the local file is copied to the end of the master file, and then emptied. The names of both command trace files can be changed in ILIR:MicDef.

A *runfile* can also be stored in the directory file. There is no default name. Such a file contains commands which are processed when Micro is $Run, before any command prompting. It is used to initialize a session with Micro. The runfile is specified with the Set command, and can also be changed with ILIR:MicDef.

Dataset Information

When a dataset is created a line containing certain information is inserted in the directory file.

A dataset has a name, which may be up to 16 characters long. A description, up to 979 characters, can also be supplied with a dataset. Micro names, including datasets, fields and categories, are case-insensitive. Most (but not all) keyboard characters may be used in Micro names:

```
a-Z  0-9  #  $  %  ^  *  _  -  +  {  }  [  ]  /
```

(There may be no embedded blanks in names.)

Each dataset has at least two files associated with it, a *dictionary file*, which describes the structure of the data, and a *data file*, which contains the actual data. These files are MTS sequential files. The default file names are the dataset name followed by '#' for the dictionary file and '$' for the data file. If the default names are used, the dataset name should be unique (within a directory) in 11 characters, and further restricted to these characters:

```
a-Z  0-9  $  *  -  %  #  /  .  _
```

These are legal characters for MTS file names from the Micro name set given above. In practice, the programs create datasets and their files using the default names, unless they encounter a duplicate or an illegal name, when they prompt for a replacement.

A third file, the *conversion format file*, contains the format of input data in terms of record size, starting position and length of fields, if data are entered in fixed-format from a file or other source. The format file is created with ILIR:Micro.Form.

In addition to the dataset name, description, and file names, the directory file contains three switches indicating whether the dataset is *destroyable*, *replaceable*, or *scrambled* (encrypted).

A *destroyable* dataset can be destroyed with the Micro Destroy command. When a dataset is destroyed, its entry is deleted from the directory and the associated files are destroyed if no other datasets are using them.

A *replaceable* dataset can be replaced by another, with the Micro Replace command. The replacement set need not have the same structure (number and types of fields) as the set being replaced.

A *scrambled* dataset has its data encrypted for security reasons. The *scramble key* is a string of up to 40 characters which is prompted for when data is entered, and when the dataset is first referred to in Micro. The key is not stored, and data encrypted cannot be decrypted by the Micro staff if the key is lost. This key is case-sensitive; capitalization must be the same each time it is used.

Dictionary File

Fields

Fields are the components of records in which data is stored. A field of a dataset exists in all records in the dataset, like a column in a table. Fields are referred to by name, abbreviation or number (e.g., F1, F2, by order of appearance in the record) in Micro commands, to perform operations on the data. Fields are defined and edited through ILIR:MicDef, the dataset definition program.

A field has a name of up to 16 characters, and an abbreviation of up to four characters. The name and abbreviation for a field may be the same, but must be unique among other field names and their abbreviations within the dataset. The name and abbreviation must also be unique among categories within the dataset. The default abbreviation is the first four characters of the field name. A description of up to 979 characters also may be provided for a field.

Data in Micro may be stored in seven ways:

Field Types in Micro

Type	S	SC	U	UC
Contents	signed integer	signed integer with with categories (for data coded as integer)	unsigned integer	unsigned integer with categories (for data coded as integer)
Type	C	CC	E	
Contents	character, fixed-length, <=255	character with categories (for data <= 4 chars, coded as character)	external (variable-length >255 chars if necessary)	

At the moment, there is no real (floating-point) field type in Micro. Real values can be stored as signed or unsigned integers. A scaling function and factor can be supplied as attributes of the field, and they will automatically be applied to the real value during data entry, and to the internal value when it is output. Scaling is discussed below.

Field size for numeric values is at most four bytes. Character fields with categories are limited to four bytes. Fixed-length character fields without categories may be up to 255 characters long. Variable-length character fields of up to 32,767 characters can be stored as *external* fields. A file name must be supplied when an external field is created. When data is entered into an external field, Micro stores data in the external file, and the four-byte MTS line number in the data file. Data in external fields is not available for operations in Micro which refer to field values.

Size of a numeric field is determined when the field is created in ILIR:MicDef, implicitly, from the maximum absolute value to be stored, or explicitly.

Minimum and Maximum Values by Field Size and Type

Size in	Type U/UC		Type S/SC	
Bytes	Min	Max	Min	Max
1	0	255	−128	127
2	0	65535	−32768	32767
3	0	16777215	−8388608	8388607
4	0	2147483647	−2147483648	2147483647

When an integer field is defined, one of the four operations of addition, subtraction, multiplication and division, may be specified as a *scaling function*. If a function is supplied a *scale factor* is also required. The function and factor are applied when data is entered, and when data is output. For example, dollar values entered as *d.dd* are stored as *ddd*, and output as *d.dd*, where *d* is a decimal digit. The maximum value to be stored, used in calculating field size, is the value *after* the scaling function and factor have been applied. Factors for multiplication and division must be a power of ten.

The setting of the *data required* switch affects data entry. If the switch is set to yes, a null reply is not accepted, if data is entered interactively. All fields must have values if data is read from a file.

The setting of the *categories required* switch forces data to be the value of one of the field's categories. Other values are rejected, if data are entered interactively. If data are machine-entered, the record is rejected.

Character fields may defined to overlap. A large, inclusive field must be defined first, then the overlapping fields. For example, a field for address could include overlapping fields for street, city, state, zip code, etc. Overlapping is indicated when the length of a subfield is being defined. The name of the inclusive field is given for the length, and ILIR:MicDef prompts for a starting column and length for the subfield.

Categories

Data is *categorical* when it represent groupings, strata, or some other descriptive classification, rather than an analytical value. A field for gender might have categories representing female, male and unknown (missing data).

A category has a name of up to 11 characters, an abbreviation of up to four characters, and a value. The name and abbreviation may be the same, but must be distinct from names and abbreviations of other categories within the field, or other field names and abbreviations within the dataset. The value is the data value that will be entered to represent the category. It is used to match category name and abbreviation to data. The category value must fit within the field. For CC fields, it may be up to the length of the field; for UC or SC it may not exceed the min/max allowed for the field size. Category names and abbreviations are case-insensitive, and may be given in upper or lower case. Category values are case-sensitive, and must match the data value exactly, when the field is type CC, character with categories. In such a field, representing sex, one category name might be Female, another Male, another Unknown, with abbreviations F, M and U, and values '`f`', '`m`', and '`u`'.

Category names, abbreviations and values are used on Micro commands to test field values, e.g., retrieve all records where '`sex=female`' (category name), '`sex=F`' (abbreviation, but not value), or '`sex=f`' (abbreviation or value) for the type CC example.

For a type UC field (unsigned integer with categories) representing sex, one category name might be female, another male, another unknown, with abbreviations F, M and U, and values 1, 2 and 0. The retrieval would be for all records where '`sex=female`' (name) '`sex=f`' (abbreviation) or '`sex=1`' (value). Retrieval by category name and abbreviation are the same, whether the field is numeric or character. Only retrieval by value changes with the field type, if the category name and abbreviation are the same; the category name and abbreviation "mask" the underlying value.

A default category may be specified when a field is defined. The default value is assigned at data entry, if no value is supplied and data is not required. The default category value is also used when filler values are assigned implicitly by the Micro Join command.

When data is entered, data values may be restricted to category values, if the `category required` switch is set to yes when the field is defined. The category value, name or abbreviation may be entered, and the value is stored.

A description may be supplied when a category is defined, and may be up to 979 characters long.

Security

Datasets may be encrypted, with a user-supplied key, which is not recorded or stored by the Micro programs, and remains unknown to the Micro staff. Encryption is specified when the dataset is defined, in ILIR:MicDef, and the key is prompted for each time data is accessed. The command trace facility can be enabled to record all operations on datasets, to maintain data integrity. MTS file permissions restrict access to datasets as necessary.

Documentation for Datasets

Documentation for the dataset structure described above—dataset filenames, switch settings, field and category names, abbreviations, values, sizes, etc., can be produced in ILIR:MicDef, the dataset definition program, and from within ILIR:Micro.

Documentation is provided in three formats, *descriptive*, *technical*, and *full*. The descriptive format includes information necessary to work with the dataset in Micro, such as the dataset description, and field and category names, descriptions and abbreviations.

The technical format excludes field and category descriptions, but includes additional information such as field length, type, displacement, scaling factor and function.

The full format includes descriptive and technical information, and dataset information as well: replaceable, destroyable and encrypted status, and file names.

These formats distinguish most of the commands in Micro that produce documentation; the Describe command produces the dataset, field or category description. All of these commands can document either an entire dataset or a subset of fields or categories. An asterisk, when appended to a documentation command, omits category documentation.

Micro and MicDef Documentation Commands

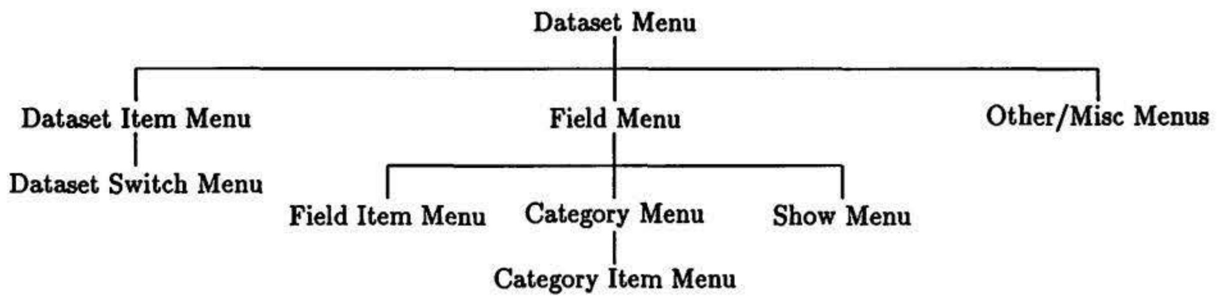
Command	Descriptions	Dataset Information	Descriptive Information	Technical Information	Field and Category
		ILIR:Micro			
DEscribe	×				×
DOcument	×	×	×		×
FUlldoc	×	×	×	×	×
TEchdoc	×	×		×	×
		ILIR:MicDef			
Describe	×	×			
DOcument			×		×
FDoc	×	×	×	×	×
TDoc				×	×

Dataset Definition

The dataset definition program, ILIR:MicDef, can be used in menu mode, command mode, or MTS macro mode. In menu mode, the program accepts choices from menus and prompts for the information it needs. Command mode allows a more experienced user to act directly. MTS macro mode allows an entire dataset definition to be entered line by line into a file, which MTS macros then feed to the program. The menus, commands, and an example definition file are discussed in this section. The tutorial on Database Definition further illustrates the use of all three modes.

Menu Mode

Menu mode is entered automatically when ILIR:MicDef is run. The menus are organized according to the following diagram.



The Dataset Menu is entered first. Item 6 allows command mode to be entered immediately rather than continuing in menu mode.

```
Dataset menu                               Active dataset: "No.Active.Set"

     1. List existing datasets             5. Find datasets matching pattern
     2. Activate a set and edit fields     6. Exit menu (go to command mode)
     3. Change/describe dataset info       7. Help
     4. Other/misc commands                8. Stop

          Enter command:
```

Each menu contains a Help item, Item 7 in this menu, which allows the user to select help on the items in the menu.

Entering '1' (or 'l'), for '`List existing datasets`', lists the datasets the program knows about, from the directory file on the current userid.

Entering '2' (or 'a'), for '`Activate a set and edit fields`', causes the program to prompt for a dataset name. If the dataset is new, the program prompts for dataset attributes. Then it prompts for the field and category attributes, one field at a time. A null reply (pressing the Return key without having typed anything) at the start of the field prompts completes the definition and causes a return to the Field Menu.

Entering '3' (or 'c'), for '`Change/describe dataset info`', causes a prompt for a dataset name, if there is no active set, and then enters the Dataset Item Menu. After a dataset has been created, this menu can be used to change some of the default values for attributes.

```
Dataset item menu                              Active dataset: "ART"

     1. List files in active set            7.  Edit the dataset description
     2. Pattern matching dataset names      8.  Set scramble, dest & rep swit
     3. DSN (dataset name) change           9.  Redo dataset info
     4. DICtionary file name change        10.  Help
     5. Micro data file name change        11.  Back to master menu (return)
     6. Format file name change

          Enter command:
```

Item 1 lists the names of the dictionary, dataset and conversion format files for the dataset. Item 2 takes a *pattern*, such as '`xyz?`' and lists all datasets beginning with '`xyz`'. Items 3-7 change various dataset attributes. Item 8 enters the Dataset Switches Menu.

```
Switches menu                                  Active dataset: "ART"

     1. Replace switch                      4.  Help
     2. Destroy switch                      5.  Back to dataset files (return)
     3. Scramble switch
           Enter command:
```

Entering '1', '2' or '3' causes prompting for the new value of the switch. A null reply returns to the previous menu, in this case the Dataset Item Menu; another null reply returns to the Dataset Menu.

Item 2 in the Dataset Menu, '`Activate a set and edit fields`', causes a prompt for a dataset name. If the set already exists, the Field Menu, shown here, is entered. The dataset becomes the active dataset.

```
Field menu                                     Active dataset: "ART"

     1. Append fields (terse)               6.  Modify/add categories
     2. Insert fields                       7.  Show documentation
     3. Redo a field                        8.  Help
     4. Delete fields                       9.  Back to dataset menu (return)
     5. Change field item

          Enter command:
```

Items 1-4 cause prompts for the information necessary to perform those functions. Items 5, 6 and 7 produce further menus. Item 5 is the Field Item Menu:

```
Field item menu        Dataset: "ART"        Field: "No.active.field"

    1. Name                                  7.  Categories required
    2. Abbreviation                          8.  DATA required
    3. Length or max value                   9.  View field
    4. Scale  function                      10.  Edit another field
    5. Factor                               11.  Help
    6. Description                          12.  Back to field menu (return)

        Enter command:
```

Items 1-8 correspond to the field attributes described in the section on Dataset Structure. Item 9 "views" or documents the current status of those attributes. Item 10 prompts for the name of another field, which then becomes the active field.

Item 6 from the Field Menu is the Category Menu:

```
Category menu          Dataset: "ART"        Field: "No.active.field"

    1. Append categories                     6. Show categories
    2. Insert categories                     7. READ categories from a file
    3. Redo a category                       8. Edit another field
    4. Delete categories                     9. Help
    5. Change category item                 10. Back to field menu (return)

        Enter command:
```

Items 1-4 cause prompts for the information required for those functions. Item 6 "shows" or documents the categories. Item 7 reads category definitions from a file. Item 8 allows another field to be edited directly, without first returning to the Field Menu. Item 5 from this menu is the Category Item Menu:

```
Category item menu     Dataset: "ART"        Field: "No.active.field"

    1. Name                                  5.  Description
    2. Abbreviation                          6.  Edit another field
    3. Status (default)                      7.  Help
    4. Value                                 8.  Back to categorymenu (return)

        Enter command:
```

Items 1-5 correspond to the category attributes described in the section on Dataset Structure. Item 6 allows another field to be edited without first returning, through two menus, to the Field Menu.

Item 7 from the Field Menu is the Show Menu, which provides choices for documenting the field in different formats:

```
Show menu                                    Active dataset: "ART"

     1. Fulldoc on screen                  6.  PF (Print Fulldoc on file
     2. Doc on screen                      7.  PD (Print Doc on file)
     3. Tdoc on screen                     8.  PT (Print Tdoc on file)
     4. DT (Doc Terse)                     9.  Help
     5. TT (Tdoc Terse)                   10.  Back to field menu (return)

          Enter command:
```

The various formats are discussed in the section on Dataset Structure above.

Item 4 from the Dataset Menu, '`Other/misc commands`', enters the Miscellaneous Menu.

```
Miscellaneous menu                           Active dataset: "ART"

     1. Transfer sets from another USERID   6. Master cmd trace file change
     2. Permit sets to another USERID       7. Write changes to file
     3. System master signon IDchange       8. Destroy dataset
     4. Runfile name change                 9. Help
     5. Local cmd trace file change        10. Back to master menu (return)

          Enter command:
```

Item 1 and 2 allow datasets to be moved or copied between different userids. Items 3-6 are the Micro run-time parameters stored in the directory file and discussed in the section on Dataset Structure. Item 7 makes permanent any changes to attributes of the active dataset. Item 8 destroys the active dataset.

Command Mode

Command mode in ILIR:MicDef can be entered from the $Run command line in MTS

```
   $Run ILIR:MicDef par=nomenu
```

or from the Dataset Menu shown above, which is displayed by default. Command mode provides all the dataset editing facilities of menu mode.

MicDef Dataset Editing Commands

Command	Example	Effect
Alter	`alt xyz maxvalue to 255`	changes the maximum value of field **xyz** in the active dataset to 255
APpend	app	prompts for field attributes, appending the new fields to the end of the dataset
COpy	`cop from dset f2 f3 to f2`	copies the second and third fields of dataset **dset** to just after the second field of the active dataset
DELete	`del xyz`	deletes field **xyz**
Insert	`ins f7`	prompts for field attributes and inserts new fields after the seventh field
MOve	`move f5 thru f7 to f2`	moves the fifth through seventh fields to just after the second field
Redo	`redo f4`	"redoes" the fourth field in the active dataset (causes the field attributes to be prompted for again)

The Alter command can change dataset, field and category attributes, with the exception of field type. It can also change Micro run-time parameters like the names of command trace files, etc.

The Append, Delete, Insert and Move commands can perform their operations on categories within a field, as well as on fields within a dataset, as shown here.

The Append command can also read category definitions from a file. Each category definition should occupy one line in the file, containing the name, abbreviation, 'y' or 'n' to indicate whether the category is the default category, the category value, and the description. A numeric field for gender might have categories:

```
female f n 1 female
male m n 2 male
unknown u y 0 unknown
```

The Redo command can redo dataset, field or category attributes. The difference between Redo and Alter is that Redo prompts for the entire list of attributes of the item being changed. Also, the field type cannot be changed with Alter, only with Redo.

MicDef Dataset Management Commands

Command	Example	Effect
DESTroy	`dest`	destroys the active dataset
EDit	`edit students`	activates the dataset; if it doesn't exist it is created, with default values for attributes
Permit	`permit copy students to kabc`	permits the dataset '**students**' to be copied by account KABC
Get	`get copy students from kabc`	makes a copy of the dataset **students** from the userid KABC on the current userid

The Destroy command destroys the dictionary file, but not the data file, which is left unchanged, and deletes the directory entry.

The Edit command makes the named dataset the active dataset, creating it if it doesn't exist.

The Permit command permits a dataset's files to allow them to be moved or copied to another *userid*, KABC in this example.

The Get command copies or moves a dataset from one *userid* to another. The dataset must first be permitted to the userid to which it is to be transferred. This is done by running ILIR:MicDef from the userid which owns the dataset, and issuing the Permit command on behalf of the userid which is to receive the dataset.

Dataset Documentation Commands

Command	Example	Effect
Describe	`desc`	gives attributes of the active dataset
DOc	`doc f3 thru f4`	gives descriptive information of the third and fourth fields in the active dataset
FDoc	`fd`	gives descriptive and technical information on all fields in the active dataset
TDoc	`td f4`	gives technical information on Field 4 in the active dataset

The Describe command documents datasets, the Doc, FDoc and TDoc commands fields and categories. The Doc command produces only descriptive information, the FDoc command produces descriptive and technical information, and TDoc technical information.

Miscellaneous Commands

Command	Example	Effect
HELP	`help`	prints a menu of help topics in menu mode, lists commands in command mode
Menu	`menu`	returns to menu mode

The Help command presents help topics organized by command.

The Menu command returns to menu mode.

MTS Macro Mode

The file ILIR:MICDEF.MAC is an MTS macro library whose contents invoke ILIR:MicDef and generate responses to its prompts. The file ILIR:DATASET.GEN is an example of a file which uses the macros. The dataset definition is specified in a series of macro invocations, one to run ILIR:MicDef, one for each dataset, one for each field, and one for each category. The attributes are arguments to the macro. The dataset definition process for the ILIR:DATASET.GEN example file is invoked in MTS command mode by:

```
#$set macros=on
#>macrolib ilir:micdef.mac
#source ilir:dataset.gen
```

The first two lines activate the macro library, the third invokes an MTS command file which invokes the macros. The notation is general and simple enough to meet most needs, though the macros contain default values for some attributes. Lines beginning with the words **dataset**, **field** and **category** define those items. Attributes of those items, such as dataset name, follow on the line, as indicated by the following table.

Dataset Definition Notation

Macro	Attributes							
dataset	**Dataset_Name**	**"Dest"**	**"Repl"**	**"Scram"**	**Dict=""**	**Data=""**	**CFF=""**	**Desc=""**
defaults:	×	destroyable	replaceable	not scrambled	**dataset#**	**dataset$**		
field	**Field_Name**	**Field_Abr**	**Field_Type**	**Length**	**"DReq"**	**"CReq"**	**Scale=""**	**Desc=""**
defaults:	×	×	×	×	not reqd	not reqd		
cat	**Cat_NAme**	**Cat_Abbr**	**Cat_Value**	**"Default"**	**Desc=""**			
defaults:	×	×	×	not default				

For the **dataset** macro, there are, left to right, one required positional attribute, **Dataset_Name**, three optional dataset switches, **Dest**, **Repl**, **Scram**, and four optional keyword attributes, **Dict**, **Data**, **CFF**, the dictionary, data and conversion format files, and **Desc**, the dataset description.

The dataset name is required (default value of '**x**') and must appear immediately after the word '**dataset**' at the start of the line. (Required attributes, such as the dataset name, or the first four items of the field definition, must appear in the order given in the table). Attributes in quotes, such as **"Dest"**, the destroyable switch, may appear literally ('**dest**', for "destroyable"), negated ('**nodest**', "not destroyable"), or not at all (defaulting to "destroyable"). Keyword attributes (**Dict=""**, for dictionary file name) are always optional, but must be given in keyword form, e.g., '**dict=ccid:globaldict**'.

The **field** macro assigns the attributes **Field_Name**, **Field_Abr**, **Field_Type**, and **Length**, which are required in that order. The switches **"DReq"** and **"CReq"** are optional. The **Scale=""** and **Desc=""** are optional and must be given in keyword form.

The value supplied for **Length** depends on the field type. For numeric fields (types S, SC, U and UC) the length attribute can be a maximum absolute value, from which is calculated the field size in bytes. Or it can be a size, prefixed with '**b**', for bytes, to denote size rather than value. For character fields (types C and CC) the length attribute must be the size in bytes. For external fields (type E) the length is the external file name, and can be the field name.

The `category` macro requires a `Cat_NAme`, a `Cat_Abbr` and `Cat_Value`. The `"Default"` switch defaults to off if omitted, and the `Desc=""` is in keyword form if given.

The Fin-Aid definition is:

```
micdef
dataset Fin-Aid nodest norepl desc="Example data set -- Financial aid information"
  field Student-ID-Num SIN U 999999999 dreq desc="Student Identification Number"
  field Year-Awarded Year U 99 dreq desc="Year Awarded (YY format)"
  field Term-Awarded Term UC 3 dreq creq desc="Term Awarded"
    cat Fall F 1 desc="Fall Term"
    cat Winter W 2 desc="Winter Term"
    cat Summer S 3 desc="Summer Term"
    endcats
  field Class-Level CL UC 4 dreq creq desc="Class Level"
    cat Freshman Fr 1
    cat Sophomore So 2
    cat Junior Jr 3
    cat Senior Sr 4
    endcats
  field Type-of-Aid Type UC 6 dreq creq desc="Type of Aid"
    cat Tuition Tu 1
    cat Room-Board RB 2 desc="Room and Board"
    cat Cash Ca 3
    cat Loan Lo 4
    cat Work-Study WS 5 desc="Work Study Program"
    cat Other Oth 6
    endcats
  field Source-of-Funds SOF UC 6 dreq creq desc="Source of Funds"
    cat University Univ 1
    cat State-Gov SGov 2 desc="State Government"
    cat Federal-Gov FGov 3 desc="Federal Government"
    cat Industry Ind 4
    cat Foundation Foun 5
    cat Other Oth 6
    endcats
  field Amount Amt S b4 scale=/100 dreq desc="Amount of Award"
  field Remark* Rem* E Remark desc="Additional Comments"
  endfields
enddef
```

The line '`micdef`' invokes ILIR:MicDef. The lines beginning with '`dataset`', '`field`' and '`category`' define elements of a dataset. The lines '`endfields`' and '`endcats`' denote the end of field and category definitions. The Micdef macro can be copied and modified as desired.

Data Entry and Updating

Data may be entered in the Micro system interactively (in free format), or by machine, in free or fixed format, via ILIR:Micro or with a stand-alone program. Existing data may be updated by the same means, except that no method of updating from fixed-format data is available. The methods are summarized by the following table, and discussed below.

Methods of Data Entry and Updating

Format	Method
Interactive (free format)	Micro Enterdata and Update commands or stand-alone programs ILIR:Micro.Add and ILIR:Micro.Up
Free format	Micro Enterdata and Update commands or stand-alone programs ILIR:Micro.Add and ILIR:Micro.Up
Fixed format	format, in terms of record size, field starting position and length, and other data, specified with ILIR:Micro.Form; data read with ILIR:Micro.Cnvrt
One of the above	Custom programming—user program generates a file to be read by Micro data entry programs

Interactive and Free Format

The stand-alone programs are discussed first, since they have behavior in common with the Micro data entry commands.

ILIR:Micro.Add and ILIR:Micro.Up

The syntax for invoking ILIR:Micro.Add or ILIR:Micro.Up is the same:

`$Run ILIR:Micro.Up [input=`*file* `print=`*file* `object=`*file* `par=freeform dsn=`*dataset* `time={`*n*`|OFF}]`

`$Run ILIR:Micro.Add [input=`*file* `print=`*file* `object=`*file* `par=freeform dsn=`*dataset* `time={`*n*`|OFF}]`

If the optional parameters (in brackets) are not given, the programs will prompt for data interactively.

If the `input` and `object` files and the `par=freeform` parameter are given, the programs read data from `input`, and write error records on `object`. If `print` is given, error information is written there instead of on the screen. The '`time=`' gives a time limit for execution of the program, or disables it. If omitted, the program sets its own limit of 5 CPU seconds. If *dataset* is given, the following prompts are suppressed. If not, both programs prompt for one or more of the following:

- Micro dataset name;
- userid of another directory (or '`cancel`');
- data file name; and
- scramble key.

Micro dataset name

This may be either the name of an existing dataset or the name of an existing Micro dictionary file (indicated by a four-character userid on which the dictionary is stored followed by a colon (`:`) and the dictionary file name). If no reply is given the program terminates.

Userid of another directory or '`cancel`'

This is requested only if the dataset name cannot be found in the directory that was searched or if that directory does not exist or is not accessible. A four-character userid is all that is required. If '`cancel`' is the reply, a new dataset name is prompted for and user's directory is searched.

Data file name

This is requested only if a dictionary file name was given instead of a Micro dataset name. The file may not exist, may exist but be empty or, may exist and contain data. If it does not exist, it is created. If it exists then it must be a sequential file permitted RW (read-write access). If it contains data then the data must be compatible with the dictionary. A reply is required.

Scramble key

This is requested only if the dictionary indicates that the dataset is or should be scrambled. The same key is used both to scramble and to unscramble a dataset. The user must remember the key because it is not stored permanently by Micro. Garbled data indicates that the wrong key was used to unscramble the dataset. A reply is required.

Micro.Up prompts for *identifying* and *update* fields. The identifying fields are those whose values will identify records in which the values of update fields will be changed. The field names or abbreviations must be separated by blanks, and may be continued to another line by ending the current line with a dash (`-`).

The update field list is given in the same fashion, field names or abbreviations separated by blanks, lines continued with a dash; identifying fields may also be updated. These two lists of fields constitute an intermediate dataset, used in updating the permanent set. The input data must correspond in type and size to the identifying and update fields, in the order given.

Free format data, whether entered interactively, or by machine from a file, must still fulfill certain (minimal) requirements. Each method has the same requirements, whether used in Micro.Add (Enterdata) or Micro.Up (Update).

Interactive

Interactive data entry requires no programming skills. Data is prompted for field by field. Data validation is performed according to dictionary information for the dataset, and all errors are corrected at data entry. This is the most costly method, in terms of computing cost and labor time, and is usually suitable only for small amounts of data.

Running ILIR:Micro.Add or ILIR:Micro.Up is explained above. The syntax for using the Enterdata and Update commands interactively is given in the Micro command reference section. Micro.Add asks '`After how many records should data be saved?`'. It will accumulate this number before writing them to the data file. Enterdata asks this question if data is being added to a permanent dataset. Except for this difference and the information requested at the outset, behavior of Micro.Add and Enterdata is the same. Micro.Up does not ask that question; except for the information requested at the outset by Micro.Up, behavior of it and Update is the same.

The programs ask if abbreviated prompting is desired. If the answer is '`yes`', field abbreviations are used in prompting for data, otherwise the full names are used.

Data for more than one field may be entered at once. Prompting is suppressed for fields for which data is entered in this manner. If more values are provided than fields remain to be filled, a warning message is printed and the extra data is ignored. The input for each field must be separated by one or more blanks. A category may be given for a field by entering the category name, abbreviation or value. If the categories required switch for the field is '`yes`', one of those must be given.

If a character value contains embedded blanks it must be enclosed in right single quotes (`'A B'`). A right quote inside a string is represented by doubling it (`'Bill''s'`). A long character value may be continued to the next line if the dash (-) is the last character on the line.

Data values for scaled fields should be entered in the way the user views the data. If field Amount is an unsigned integer scaled by division by 100 (two decimal places, as in dollars and cents), then 100 and 100.0 and 100.00 are all represented internally as 10000. Three dollars and forty-six cents is entered as 3.46, not 346.

If erroneous data (wrong type, or illegal value for field size, etc.) is entered, the program prompts for a new value. At that time documentation for the field and any of its categories may be obtained by typing a question mark (`?`).

If error correction occurs and multiple values have been entered, only the needed replacement value should be entered. The program remembers any additional values entered in anticipation.

After data values for all fields of a record have been entered, the program prompts '`Corrections, "STOP" OR "SAVE"?`' Values for any field may be corrected by entering:

field <verb> {*value*|*category*}

where *field* is the field name, <verb> may be an equal sign (`=`) or colon (`:`). The *value* should be an appropriate value, or a *category* may be entered by name in which case the value for the category is assigned. Only one correction may be given per line. If no more corrections are required, entering Return begins data entry on the next record.

A default value is entered for the field if a null reply is given. If the data required switch for a field is '`no`', and if there is a default category, the value of that category is assigned. If the field has more than one default category, the first one is always used. Otherwise, if a null reply is given and data is not required, blanks are assigned for character fields, zero for numeric. If data is required, the program prompts again.

The notation '`/D`' causes all fields of a record which have yet to receive data and for which the data required switch is '`no`', to be assigned their default value. This may also be done by sending an end-of-file indicator (by typing '<ctrl>-`C`' or '`$ENDFILE`'). Required fields will still be prompted for.

The notation '`/J`' jumps to the next required field, assigning default values to the jumped fields.

If a field is external (type E, for variable-length character fields), and the external data file does not exist, the program creates it, after prompting for authorization. When data for an external field is entered, a '`/S`' (for same) may be used to repeat the value from the previous record. The line number in the external data file associated with the previous character string is used for the field value. External data is never written to the external file until the record is complete, including corrections.

An MTS command may be given if it begins with a dollar sign (`$`) and appears in the first position of the line.

Free Format

If `input` and `object` are assigned on the $Run command which invokes Micro.Add or Micro.Up, or if files are supplied on the Enterdata or Update commands, the programs read data from `input` and write errors to `object`.

As with interactive data entry, more than one value may appear on a line if the values are separated by blanks. Character string values with embedded blanks must be enclosed in right single quotes; in strings a right single quote must be doubled (`'U M''S'`. Long character strings may be continued to the next line if a dash (-) is the last character on the line.

Input lines may be up to 32,767 characters long. An end-of-record indicator, '`/E`', must appear at the end of data for each record, as the last characters on the line. The first field of each record must begin on a new line. There must be sufficient data prior to the '`/E`' for all fields. If too many or too few values are provided before the end-of-record indicator, the record is rejected and an error printed.

Unlike interactive mode, there is no way to give a null reply for one particular field in order to give the field its default value, but fields for which the data required switch is '`no`' and which have not been assigned data are assigned default values if '`/D`' appears in the input. Data must be given for fields whose data required switch is set to '`yes`'.

The notation '`/J`' jumps to the next required field, assigning default values to the jumped fields.

As in interactive mode, '`/S`' may be used to repeat the data for an external field.

When the data for a field is incorrect, the program identifies the error, as in interactive mode. The number of the line containing the error, the number of the first input line for the aborted record and all of the input lines (with their line numbers) for the aborted record are printed on `print` (which defaults to the terminal if not reassigned).

The input lines for the aborted record are copied into `object` at the line numbers from which they were read. When data entry is complete, the error records in the `object` file can be corrected and then entered by running the program again, with this file assigned to `input`.

Data is written to the data file when enough records have been collected to fill as much of 32,767 bytes of storage as possible and/or when all data has been read from `input`. After the final save, the numbers of records read, rejected and added to the dataset are printed in addition to the new total number of records in the dataset.

Fixed Format

Two programs underlie this method, ILIR:Micro.Form and ILIR:Micro.Cnvrt. Micro.Form constructs the *conversion format file*, a description of incoming data in terms of record size, field starting position and length, and other information. Micro.Cnvrt uses the conversion format file to read data and build the Micro data file.

ILIR:Micro.Form

In this discussion, *card* means a physical record that is being converted to Micro format. If data are read from a file, a card is one line; if from a magnetic tape, one tape record; if from punched cards, one card. The word *record* means one logical record, which may reside on one or more cards.

The program is invoked in MTS by:

`$Run ILIR:Micro.Form [par=dsn=`*dataset* `dir=`*userid* `cff=`*file* `time={`*n*`|OFF}]`

No parameters are required, but if given they suppress the prompts described below. `dsn=`*dataset* identifies the dataset. `dir=` specifies what directory to get the dataset from. `cff=`*file* names the conversion format file. `time=`*n* sets a time limit, which defaults to 5 CPU seconds.

As necessary, Micro.Form prompts for:

- a Micro dataset name;
- Userid of another directory or '`cancel`'
- conversion format file name;
- the maximum total number of characters on all cards for one record;
- the maximum number of cards per record;
- whether the number of cards per record varies;
- criteria for including or excluding certain records; and
- a sampling string, for including a subset of records.

Dataset Name

The name of the Micro dataset into which the data will be entered or the name of an existing Micro dictionary file (indicated by a four character MTS userid followed by a colon, followed by the dictionary file name) must be provided.

Userid of another directory or '`cancel`'

This is requested only if the dataset name cannot be found in the directory that was searched or if that directory does not exist or is not accessible. A four-character signon-ID is all that is required. If '`cancel`' is the reply, a new dataset name is prompted for and user's directory is searched.

Conversion Format File

If the reply is a dictionary file name or if no conversion format file name was provided when the dataset was defined, Micro.Form asks for the name of a conversion format file. If the conversion file does not exist, the program will ask permission to create it. The conversion file will contain the output of the program, i.e. the conversion tables necessary to perform the actual data input and conversion. The dataset definition must be complete at this time. If no reply is given the program terminates.

Maximum Total Number of Characters for One Record

The maximum total number of characters for all cards in one record is requested. This is the maximum length of a card used, over all cards, times the number of cards in a record. A reply is required.

Maximum Number of Cards in One Record

The program starts with the first card and reads in N cards for each record, where N is the response to this question. A reply is required.

Varying Number of Cards Per Record

If the number of cards per record is greater than one, the program asks if the number of cards per record will vary. A reply is required. If the answer is '`NO`', the program proceeds to the next major item of information. If the response is '`YES`', the program prompts for the method of determining record boundaries. The user then enters one of the following three-letter abbreviations:

Record Boundaries for Fixed-Format Data Entry

Notation	Meaning
SUB	User-supplied subroutine (S-type calling sequence). The subroutine must read a record into a buffer. Arguments to the subroutine are the buffer (supplied by the program), maximum record length (supplied by the program), and actual record length (returned by the user subroutine) in that order. Return codes are: 0=successful read; 4=record is invalid and should be rejected; 8=no more records, end of file. ILIR:Micro.Cnvrt prompts for the subroutine name.
EOB	End of block indicator. When ILIR:Micro.Cnvrt encounters the specified string in the specified columns, it will treat the card being read as the last one in the record. The program prompts for the starting column and length of the EOB indicator and the indicator itself.
BOB	Beginning of block indicator. When ILIR:Micro.Cnvrt encounters the specified string in the specified columns, it will treat the card being read as the first one in the record. ILIR:Micro.Form prompts for the starting column and length of the BOB indicator and the indicator itself.
CHK	Changing key. This method can be used when every card for a record contains a key that is unique for that record. When the key on a card is different from the key on the previous card, the program treats the new card as part of the next record. A Social Security number, for example, might be used to tie together into one record all the cards containing information about one person. The program prompts for the starting column and length of the key field.

Filtering Records (Include/Exclude)

By entering 'E' the user can exclude all records that have a specified string in specified columns. By entering 'I' the user can include only records that have a specified string in specified columns. A null reply (Return) ignores this feature, including all records that are read. If the response is 'I' or 'E', the program prompts for starting column, card number and length of an identifying string, and the string itself.

Sampling String

The sampling string is a sequence of N 0's and 1's which for N records causes each record corresponding to a 0 to be excluded, each record corresponding to a 1 included. The string '110' includes the first two of each three records, creating a dataset with a two-thirds sample. A null reply (Return) ignores this feature. If sampling is in effect, it is performed *after* the I/E record inclusion/exclusion filter is applied.

Fixed Format Information

For each field of a dataset, the user is given the name of the field and then is prompted for format information:

- starting column;
- card number;
- input type;
- length;
- missing data value; and
- recode type.

Responses for these prompts may be typed on one line, separated by blanks.

Starting Column (or Subroutine File) The starting column is the position of the card in which the data for the field starts. Some of the flexibility of a program may be gained by using a subroutine which returns a value for a field. An S-type subroutine is indicated by replying with a four character MTS userid followed by a colon and the name of the object file containing the subroutine. The name of the subroutine is then prompted for. Arguments to the subroutine are:

subr(record, len, fld#, fldname, value)

The argument *record* is the buffer containing the entire record, *len* the length of the record, *fld#* the number of the field for which the subroutine is called, *fldname* the name of the field for which the subroutine is called. All of these are supplied by Micro.Cnvrt. The argument *value* is the value for the field, returned by the subroutine. This value is validated and recoded as usual. If the subroutine returns with a return code of 4 then the record is considered to be in error and is rejected. The record can be declared as a variable of appropriate length. All numeric values should be fullwords. A reply, either a starting position or a subroutine object code file, is required for this prompt.

Card Number Card number information is requested only if the maximum number of cards in one record is greater than 1. The reply indicates in which card the data for the field appears. A reply is required.

Input Type If character data (field type C or CC) is specified in the dictionary, no input type information is requested, and the program prints the message '`CHARACTER INPUT ASSUMED`'. The same is true of external data fields (type E). If the data is numeric, the program asks for input type, and the user should enter one of the following one-letter abbreviations:

Input Data Types

Notation	Meaning
B	Binary—data value is binary representation of the data. For signed fields (types S and SC) the incoming data is considered signed. Data for unsigned fields (types U and UC) is not signed.
I	Integer—data value is a positive or negative integer.
R	Real—data value is a positive or negative real number with a decimal point. The decimal point may occur in any column which is part of the number. Although Micro does not handle real numbers, during data entry a floating point value for the incoming number is created. After performing any optional floating point arithmetic operations the final floating point value is *rounded* (not truncated) to the nearest integer which is then stored in the dataset. By multiplying real values by a power of ten, using a recode, and using the division scale function, precision may be retained in Micro.
X	Hexadecimal—data value is read as hexadecimal characters. The incoming data is considered signed only if the field type is also signed (types S and SC). Use of this input type will be rare.

Length The length is the number of columns that the data for the field occupies in the card.

Missing Data Value The incoming data for a field is considered missing if columns contain all blanks or all minus signs. When data is missing, the value used for the field is the missing data value specified here. Character fields are filled with the first character of this value. If no missing data value is provided then the value of the first default category (if any) is used. If there is no default category for the field, then zeros are placed in numeric fields and blanks are placed in character fields. A reply is not required for this question.

Recode Type For character fields up to 24 characters long, as well as numeric fields, certain data values or ranges of values may be changed to some other value when data are entered. For each field, a logical relation specifying when the value should be changed, called a *recode type*, and a new value, may be supplied. A field may also be changed to a specified value for all records. An index field may provide increasing key values. Simple arithmetic operations may be performed on the data in a field.

After prompting for missing data, Micro.Form prompts for '`recode type`'. A null reply cancels recode entry and moves to the next field in the dataset. Otherwise, one or more recode operations may be entered for a field.

The eight recode types and other information that must be supplied with them are shown below. The information for a recode may be placed on one line with each item separated by blanks. If not entered on one line it is prompted for.

Recode Types

Recode	Associated Values
LT	*oldvalue newvalue*
LE	*oldvalue newvalue*
EQ	*oldvalue newvalue*
GE	*oldvalue newvalue*
GT	*oldvalue newvalue*
BTW	*minimum maximum newvalue*
IX	*initialvalue increment*
ALL	*newvalue*

For recodes `LT` through `GT`, if the value read has the specified relationship to *oldvalue*, the value assigned to the field is *newvalue*. I.e., if the *oldvalue* is 4, and *newvalue* is 6, recode `LT` assigns the field 6 if the value read is strictly less than 4, else it assigns the value read.

For recode `BTW`, if the value read is greater than or equal to the *minimum* and less than or equal to the *maximum*, the field is assigned the *newvalue*.

For recode `IX`, Micro.Cnvrt places the *initialvalue* into the field in the first record, adds *increment* to produce a value for insertion in the second record, etc. Each record thus contains a Key field, a uniquely identifying integer. The default values for both *initial value* and *increment* are 1, which are assigned if a null return is entered in response to a prompt.

For record `ALL`, the field in all records of the dataset is assigned the *newvalue*.

For character values, collating order of characters is applied in evaluating recodes, except that `IX` is not permitted. Collating order has punctuation first, followed by the lowercase alphabet, the uppercase alphabet, and digits 0-9. A table of characters in collating order is in Appendix A. For example, recode `GE`, with *oldvalue* '`E`' and *newvalue* '`K`', assigns the field '`K`' if the data read begins with '`E`' or greater in the alphabet, else assigns the value read.

For data input types `B`, `I`, `X` and `R`, the *newvalue* may be an arithmetic operator (+,-,*,/) followed by a constant. However, for `R` (real number) the constant may be a real number since floating point arithmetic is performed. The resulting number is then rounded to the nearest integer value. For example, for integer input, the following recode sequence adds 3 to all data values greater than 100.

```
RECODE TYPE?gt 100 + 3
```

More than one recode for a field may be given, though only one may be given per line. If a recode operation conflicts with one given previously, the program prints an error message and ignores it.

Conflicts occur when more than one recode is given for the same data value.

Command Mode

When format information for all fields has been entered, Micro.Form enters command mode, from which the format can be examined and edited.

Micro.Form Commands

Command	Example	Effect
CLose	`close`	stores the current conversion format information in the cf file
DElete	`del idnum 2`	deletes the second recode for field Idnum; omitting the number would delete all recodes; field format information may not be deleted, but can be changed with the Redo command
Document	`doc`	prints format and recode information for the entire dataset; a single field or single set of recode information for a field may also be documented
RECode	`rec idnum`	prompts for recode information for field Idnum
Redo	`redo idnum`	prompts for format for field Idnum; if field name is omitted, prompts for general information (card size, number of cards in record, etc.)
STop	`stop`	writes the conversion format information to the cf file and ends the session

ILIR:Micro.Cnvrt

The conversion format file created with ILIR:Micro.Form is used to convert data to Micro internal format with the program ILIR:Micro.Cnvrt.

`$Run ILIR:Micro.Cnvrt` `input=`*file* `object=`*file* [`print=`*file*]
[`par=dsn=`*dataset* `dir=`*userid* `key=`*scramblekey* `cff=`*file* `time={`*n*`|OFF}`]

The `input` and `object` files are required. The input file contains data according to the format specified in the `cff` file, the conversion format file, which is prompted for if omitted. Error records are written to the `object` file. Specifying '`print=`*file*' writes error information to the file instead of the screen. A dataset can be given with '`dsn=`*dataset*' which suppresses the prompts described below. A time limit is given with '`time=`*n*'; the default is 5 CPU seconds if omitted.

If the dataset and/or conversion format file are omitted, the program prompts for one or more of:

- Micro dataset name;
- userid of another directory (or '`cancel`');
- data file name;
- scramble key; and
- name of conversion format file.

Micro dataset name

This may be either the name of an existing dataset (indicated by the absence of a colon (`:`) in the fifth column of the reply) or the name of an existing Micro dictionary file (indicated by a four-character userid on which the dictionary is stored followed by a colon and the dictionary file name). If no reply is given the program terminates.

Userid of another directory or '`cancel`'

This is requested only if the dataset name cannot be found in the directory that was searched or if that directory does not exist or is not accessible. A four-character signon-ID is all that is required. If '`cancel`' is the reply, a new dataset name is prompted for and user's directory is searched.

Data file name

This is requested only if a dictionary file name was given instead of a Micro dataset name. If the file does not exist, it is created. If it exists then it must be a sequential file permitted RW (read-write access). If it contains data then the data must be compatible with the dictionary. A reply is required.

Scramble key

This is requested only if the dataset was defined to be scrambled. The same key will be used to unscramble the dataset and to scramble the dataset with the added records. The same key is used both to scramble and to unscramble a dataset. The user must remember the key because it is not stored permanently by Micro. Garbled data indicates that the wrong key was used to unscramble the dataset. A reply is required.

Name of conversion format file

This is requested if the dataset was initially defined without a conversion format file or if a dictionary file name was given instead of a Micro dataset name. The file must contain format information for each field found in the dictionary.

The program reads data from the input file and converts it using the conversion format. Micro.Cnvrt enforces the "data required" condition for each field. The converted data is recoded as necessary and checked to see that the final value does not exceed the size of the field. The value is checked to see if it is a category value if categories are required for the field.

When data for a field is invalid Micro.Cnvrt prints a message on `print` (which defaults to the terminal) specifying the invalid value and the field. Processing of the record continues with the next field so that all erroneous values are identified. Then the record is written on `object`, one card per line.

Records converted for Micro are written at the end of the sequential file -RESTDS (created automatically if necessary). When there is no more data to be converted Micro.Cnvrt rereads the converted data from -RESTDS, sorts and weeds these records, combines them with any existing data in the file and then writes out the final dataset.

After completing the dataset, Micro.Cnvrt destroys the file -RESTDS and reports the number of records in the original set, the number of records read in, the number of records added, the number of records rejected, the number which were duplicates and were weeded out, and the number of records in the resulting permanent set.

Custom Programming Method

A user program can write data to a file for entry via one of the above methods. The Micro Read Array command can read a binary "rectangular" file, one line per record, with numeric values in binary. Scaled data must be transformed appropriately, and categorical data converted to valid category values.

Visual Data Entry and Editing

Visual data entry and editing are provided by ILIR:MicEdit. MicEdit allows data to be displayed visually, customized screens to be designed, and existing records modified or new ones added. The program can be run from any terminal which supports the MTS visual editor or the $FSMessage command, including IBM PC family compatibles, Macintoshes, and Ontels. The program is run by:

`$Run ILIR:MicEdit` [`PAR=KEYDEFS=`*file*]

The program prompts for the dataset name, then displays data in visual mode. The default screen shows every field name and every data value in a record. The cursor can be moved from one field to the next with the return key. Changes to the fields are sent to MicEdit when a key corresponding to a Visual Program Function (VPF) is pressed. The VPFs are assigned to different keys on the various types of terminals and microcomputers, and are associated with editing and screen manipulation functions in MicEdit. The table on the following page shows the VPF keys and their MicEdit functions on the common devices.

Visual Program Function Key Definition

The "content" of a VPF key, the MicEdit function it corresponds to, can be redefined in a file and the file specified when MicEdit is run, as shown above. The *file* is an MTS line file containing key definitions in the format

function=*cmd* [,*function*=*cmd*]...

where *function* is one of VPF2 through VPF19, or in IBM PC keyboard notation, one of F2 through F10, A1, A2, PgUp, A4, A5, PgDn, A7, A8, or A9.

The F1 function, the unshifted `F1` key on the IBM keyboard and the VPF1 function, is always `HELP` in MicEdit, and can not be redefined. The functions F2 through F10 are the unshifted keys `F2` through `F10` on the IBM keyboard, and VPF 2 through VPF10. The functions A1 through A10 are the "alternate" `F1` through `F10` keys, i.e., `Alt F1` through `Alt F10`, each pair pressed simultaneously, and VPF 11 through VPF20. PgUp is the `PgUp` key and VPF13, PgDn the `PgDn` key and VPF16.

The *cmd* can be one of entries in the last column of the following table, which are explained in more detail below.

Visual Program Function Key Definitions

Visual Function	IBM PC	Ontel (%pad pfkeys)	Macintosh (numpad)	Command
VPF0	End	numpad -	0	PgDn (next screen)
VPF1	F1	Shift numpad 7	1	Help
VPF2	F2	Shift numpad 8	2	Goto (record #)
VPF3	F3	Shift numpad 9	3	Window Left
VPF4	F4	Shift numpad 4	4	Window Right
VPF5	F5	Shift numpad 5	5	Top (first record)
VPF6	F6	Shift numpad 6	6	Bottom (last record)
VPF7	F7	Shift numpad 1	7	PgUp (previous screen)
VPF8	F8	Shift numpad 2	8	Next (record)
VPF9	F9	Shift numpad 3	9	Save (changes)
VPF10	F10	Shift numpad 0	Clear	Stop
VPF11	Alt F1	Shift numpad .	= or -	Enter (new record)
VPF12	Alt F2	Shift numpad +	/ or +	Delete (record)
VPF13	PgUp	numpad 7	*	PgUp (previous screen)
VPF14	Alt F4	numpad 8	- or /	Undelete (record)
VPF15	Alt F5	numpad 9	+ or ,	Ooops (exit, no save)
VPF16	PgDn	numpad 4	Enter	Next (screen)
VPF17	Alt F7	numpad 5	.	Undo
VPF18	Alt F8	numpad 6		Options
VPF19	Alt F9	numpad 1		Add Field
VPF20	Alt F10	numpad 2		future use
VPF21	Shift F1	Shift PF5		Scan
VPF22	Shift F2	Shift PF6		Scan again
VPF23-35				undefined

HELP The Help command displays the current definition of the function keys if they have not been redefined globally. It also will offer online documentation of the program. Help is always available as VPF1 (F1) but can be defined as another function as well.

GOTO The Goto command prompts you for a record number. Allowable record numbers range from 1 to the number of records in the dataset. Entering a number N will display the Nth record in the dataset.

WL The WL command windows or scrolls left one screen.

WR The WR command windows or scrolls right one screen.

TOP The Top command displays the first record in the dataset.

BOTTOM The Bottom command displays the last record in the dataset.

`NEXT`	The Next command displays the record following the record currently displayed. If you are displaying the last record in the dataset, this function adds a new record.
`PREVIOUS`	The Previous command displays the record previous to the record currently displayed, or the last record if the first is the current.
`SAVE`	The Save command makes the changes, additions and deletions made to the dataset so far permanent.
`STOP`	The Stop command saves the changes made to the dataset and returns to MTS.
`ENTER`	The Enter command creates a new record and places default values in the fields. The new record will be appended to the end of the dataset if any modification is made.
`DELETE`	The Delete command marks the record displayed for deletion. The record will be deleted when you Stop or Save the dataset.
`PGDN`	The PgDn command displays the next page of data if the record needs multiple pages (screens) to display all the fields. If the last page of the record is displayed, PgDn will display the first page of the next record.
`UNDELETE`	The Undelete command removes the deletion mark placed on a record by the Delete command. The record will remain in the dataset when you Stop or Save the dataset.
`OOPS`	The Oops command exits the program without making any permanent changes. All changes since the last Save or Stop will be lost.
`PGUP`	The PgUp command displays the previous page of data if the record needs multiple pages (screens) to display all the fields. If the first page of the record is displayed, PgUp will display the last page of the previous record.
`UNDO`	The Undo command forgets all the changes you made to the last record altered.
`OPTIONS`	The Options command describes the type of data you may enter into the field that the cursor was on when this function was invoked. It presents a category list to choose from for categorical fields.

SCAN	The Scan command searches records for field values. The command prompts for a scan phrase, then examines each record, starting with the one on the screen, to see if it matches all of the criteria set in the scan phrase. The first record examined that matches all the criteria will be displayed. The scan phrase has the following syntax:

field{=\|==}*value* [*field*{=\|==}*value*]...

where *field* is a field name or abbreviation and *value* is a category name, abbreviation, value, character string or pattern. Character strings or patterns with embedded blanks must be enclosed in primes or quotes. Each field is separated from its value by either '=' or '=='. The '==' performs case sensitive matching in character fields. The '=' is not case sensitive. Patterns (such as 'A?') can be used to match Type E (external) or type C (fixed length) data. For example, entering a scan phrase

```
remark=="A?"
```

causes MicEdit to display the next record that had a remark whose first letter was an upper case A. You may have up to five scan criteria.

SCANAGAIN	The ScanAgain command continues the previous scan starting with the next record. The Help command will display the active scan.
ADDFIELD	The Addfield command adds up to five fields to the dataset, starting after a given field, which is initially the field the cursor was on when the Addfield key was pressed. A field entry screen is displayed, with five "new field templates," each containing empty fields for the field name, abbreviation, type, size, etc. A field at the top denotes the field in the dataset after which the new fields are to be inserted. If new fields are needed at different locations in the dataset, the field entry screen must be exited after the first set of fields is entered, and re-entered from the main screen.

Following is an example of a file which customizes the VPF key assignments for an IBM PC compatible keyboard.

```
F2=Options,
F3=WL,          F4=WR,
F5=Previous,    F6=Next,
F7=Top,         F8=Bottom,
F9=GoTo,        F10=Enter,
A1=Save,        A2=Stop,
A4=MTS,
A5=Oops,
A7=Undo,        A8=AddField,
A9=Undelete,    A10=Delete
```

Screen Definition

The default screen layout shows every field name and every data value in a record. It attempts to display all information on one page (screen). Text can be added to the screen, fields excluded, and the format of displayed fields changed by creating a customized screen file, which is specified as a parameter on the $Run command:

```
$Run ILIR:MicEdit PAR=SCREEN=file
```

where *file* is the name of a MTS line file which describes the locations and formats of the data displayed on the screen. The screen file may optionally contain the name of the dataset. Specify

```
DSN=dataset
```

in the first column of the the first line of the screen file and the prompt for dataset name is suppressed.

A MicEdit data entry screen need not correspond to the physical screen, but can be longer or wider, the VPF keys being used to window left or right, or up or down. A screen is made up of cells. The type of information the cells contain, their location, width, and format is described in the screen file.

Cells contain two types of information, text and data. In the screen file, text must be placed in quotes. It is displayed on the screen exactly as it is between the quotation marks. Data is information retrieved from a field in the dataset. Data can be changed on the screen. A field is placed on the screen by entering its name or abbreviation in the screen file.

Often a field name in quotes is used to identify the data field that follows it. The line

```
"Student Identification Number"@row=3@col=10 SIN@lj
```

causes MicEdit to display the quoted text on row 3 starting in column 10. The data (from the field whose abbreviation is '`SIN`') is displayed left justified ('`@lj`'), in row 3 in the first available column after the text (column 40, or column 10 plus the length of the text).

'`@row=x`', '`@col=y`' and '`@lj`' are cell modifiers. Modifiers apply to the text or data immediately preceding them on the same line of the screen definition file. A new line in the file starts a new row on the screen, unless the first cell on the line has a `row` modifier. Subsequent cells on that line appear on that row. Other modifiers control the column position, and the format of the fields being displayed. Both text and data display may be affected with these modifiers:

`@row=`*i*	Specifies the row number *i*, where *i* must be at least 1 but can exceed the number of rows on the physical screen.
`@col=`*j*	Specifies the beginning column number *j*, where *j* must be at least 0 but can exceed the width of the physical screen.

`@width=n`	Specifies the number of columns in the cell.
`@center`	Centers the data within the cell.
`@rj`	Right justifies the data within the cell.
`@lj`	Left justifies the data within the cell.
`@rv`	Displays field in reverse video (default for data)
`@prot`	Protects field from overwriting (default for quoted text).
`@$`	Includes a dollar sign in the data cell.
`@abbr`	Displays data as category abbreviations.
`@value`	Displays data as category values.
`@date`	Prints date data in mm-dd-yy format.
`@ssn`	Prints data in social security number format, xxx-xx-xxxx.
`@zip`	Displays data with leading zeros.

Keep in mind when constructing a screen file that cells cannot overlap on the screen and they must have at least one space between them. Cells defined to exceed the width of the physical screen will be wrapped around when possible. You may scroll right or page down to see cells defined to start in columns or rows wider or longer than the physical screen. These cells are stored on separate pages. Multiple page screens are not significantly more expensive to use but accessing them may take more time. Following is a screen file for the Fin-Aid dataset. Blank lines appear on the screen as blank lines. Lines beginning with '*' are explanatory comments, and do not appear on the screen.

```
DSN=FIN-AID
* Data Entry Screen for Fin-Aid Dataset
"Financial Aid Data Entry Screen"@width=80@center

"Student number: " sin@prot@ssn class-level
```

```
"Term (Fall,Spring,Summer)"@col=30 "Year"@col=56
term@col=36 year@col=56

"awarded"@col=22 amount@col=30@$@rj type@lj "from" sof

"=============================================================================="
rem*
```

Other Parameters

ILIR:MicEdit allows the dataset to be edited to be specified on the $Run command:

`$Run ILIR:MICEDIT PAR=DSN=`*dataset*

This suppresses the prompt for dataset name. Specifying the dataset name on the $Run command takes precedence over a dataset name identified in the screen file.

Datasets on a different account can be edited by specifying the directory file.

`$Run ILIR:MICEDIT PAR=DIR=`*userid:directory_file*

MTS

Documentation on MTS (Michigan Terminal System) ranges from a pocket-sized MTS Reference Summary, to single-topic QuickNotes, to User Guide tutorials, to CCmemos, to a multi-volume series of reference manuals. The MTS file *ITDBIBLIOGRAPHY is a comprehensive list of all the MTS and Merit Computer Network documentation, much of which is available through the program *DOCINFO, for printing at the high-speed Xerox 9790 laser printers on campus. These printers print on 8.5 × 11" paper. MTS and Merit publications are also available in campus bookstores. *ITDBIBLIOGRAPHY also describes MTS and Merit consulting services. Prospective users of Micro can contact the Information Technology Division Accounts Office to obtain an MTS account.

ITD Accounts Office
Argus Building
535 West William Street
University of Michigan
Ann Arbor, MI 48109–4943

313-764-3518

It is possible to obtain output from the Xerox 9790 laser printers through US Mail or UPS. The keyword '`DELIVERY=MAIL`' must be specified, as in

```
$Control *print* delivery=mail priority=minimum
```

which provides a receipt number. Then a message must be sent to the MTS Mail Librarian—`$mess send to 'mail.librarian'`—giving the receipt number for the job, the shipping address, the shipping service (US mail or UPS) and the MTS account to which the shipping cost can be billed. That account must be an external or sponsored research account.

It is also possible to submit and retrieve magnetic tapes by mail. The program *TAPESUBMIT must first be run; entering '`submit`' elicits a series of prompts about the tape, and produces eventually a tape id and receipt number. These should be enclosed with the tape, and mailed to MTS Mail Librarian (rather than ITD Accounts Office) at the above William Street address. A tape can be retrieved by sending a message to the Mail Librarian with the tape id and receipt number. Volume 19 in the MTS series documents magnetic tape handling facilities, a subject usually of interest to Micro users.

Part II

Micro Tutorial

Introduction

This part contains a series of tutorials, starting with the very basics of Micro and proceeding to advanced topics. If you are new to Micro, this is a painless way to learn the program. If you have used Micro before, you can skim the tutorials to refresh your memory. If you are looking for specific examples of how to do something, you can use the index or the table of contents to find the particular operation or concept that is of interest to you.

The tutorials have a simple format: there are examples of Micro operations printed in `typewriter` style surrounded by text explaining what's going on. The examples come directly from logs of actual Micro sessions. If you try the examples yourself, you should see exactly the same thing on your screen or terminal.

All of the tutorials were run on the UM MTS host after signing on to the K59F userid.

Command Language I

Running Micro

This is the first Micro Tutorial. We will introduce the basic Micro concepts and terminology while using the elementary Micro commands. To start Micro, use the $Run command at the MTS pound sign (#) prompt.

```
#$Run ILIR:Micro
#Execution begins   14:47:08
 ILIR:Micro (15 Dec 91)
  University of Michigan
 Thu 5 Mar 1992  14:47:09

 Ready:
-
```

When Micro is ready to go it prints '`Ready:`', prompts with a hyphen (-) on the next line, and then waits for you to enter a command. After typing a command, you must press the Return key (sometimes called the Enter key) to send it to Micro. Unless otherwise noted, Micro is not case-sensitive: you may type letters in either upper or lower case.

Some of the commands in this tutorial generate output that is wider than a normal terminal or screen. The next command changes the Micro output length from the default value of 79 characters to 132 characters. There will be more about the Settings command later.

```
 Ready:
-Settings  Command: OutLen = 132
```

Help Command

If you are new to Micro, you may want some help. The Help command presents a menu-based help system that you can use to explore any topic in Micro. By choosing '`1`' from the first menu, we get another menu listing all of the Micro commands.

```
 Ready:
-Help
```

```
Hopefully, at least one of these topics will be of service ...
-*-
 1. Micro command list          4. Learning the ropes
 2. Micro command information   5. Having problems
 3. Information, please         6. Using the Micro Help system
-*-
Choose a Topic (or just Return to exit menu): 1

Micro command list

This menu contains a list of Micro commands.  For information about a
command, select it from the menu.
-*-
  1. Calculate command         16. FullDoc command  31. Remove command
  2. Call command              17. Get command      32. Replace command
  3. Change command            18. Help command     33. Resettings command
  4. Combine command           19. Join command     34. Restrict command
  5. Comment command           20. Key command      35. RestrictAndJoin command
  6. Compute command           21. List command     36. Save command
  7. Crosstabulate command     22. Lock command     37. Select command
  8. Datestamp command         23. Message command  38. Settings command
  9. Describe command          24. MICall command   39. Sort command
 10. Destroy command           25. MTS command      40. Stop command
 11. Display command           26. Name command     41. TechDoc command
 12. Document command          27. Print command    42. Update command
 13. Enterdata command         28. Purge command    43. Use command
 14. ExtCall command           29. Read command     44. Write command
 15. Find command              30. Release command
-*-
Choose a Topic (or just Return to exit menu):
-*-
 1. Micro command list          4. Learning the ropes
 2. Micro command information   5. Having problems
 3. Information, please         6. Using the Micro Help system
-*-
Choose a Topic (or just Return to exit menu):
```

You can get help for any command by choosing it from the menu (try it yourself) but for now a null reply (just pressing Return) gets us back to the first menu and another null reply gets us out.

As you can see, Micro has many commands. But fear not, we will cover them one at a time.

Making Datasets Available

Most of the Micro commands operate on tables called *datasets* (often referred to as just *sets*). Datasets "contain" the data and, collectively, they make up a Micro *database*. To list the datasets in the current Micro database, use the List command.

```
Ready:
-List datasets
 You have 0 datasets.
```

There are no datasets since a Micro database has not been built on this MTS userid. For this tutorial we will use some existing datasets from a database on the ILIR userid.

```
Ready:
-Get ILIR
  8 datasets from ILIR:Directory
```

```
 Ready:
-List datasets
 You have 8 datasets.

 Dataset name     Status Owner
  Students         Perm   ILIR
  Fin-Aid          Perm   ILIR
  Cars             Perm   ILIR
  ART              Perm   ILIR
  Courses          Perm   ILIR
  CPS.Pers         Perm   ILIR
  CPS.Family       Perm   ILIR
  CPS.house        Perm   ILIR
```

A dataset looks like a table of information with columns called *fields* and rows called *records.* There can be any number of datasets in a database. Each dataset has a *dataset name* with which to identify it in Micro.

Micro stores a dataset in MTS files. You can do a significant amount of work in Micro without ever dealing with files directly. But for those who are interested, a dataset is stored in:

- a *dictionary file* containing the format or structure (fields) of the dataset;
- a *Micro data file* containing the data (records) in the form specified by the dictionary; and
- an entry in the *directory file* connecting these files with a dataset name.

When building your own database, you create datasets using the interactive dataset definition program, ILIR:MicDef, and then enter records into the datasets using any of the Micro data entry facilities. All of these programs will be discussed in future tutorials. For now we will use two existing datasets, Students and Fin-Aid, from the ILIR database.

Print Command: Printing the Contents of a Dataset

To print the contents of a dataset, use the Print command. The normal print layout reflects the view of a dataset as a table with rows and columns. Note: we use '@count=5' here to print only the first 5 records. More about this later.

```
 Ready:
-Print@count=5 Fin-Aid

 Student-ID-Num Year-Awarded Term-Awarded Class-Level Type-of-Aid Source-of-Funds       Amount  Remark*
-----------------------------------------------------------------------------------------------------------------
      113591707           74         Fall      Senior     Tuition      University 1200.00
      123456789           75         Fall   Sophomore     Tuition      University  235.00
      133680736           74         Fall      Junior  Room-Board     Federal-Gov  485.00  Outstanding senior
      142668782           74         Fall      Junior     Tuition        Industry 1050.00  Award from
                                                                                            International
                                                                                            Business Systems of
                                                                                            Armonk, New York
      149514226           75       Winter      Junior        Cash        Industry  100.00  Grant from the Oily
                                                                                            Petroleum Co. to
                                                                                            study oil slicks
```

To print only the number of records in a dataset, use the 'count' keyword in the Print command.

```
 Ready:
-Print in Fin-Aid count
  34 records in Fin-Aid
```

If your database requires security beyond restricted file access, you can *scramble* records as they are stored in a dataset. Micro asks you for a *scramble key* (just a string of characters) that is used to scramble the records. You must use exactly the same scramble key to unscramble the records when you access them (upper or lower case is important here!). The Students dataset is scrambled so Micro asks for the scramble key when we first use Students (the scramble key is 'TEST').

```
 Ready:
-Print in Students count
 The data for Students must be unscrambled;
     enter the unscramble key: TEST
  85 records in Students
```

Correcting Mistyped Commands

Now we print the first few records from Students. Notice what happens when we misspell a dataset name.

```
 Ready:
-Print@count=5 Stuents
 "Stuents" is not a dataset.
 Enter another dataset name to replace "Stuents"
     (or enter HELP, LIST [pattern], CANCEL): help

 Micro can't use the given dataset.  In order to continue the current
 operation, Micro must ask for the name of a replacement.  You have the
 following options whenever Micro asks for a replacement dataset:

    *  Enter a replacement to continue the operation.
    *  Enter a null reply (if possible) to continue the operation.
    *  Enter "CANCEL" to cancel the operation.
    *  Enter "LIST" or "LIST pattern" to see a menu of possible choices.

 More information:
 -*-
  1. Datasets        3. Entering a replacement  5. Entering "CANCEL"
  2. Dataset Names   4. Entering a null reply   6. Entering "LIST"
 -*-
 Choose a Topic (or just Return to exit menu):
 Enter another dataset name to replace "Stuents"
     (or enter HELP, LIST [pattern], CANCEL): list
 You have 8 datasets.
 -*-
  1. Students  3. Cars  5. Courses   7. CPS.Family
  2. Fin-Aid   4. ART   6. CPS.Pers  8. CPS.house
 -*-
 Choose a Dataset (or just Return to exit menu): 1

  Student-ID-Num  Name                        Ethnicity  Sex        Birthdate  Class-Year
 ------------------------------------------------------------------------------------------
        13222340  Richardson Deborah              Black  Female        550310          77
        95887328  Pond Stuart Leslie              White  Male          560528          78
       113591707  Jones William                   White  Male          551010          77
       122045525  Grace Kenneth R                 White  Male          580418          80
       133680736  Lee Sue Jean                 Oriental  Female        560716          78
```

Note that there are different types of fields. Student-ID-Num is a *numeric* field: it contains only numbers. Name is a *character* field: it can contain letters, numbers, and special characters. Ethnicity looks like a character field, but it is actually numeric: it has *categories* that refer to the underlying numbers.

To see the field and category information for a dataset, use the Document command.

Document Command: Documenting Dataset Structure

```
 Ready:
-Document Students

 Students          Example data set -- Demographic information on students
                   09-11-81/ILIR

         Scrambled     Not Destroyable     Not Replaceable

   Dictionary file = ILIR:STUDENTS#     Micro Data file = ILIR:STUDENTS

 -----------------------------------------------------------------------------
   F#  Field name       Abbr       Value  Description
 -----------------------------------------------------------------------------

   F1  Student-ID-Num   SIN      DataReq  Student Identification Number
   F2  Name             Na       DataReq  Name of Student (Last First Middle
                                           format)
   F3  Ethnicity        Race     DataReq  Ethnicity
              7 categories (required)
          Unknown       Unk            0  Unknown
          White         Wh             1  White
          Black         Bl             2  Black
          Amer-Indian   AI             3  American Indian
          Spanish       Sp             4  Spanish-Surnamed American
          Oriental      Or             5  Oriental
          Other         Oth            6  Other
   F4  Sex              Sex      DataReq  Sex
              3 categories (required)
          Male          Ma          M     Male
          Female        Fe          F     Female
          Unknown       Unk         U     Unknown
   F5  Birthdate        Bday     DataReq  Birthdate (YYMMDD format)
              1 category
          Missing       Miss           0  Missing
   F6  Class-Year       ClYr              Class Year (YY format)
              1 category
          Unknown       Unk           -1  Unknown
```

Micro dataset, field, and category names and abbreviations can't contain embedded blanks or any of the following special characters:

```
; , :  '  " & | = < > !  ?  @ ~ ( )
```

In addition, Micro names can't be longer than the maximums shown in the following table.

Maximum Lengths for Micro Names

	Dataset	Field	Category
Name	16 characters	16 characters	11 characters
Abbreviation		4 characters	4 characters

Now we document the Fin-aid dataset.

```
Ready:
-Document Fin-Aid

Fin-Aid           Example data set -- Financial aid information
                  09-11-81/ILIR

        Not Scrambled     Not Destroyable     Not Replaceable

  Dictionary file = ILIR:FIN-AID#      Micro Data file = ILIR:FIN-AID

----------------------------------------------------------------------------
  F#  Field name       Abbr        Value  Description
----------------------------------------------------------------------------

  F1  Student-ID-Num   SIN       DataReq  Student Identification Number
  F2  Year-Awarded     Year      DataReq  Year Awarded (YY format)
  F3  Term-Awarded     Term      DataReq  Term Awarded
            3 categories (required)
        Fall           F               1  Fall Term
        Winter         W               2  Winter Term
        Summer         S               3  Summer Term
  F4  Class-Level      CL        DataReq  Class Level
            4 categories (required)
        Freshman       Fr              1  Freshman
        Sophomore      So              2  Sophomore
        Junior         Jr              3  Junior
        Senior         Sr              4  Senior
  F5  Type-of-Aid      Type      DataReq  Type of Aid
            6 categories (required)
        Tuition        Tu              1  Tuition
        Room-Board     RB              2  Room and Board
        Cash           Ca              3  Cash
        Loan           Lo              4  Loan
        Work-Study     WS              5  Work Study Program
        Other          Oth             6  Other
  F6  Source-of-Funds  SOF       DataReq  Source of Funds
            6 categories (required)
        University     Univ            1  University
        State-Gov      SGov            2  State Government
        Federal-Gov    FGov            3  Federal Government
        Industry       Ind             4  Industry
        Foundation     Foun            5  Foundation
        Other          Oth             6  Other
  F7  Amount           Amt       DataReq  Amount of Award
  F8  Remark*          Rem*               Additional Comments
          External file = ILIR:REMARKS*
```

Find Command: Data Retrieval and the Result Dataset

Data retrieval is done in Micro using the Find command. It retrieves records that satisfy the criteria specified in the command.

```
Ready:
-Find in Fin-Aid where Type-of-Aid is Loan
 4 records found (11.765%)
 4 records in RESULT
```

The Find command creates a new dataset, the *Result dataset* (usually just called *Result*). The Result always has the name '`Result`'; '`It`' can be used as a synonym. The Result of this Find command contains only those records satisfying the criteria. To print the Result, simply ...

```
Ready:
-Print Result

 Student-ID-Num Year-Awarded Term-Awarded Class-Level Type-of-Aid Source-of-Funds       Amount  Remark*
------------------------------------------------------------------------------------------------------------
      408823168           76         Fall   Sophomore        Loan      University       225.00
      498930942           73         Fall      Senior        Loan       State-Gov       820.00
      805219718           77       Winter    Freshman        Loan      University       150.00
     1399094799           73         Fall      Senior        Loan     Federal-Gov       180.00
```

Permanent and Temporary Datasets

Let's list the datasets again to see how the Result fits into our database. If you just type '`List`' (omitting the '`datasets`' keyword), Micro assumes that you want a list of datasets.

```
Ready:
-List
You have 9 datasets.

Dataset name     Status Owner
 RESULT           Temp*  K59F
 Students         Perm*  ILIR
 Fin-Aid          Perm*  ILIR
 Cars             Perm   ILIR
 ART              Perm   ILIR
 Courses          Perm   ILIR
 CPS.Pers         Perm   ILIR
 CPS.Family       Perm   ILIR
 CPS.house        Perm   ILIR
```

Students and Fin-Aid are listed with `Perm*` status and `ILIR` as the owner. '`Perm`' means that they are *permanent datasets*: they will remain when we exit Micro. The Result is listed with `Temp*` status and the current userid as the owner. '`Temp`' means that the Result is a *temporary dataset*: it is a full-fledged dataset that can be used in any Micro command, but it will be lost when we exit Micro. You will see later how to make a temporary set permanent. The '`*`' indicates that a dataset is *active*. The Result is active because it is a temporary set (temporary sets are always active).

Students and Fin-Aid are active because they have been used in previous commands. The other permanent datasets have not been used so they are not active.

The next Find command creates a new Result dataset which replaces the old one. Only one dataset named Result can exist at one time.

```
 Ready:
-Find in Fin-Aid where Class-Level = Freshman
  6 records found (17.647%)
  6 records in RESULT
```

Name Command: Retaining the Result Dataset

To keep a Result around for use later in the same Micro session, change its name to something other than 'Result' or 'It' using the Name command. The dataset will still be temporary, but it won't be replaced when a new Result is created.

```
 Ready:
-Name It Freshmen

 Ready:
-List
 You have 9 datasets.

 Dataset name     Status Owner
  Freshmen         Temp*  K59F
  Students         Perm*  ILIR
  Fin-Aid          Perm*  ILIR
  Cars             Perm   ILIR
  ART              Perm   ILIR
  Courses          Perm   ILIR
  CPS.Pers         Perm   ILIR
  CPS.Family       Perm   ILIR
  CPS.house        Perm   ILIR
```

Abbreviations and Synonyms

To make typing easier, you can use short forms for command names, keywords, fields, and categories. In the next command '`F`' is a *minimum abbreviation* for the Find command ('`Fi`' and '`Fin`' will work, too); '`w`' is the *synonym* for the `where` keyword ('`wh`' will not work); '`Race`' is the abbreviation for the field Ethnicity; and '`Oth`' is the abbreviation for the category Other. Dataset names do not have short forms.

```
 Ready:
-F in Students w Race = Oth
  2 records found (2.353%)
  2 records in RESULT
```

Print Command: Horizontal and Vertical Layouts

If you do not specify a dataset in the Print command, Micro assumes that you want to print the Result.

```
 Ready:
-P

 Student-ID-Num  Name                             Ethnicity  Sex          Birthdate  Class-Year
-----------------------------------------------------------------------------------------------
      906922815  Foster Harold Peter                  Other  Male            561130          78
      921293985  Arrillaga Paul                       Other  Male            560225          78
```

Unless told to do otherwise, the Print command uses the *horizontal layout*: the field names run horizontally across the page and the records form rows. You can change this by applying a *command modifier* to the Print command. The `@Vertical` modifier forces Micro to use the *vertical layout*: the field names run vertically down the left side of the page with the value for each field on the same line. There are many command modifiers for the Print command. A number of them will be introduced during the course of the tutorials.

```
 Ready:
-P@vertical

 Student-ID-Num  906922815
 Name            Foster Harold Peter
 Ethnicity       Other
 Sex             Male
 Birthdate       561130
 Class-Year      78

 Student-ID-Num  921293985
 Name            Arrillaga Paul
 Ethnicity       Other
 Sex             Male
 Birthdate       560225
 Class-Year      78
```

By default, Micro prints field and category names. This can be changed using the `@ABbr` command modifier.

```
 Ready:
-P@abbr

        SIN  Na                             Race  Sex      Bday ClYr
--------------------------------------------------------------------
  906922815  Foster Harold Peter            Oth  Ma      561130   78
  921293985  Arrillaga Paul                 Oth  Ma      560225   78
```

Find Command: The AND Connective

Suppose we want to find the records for all the women in Students who were born in 1955. To do this we must look at both the Sex and Birthdate fields. First, find the females ...

```
 Ready:
-F in Students w Sex = Female
  45 records found (52.941%)
  45 records in RESULT
```

... then search the Result set for those born in 1955. Birthdate is stored as a number in YYMMDD form.

```
 Ready:
-Find in It where Birthdate is from 550101 to 551231
  11 records found (24.444%)
  11 records in RESULT
```

Of course, we can obtain the same result in one command using and.

```
 Ready:
-F in Students w Sex = Female and Birthdate is from 550101 to 551231
  11 records found (12.941%)
  11 records in RESULT
```

Print Command: Specifying Fields

If you don't want to print all of the fields in a dataset, you can specify the fields you do want to print in the Print command. You can also modify the format of a field by attaching a *field modifier* to its name. In the example below, we use the @SSn and @Date field modifiers to obtain a more readable printout.

```
 Ready:
-Print in Result SIN@ssn, Name, Sex, and Birthdate@date

  Student-ID-Num Name                     Sex          Birthdate
-----------------------------------------------------------------
     013-22-2340 Richardson Deborah       Female        03-10-55
     251-76-0359 Jones Dorothy            Female        03-14-55
     463-89-8163 Borchinsky Ethel         Female        01-23-55
     498-83-0942 Newar Evelyn             Female        05-13-55
     508-99-2890 Curtis Elizabeth Ann     Female        10-15-55
     514-31-2603 Decker Gertrude Mary     Female        07-06-55
     540-13-4327 Segnalla Lynn Marie      Female        06-20-55
     686-85-0331 Bruza Leslie             Female        05-11-55
     733-26-7655 Smothers Mae S           Female        08-20-55
     796-20-2403 Rogers Lucille A         Female        07-12-55
     806-74-6481 Hardy Coleen             Female        01-02-55
```

Sometimes it's easier to specify the fields you don't want to print. However, you can't use field modifiers.

```
 Ready:
-P in It all but Clyr and Race

  Student-ID-Num Name                     Sex          Birthdate
-----------------------------------------------------------------
        13222340 Richardson Deborah       Female          550310
       251760359 Jones Dorothy            Female          550314
       463898163 Borchinsky Ethel         Female          550123
       498830942 Newar Evelyn             Female          550513
       508992890 Curtis Elizabeth Ann     Female          551015
       514312603 Decker Gertrude Mary     Female          550706
       540134327 Segnalla Lynn Marie      Female          550620
       686850331 Bruza Leslie             Female          550511
       733267655 Smothers Mae S           Female          550820
       796202403 Rogers Lucille A         Female          550712
       806746481 Hardy Coleen             Female          550102
```

Find Command: The OR Connective

Suppose we want to find the records of students who are either American Indian, Oriental, or Other. We can do this with one command.

```
 Ready:
-F in Students w Race is Amer-Indian or Race is Oriental or Race is Other
  8 records found (9.412%)
  8 records in RESULT
```

If you are searching for more than one value in the same field, you can just specify the values separated by or's; Micro assumes you are using the same field. This comes in handy when you have many possible values. The next command is equivalent to the previous one.

```
 Ready:
-F in Students w Race is Amer-Indian or Oriental or Other
  8 records found (9.412%)
  8 records in RESULT

 Ready:
-P in It Name and Race

  Name                         Ethnicity
  --------------------------------------
  Lee Sue Jean                  Oriental
  Wong David Lee                Oriental
  Crowfoot William           Amer-Indian
  Wu Hua Chung                  Oriental
  Erickson Sheila Mary       Amer-Indian
  Redpath John               Amer-Indian
  Foster Harold Peter              Other
  Arrillaga Paul                   Other
```

Notice that the Print command does not do any sorting. The records in the Result are still sorted by Student-ID-Num.

Change Command: Changing Data

You will probably need to change the information in your database now and then. The Change command creates a new Result set containing the data from the original set, but with the specified changes applied. The original dataset is *not* changed.

```
 Ready:
-Change in Students w Sex is Unk such that Sex = Female
  1 record changed (1.176%)
  85 records in RESULT
```

If you are changing the same field used in the search, you can use a short form ...

```
 Ready:
-Change in Students w Sex is Unk to Female
  1 record changed (1.176%)
  85 records in RESULT
```

Here we rename the Result so we can use it later in this tutorial.

```
 Ready:
-Name It Fixed
```

Listing the datasets again reveals the temporary datasets created during this session.

```
 Ready:
-List Sets
 You have 10 datasets.

 Dataset name       Status Owner
  Fixed              Temp*  K59F
  Freshmen           Temp*  K59F
  Students           Perm*  ILIR
  Fin-Aid            Perm*  ILIR
  Cars               Perm   ILIR
  ART                Perm   ILIR
  Courses            Perm   ILIR
  CPS.Pers           Perm   ILIR
  CPS.Family         Perm   ILIR
  CPS.house          Perm   ILIR
```

Release Command: Releasing Accumulated Datasets

Temporary datasets can accumulate quickly. This is fine but it does use storage space. If a dataset is no longer needed in the session, it's usually a good idea to release it using the Release command.

```
 Ready:
-Release Freshmen
```

Sort Command: Sorting a Dataset

Micro datasets are always *sorted*. The records are stored such that the values for the first field are in ascending order; if more than one record has the same value for the first field, they are sorted by the second field; and so on. The Sort command creates a Result set in which the fields are rearranged as specified in the command. The records are then sorted based on the new order of the fields.

```
 Ready:
-Sort Fixed by Name
  85 records in RESULT
```

This Result set has the Name field first, followed by the remaining fields from Fixed. Therefore, it is sorted in alphabetical order by Name. Here the `@count` command modifier is used to print only the first 10 records of the Result.

```
 Ready:
-P@count=10 in It SIN@ssn, Name

  Student-ID-Num Name
 ----------------------------------------
     199-47-2144 Almondizer Karen S
     499-44-6234 Alterman James
```

```
    391-77-5833  Armstrong Nora M
    701-38-5226  Arnold Keith R
    921-29-3985  Arrillaga Paul
    139-90-4799  Bailey Edith
    360-98-4224  Bates Leslie Ann
    555-18-0626  Benson Shelia Joan
    336-52-2688  Bishop Arthur B
    686-60-3907  Block Cyrus
```

Select Command: Selecting Fields from a Dataset

Often only a few of the fields in a dataset are needed to perform a given task. The Select command is used to create a Result set containing only the fields you need, in the order that they are specified in the command.

```
 Ready:
-Select in Fixed Name and Student-ID-Num
  85 records in RESULT

 Ready:
-P@count=7

  Name                       Student-ID-Num
 ------------------------------------------
  Almondizer Karen S              199472144
  Alterman James                  499446234
  Armstrong Nora M                391775833
  Arnold Keith R                  701385226
  Arrillaga Paul                  921293985
  Bailey Edith                    139904799
  Bates Leslie Ann                360984224
```

Weeding Duplicate Records

Another property of Micro datasets is that they are always *weeded*. Each record is unique: there are no duplicate records. If the Result of a Select command contains duplicate records (the values for all fields are the same) then the duplicate records in the Result are "weeded" out. The Result set contains one record for each unique combination of values for the fields specified in the command. Therefore, the Result of a Select command may have fewer records than the original set. Micro tells you when records are weeded.

```
 Ready:
-Select in Fixed Sex & Race
 Note:  74 duplicate records weeded.
  11 records in RESULT

 Ready:
-P

  Sex           Ethnicity
 -------------------------
  Female            White
  Female            Black
  Female      Amer-Indian
Female         Spanish
Female        Oriental
Male             White
Male             Black
Male       Amer-Indian
Male           Spanish
Male          Oriental
Male             Other
```

Key Command: Generating a Unique Field

Weeding out duplicate records is usually desirable, but there are times when you want to prevent the weeding process and maintain duplicate records. You can do this by first numbering the records using the Key command before using the Select command. The Key command creates a Result set containing all of the fields from the specified dataset preceeded by a new field, Key, containing a unique number for each record. As long as the result of a Select command includes the Key field, the records are unique and no records will be lost to weeding.

```
 Ready:
-Key Fixed
  85 records in RESULT

 Ready:
-P@count=10 It

        Key Student-ID-Num Name                      Ethnicity Sex         Birthdate Class-Year
-----------------------------------------------------------------------------------------------
          1       13222340 Richardson Deborah            Black Female         550310         77
          2       95887328 Pond Stuart Leslie            White Male           560528         78
          3      113591707 Jones William                 White Male           551010         77
          4      122045525 Grace Kenneth R               White Male           580418         80
          5      133680736 Lee Sue Jean               Oriental Female         560716         78
          6      139904799 Bailey Edith                  Black Female         570913         77
          7      142668782 Cook Cynthia Anna             Black Female         560908         78
          8      149514226 Scott Mary Grace              White Female         561230         78
          9      173049236 Crosby Jane Lillian           White Female         570510         79
         10      179410240 Taylor Vera M                 Black Female         501004         78

 Ready:
-Select in It Key, Sex, and Race
  85 records in RESULT

 Ready:
-P@c=10 It

        Key Sex           Ethnicity
-----------------------------------
          1 Female            Black
          2 Male              White
          3 Male              White
          4 Male              White
          5 Female         Oriental
          6 Female            Black
          7 Female            Black
          8 Female            White
          9 Female            White
         10 Female            Black
```

Crosstabulate Command: Aggregating Data

An especially useful feature of Micro is the Crosstabulate command (usually referred to as the Xtab command) which summarizes or aggregates the data from a dataset. The Xtab command creates a new Result set and automatically prints it.

```
Ready:
-Xtab in Fixed Sex, Race
  11 records in RESULT
  85 records represented

 Sex           Ethnicity       Count      Percent
--------------------------------------------------
 Female            White          31        36.47
 Female            Black          11        12.94
 Female      Amer-Indian           1         1.17
 Female          Spanish           2         2.35
 Female         Oriental           1         1.17
 Male              White          27        31.76
 Male              Black           4         4.70
 Male        Amer-Indian           2         2.35
 Male            Spanish           2         2.35
 Male           Oriental           2         2.35
 Male              Other           2         2.35
```

The Xtab command creates two new fields: Count and Percent. Note the similarity of this Result to that of the Select command a page or so back. Micro counts the number of records having the same combination of values for the specified fields and puts those numbers into the Count field. The *records represented* is the sum of the Count field. The Percent field is simply the percentage of the Count field over the number of records represented.

Print Command: Stratified Layout

Another print layout is the *stratified* layout. It is a combination of the vertical and horizontal layouts. The **@st=*n*** command modifier tells Micro that the first *n* fields printed should be in the vertical layout and the remaining fields in the horizontal layout. The Print command does not do any sorting so the order of the fields is important when using the stratified format.

```
Ready:
-Print@st=1 in It Sex, Race, Count, %

Sex              Female

   Ethnicity      Count      Percent
-------------------------------------
       White         31        36.47
       Black         11        12.94
 Amer-Indian          1         1.17
     Spanish          2         2.35
    Oriental          1         1.17
```

```
Sex             Male

   Ethnicity     Count     Percent
--------------------------------------
       White        27       31.76
       Black         4        4.70
 Amer-Indian         2        2.35
     Spanish         2        2.35
    Oriental         2        2.35
       Other         2        2.35
```

Crosstabulate Command: Descriptive Statistics

The Xtab command can be used to calculate descriptive statistics.

```
 Ready:
-X in Fin-Aid Class-Level, ave Amount, tot Amount
  4 records in RESULT
  34 records represented

  Class-Level  Ave.Amount  Tot.Amount      Count     Percent
 ------------------------------------------------------------
     Freshman   263.3333     1580.00          6       17.64
    Sophomore   203.6667     1833.00          9       26.47
       Junior   388.6364     4275.00         11       32.35
       Senior   468.7500     3750.00          8       23.52
```

The Xtab command creates the Ave.Amount and Tot.Amount fields. The next command demonstrates all of the statistical operations as well as how to aggregate over the whole dataset.

```
 Ready:
-Xtab in Fin-Aid tot Amount, ave Amount, std Amount, med Amount, min Amount, max Amount
  1 record in RESULT
  34 records represented

   Tot.Amount  Ave.Amount  Std.Amount  Med.Amount  Min.Amount  Max.Amount      Count     Percent
 ------------------------------------------------------------------------------------------------
     11438.00   336.4118    263.6519     265.00       50.00     1200.00          34      100.00
```

Write Command: Exporting Data

More complex statistical operations should be done by a program specifically written for that purpose, such as MIDAS. The data from any Micro dataset can be prepared for use in MIDAS using the Write for MIDAS command (currently only the MIDAS interface is supported). First, select all fields except Remark* from Fin-Aid (comments are not useful as statistics) ...

```
 Ready:
-Select in Fin-Aid all except Remark*
  34 records in RESULT
```

...then write the Result for MIDAS. You need only specify the name of the MTS file to use (usually a temporary file) and Micro sets it up for you.

```
 Ready:
-Write It for MIDAS on -4MIDAS
 MIDAS command file: -4MIDAS   Data file: -4MIDASMD
```

The specified MTS file contains a MIDAS Read command. Analysis of the data can be initiated using the MTS command '`$Run STAT:MIDAS input=-4MIDAS`' after exiting Micro.

Command Sequence to Calculate Ages

Usually a sequence of Micro commands is needed to perform a task, with one command using the Result of a previous command. In this example, we generate the ages of the students in the Fixed dataset from their birthdates, then print them in order from youngest to oldest.

Note: It is always better to store birthdates and derive ages. If you store ages, you must update each one every year.

We use the Calculate command to generate the students' ages in a new field, Age (`@scale=0` forces Age to have no decimal places). The expression first converts the value in the Birthdate field (in YYMMDD format) to an internal date form (measured in days) using the `from_yymmdd` function. It then subtracts this value from today's date in the internal date form returned by the `date` function, yielding the age measured in days. Finally, it divides this by 365.25 to get the age in years and truncates it using the `int` function.

```
 Ready:
-Calculate in Fixed Age@scale=0 = int((date() - from_yymmdd(Birthdate)) / 365.25)
  85 records in RESULT
  0 records in *Problems*
```

You may have noticed that the Calculate command creates a second dataset, *Problems*. If all goes well, there will be no records in *Problems*. However, if any problems occur during a calculation, the current record goes into the *Problems* dataset. Such problems include division by zero, a data value not fitting into a field, or an invalid date such as 590229 or 581301.

Now sort by Birthdate (not by Age) to get the ordering from oldest to youngest and then print it in reverse order.

```
 Ready:
-Sort It by Birthdate
  85 records in RESULT

 Ready:
-Print@reverse@count=15 in It Name, Birthdate@date, Age

  Name                          Birthdate        Age
  --------------------------------------------------
  White Albert L                 09-11-59         32
  Swift Sally Jane               11-30-58         33
  Wong David Lee                 11-20-58         33
  Bates Leslie Ann               11-15-58         33
  Hayes Hazel T                  11-11-58         33
  Fitzgerald Joan                10-15-58         33
  Erickson Sheila Mary           10-10-58         33
  Mann Laura D                   10-06-58         33
  Rosen Larry Z                  10-05-58         33
 Kuester Robert E           08-12-58          33
 Mandella Joanne D          08-11-58          33
 Benson Shelia Joan         08-03-58          33
 Schmidt Raymond F          05-05-58          33
 Grace Kenneth R            04-18-58          33
 Mack Joseph                04-01-58          33
```

Settings Command: Micro Settings

There are many parameter settings which affect Micro's behavior. Each has a default value which may be changed using the Settings command. Some settings control aspects of overall behavior, some the behavior of individual Micro commands. Here we tell the Xtab command to not print the Result set and to not generate the Percent field in the Result.

```
 Ready:
-Set  Xtab: Print off, Percent off
```

Here we tell the Print command to use field abbreviations as column headers and to use blanks instead of hyphens for the underline.

```
 Ready:
-Set Print: Field Abbr, Underline = ' '

 Ready:
-Xtab Fixed by Sex
  2 records in RESULT
  85 records represented

 Ready:
-Print it

  Sex             Cnt

  Female           46
  Male             39
```

The Resettings command (usually referred to as the Reset command) restores the default value for any setting. Here, the settings are restored with one Reset command; they could have been changed with one Set command also. '`Print:  *`' means to reset all of the Print command settings to their default values.

```
 Ready:
-Resetting  Xtab: Print, Percent;  Print: *
```

Save Command: Saving a Temporary Dataset

The Stop command terminates a Micro session after releasing all of the temporary datasets. To keep a temporary dataset around for a future Micro session, you must make it permanent using the Save command. Here we use '`as Students2`' to specify a different permanent name. Had we omitted it, Micro would have just used 'Fixed'.

```
 Ready:
-Save Fixed as Students2
 If the data for Students2 is to be scrambled,
     enter the scramble key:
 Can this dataset be Destroyed? (Yes,No) n
 Can this dataset be Replaced? (Yes,No) y
 Description: [default = Students2] copy of Students dataset from Tutorial 1
 Dictionary file = K59F:STUDENTS2#   Micro data file = K59F:STUDENTS2$
 Fixed has been saved as Students2.
```

Students, the parent dataset, is scrambled, so Micro asks if Students2 should be scrambled as well. Here the null reply tells Micro to not scramble it; anything else (except `Help` or `?`) would be used as a scramble key. Micro then asks if the dataset can be Destroyed or Replaced. The Destroy and Replace commands, which will be discussed in the next tutorial, have permanent consequences. Finally, Micro asks for a description of the new dataset. Take the time to enter a description that will help you (and others) to know what's in the dataset without having to Print or Document it. If you just enter a null reply, Micro uses the dataset name.

Micro puts Students2 into permanent MTS files (it prints the file names) and appends an entry for Students2 into the directory. Since this is the first dataset in the database for this userid, Micro automatically creates the directory file. Students2 will be available for use in the next Micro session.

Stop Command: Ending the Session

The Stop command exits Micro and returns us to MTS.

```
 Ready:
-Stop
   $.01,  $4.09M ($1.43S),  $7.52T
 Thu 5 Mar 1992  15:09:09
#Execution terminated   15:09:10  T=3.527
```

Command Language II

This is the second Micro Tutorial. We will introduce more Micro concepts and terminology while using the advanced Micro commands.

```
#$Run ILIR:Micro
#Execution begins   15:17:08
 ILIR:Micro (15 Dec 91)
  University of Michigan
 Thu 5 Mar 1992  15:17:09

  1 dataset from Directory
```

We will use the Students2 dataset, created in the first tutorial, and Fin-Aid from the ILIR database.

```
 Ready:
-Set  Command: Outlen 132

 Ready:
-Get ILIR
  8 datasets from ILIR:Directory

 Ready:
-List sets
 You have 9 datasets.

 Dataset name       Status Owner
  Students2          Perm   K59F
  Students           Perm   ILIR
  Fin-Aid            Perm   ILIR
  Cars               Perm   ILIR
  ART                Perm   ILIR
  Courses            Perm   ILIR
  CPS.Pers           Perm   ILIR
  CPS.Family         Perm   ILIR
  CPS.house          Perm   ILIR
```

Find Command: More Search Methods

Here are some variations on the Find command. These demonstrate a few conveniences as well as some things to know when searching character fields.

```
 Ready:
-Find in Fin-Aid where Amt is from 200 to 500
  16 records found (47.059%)
  16 records in RESULT
```

The '`Amt is from 200 to 500`' phrase includes the endpoints in the Result set. It is equivalent to '`Amt >= 200 and Amt <= 500`'. The '`Amt is between 200 and 500`' phrase below excludes the endpoints from the Result set. It is equivalent to '`Amt > 200 and Amt < 500`'.

```
 Ready:
-Find in Fin-Aid where Amt is between 200 and 500
  13 records found (38.235%)
  13 records in RESULT
```

You can refer to a category using its name, abbreviation, or value in a Find command. You can also use category numbers, but this is not commonly done. Category names and abbreviations are not case-sensitive. With well-defined categories, you can often forget what the underlying category values are. Do you remember what the value '`3`' means in the next command?

```
 Ready:
-F in Fin-Aid w Source-of-Funds = industry or UNIV or 3
  25 records found (73.529%)
  25 records in RESULT
```

Searches using character (alphanumeric) fields are somewhat different. The search is case-sensitive, the matching always begins with the first character in the field, and if there is a blank within the search value, it must be enclosed in primes or quotes. In this example, we try to search for the students with last name '`Jones`' and first initial '`W`'.

```
 Ready:
-F in Students2 w Name is "Jones W"
  0 records found (0.000%)
  0 records in RESULT
```

This fails because the '`is`' keyword (equivalent to '`=`' here) matches over the full length of the field (the Name field is 24 characters). The '`matches`' keyword matches over the first *n* characters of the field, where *n* is the length of the search string (in this case, 7 characters).

```
 Ready:
-F in Students2 w Name matches "Jones W"
  1 record found (1.176%)
  1 record in RESULT
```

The next command finds those students whose last names begin with either '`A`' or '`B`'.

```
 Ready:
-F in Students2 w Name matches A or B
  14 records found (16.471%)
  14 records in RESULT
```

External Data

The logical view of a dataset as a table forces a restriction on character fields: they must be a fixed length. For example, the Name field in Students2 is 24 characters long; longer names are truncated and shorter names are padded with blanks. Sometimes a "variable length" character field is needed such as the Remark* field in Fin-Aid.

In Micro this is handled using the *external* data type (type E). Each type E field has an *external data file*, an MTS line file, containing the variable length text. The external data file name is stored in the dictionary (see the Document command in Tutorial 1 for Fin-Aid); only the line numbers are stored in the table. The line numbers are used to access the text in the external data file. Currently, searches can't be done on the text in the external file itself, only on the line numbers stored in the table. A value of zero indicates there is no line in the external file.

Here we select only the Remark* field (weeding duplicates) and then print the Result to show the text and the underlying line numbers.

```
 Ready:
-Select in Fin-Aid Remark*
 Note:  20 duplicate records weeded.
  14 records in RESULT

 Ready:
-P in It Remark*@value, Remark*@width=70

      Remark*  Remark*
-----------------------------------------------------------------------------------
            0
         2000  Federal work/study with the Department of Labor
         3000  State grant to study automobile industry
         4000  Grant from the Oily Petroleum Co. to study oil slicks
         5000  Part-time employment grading papers
         6000  Grant from Belt Telephone Co. to study telepathy
         7000  Work in office of Secretary of State
        10000  Scholarship from Amalgamated Industries, Inc.
        11000  Part-time employment grading papers
        17000  Outstanding senior
        18000  Regents' Award
        19000  One time award
        20000  Governor's Award
        21000  Award from International Business Systems of Armonk, New York
```

The `@value` field modifier causes the line numbers to be printed (these are MTS internal line numbers: 6000 is line 6.000). The `@width` field modifier spreads out the remarks so they each fit on one line (the default width is 20 characters).

Key Fields

The field or fields needed to identify any one record in a dataset is known as a *key* for that dataset. The field created by the Key command is a special case of a key. The SIN field is a key for Students2: there is only one record in Students2 for each student, each with a unique value for SIN. If the key is included in the result of a Select command, there will be no weeding.

```
 Ready:
-Select in Students2 SIN
  85 records in RESULT
```

Since a student can receive more than one financial aid award, we can't expect SIN to be a key in Fin-Aid.

```
 Ready:
-Sel in Fin-Aid SIN
 Note:  1 duplicate record weeded.
  33 records in RESULT
```

As expected, weeding occurred. The following commands retrieve the duplicate SINs in Fin-Aid. The `@no%` and `@noprint` Xtab command modifiers tell Micro to not generate a Percent field and to not print the Result set.

```
 Ready:
-Xtab@no%@noprint in Fin-Aid SIN
  33 records in RESULT
  34 records represented

 Ready:
-F in It w Count > 1
  1 record found (3.030%)
  1 record in RESULT

 Ready:
-P in It SIN

  Student-ID-Num
 ----------------
       480780530
```

This command sequence is a general-purpose method for checking the uniqueness of a key. In all cases, if the field or fields specified in the Xtab command forms a key, the result of the Find will be null.

Restrict Command: Retrieval Using Links to Another Dataset

The commands we have seen so far deal with only one dataset at a time. But suppose we want to find those students who have received any form of financial aid. In other words, what records in Students2 have at least one record in Fin-Aid with a matching SIN? The Find command can't do this since it can deal with only one dataset. We must use the Restrict command.

```
 Ready:
-Restrict in Students2 where SIN is SIN in Fin-Aid
  25 records in RESULT
```

The result of this Restrict command is a subset of Students2. The records were "found" using information from another dataset, in this case Fin-Aid. The field or fields used to relate one dataset with another is called a *link*. The link between Students2 and Fin-Aid is the SIN field.

Splitting and Combining Datasets

Suppose we want to perform a long sequence of operations on only the records of students with financial aid, and then put the result back into Students2. We must somehow separate the records into those with aid and those without aid, do the operations on those with aid, and then combine the result together with the students without aid.

The current Result contains the students with financial aid. To get the students without financial aid, use the Remove command.

```
 Ready:
-Name It WithAid

 Ready:
-Remove WithAid from Students2
  25 records removed (29.412%)
  60 records in RESULT

 Ready:
-Name It WithoutAid
```

The Remove command creates a new Result set containing the records from one dataset that are not exactly the same as any of the records in a second dataset. Neither of the datasets is changed. The datasets must have the same fields in the same order.

After the operations are done, we can put the datasets back together using the Combine command, which creates a new Result set containing the records from both datasets. As with the Remove command, the datasets must have the same fields in the same order.

```
 Ready:
-Combine WithAid and WithoutAid
  85 records in RESULT
```

Release Command: Specifying which Datasets to Keep

If we want to release the accumulated temporary datasets, we could specify them all in a Release command. However, it might be easier to just tell Micro what datasets we want to keep active. This can be done using the '`all but`' phrase in the Release command.

```
 Ready:
-Release all but Students2 & Fin-Aid
```

Restrict Command: The Not-Restriction

The Restrict command can also be used to check a dataset for records having links NOT matching those in another dataset. For example, we can find the records that were in WithoutAid using a *not-restriction*.

```
Ready:
-Restrict in Students2 w SIN is not SIN in Fin-Aid
 60 records in RESULT
```

The not-restriction is commonly used to check the links between datasets. For example, we can see if there are any financial aid awards having no student to go with them, something that shouldn't happen. The notation '<>' is equivalent to '**is not**'.

```
Ready:
-Restrict in Fin-Aid w SIN <> SIN in Students2
 9 records in RESULT
```

RAJ: Joining Datasets on Matching Links

Suppose we want to compare some of the demographic information in Students2 (Sex and Race) with the financial aid information in Fin-Aid (Amount) using the Xtab command. This is not possible without somehow joining the information from both datasets together into a single dataset. We must use the RestrictAndJoin command.

To aid understanding, here are a few records from each dataset.

```
Ready:
-P@abbr@count=8 Students2

         SIN  Na                        Race  Sex        Bday ClYr
-------------------------------------------------------------------
    13222340  Richardson Deborah         Bl   Fe       550310   77
    95887328  Pond Stuart Leslie         Wh   Ma       560528   78
   113591707  Jones William              Wh   Ma       551010   77
   122045525  Grace Kenneth R            Wh   Ma       580418   80
   133680736  Lee Sue Jean               Or   Fe       560716   78
   139904799  Bailey Edith               Bl   Fe       570913   77
   142668782  Cook Cynthia Anna          Bl   Fe       560908   78
   149514226  Scott Mary Grace           Wh   Fe       561230   78

Ready:
-P@abbr@count=5 Fin-Aid

         SIN Year Term   CL Type  SOF           Amt  Rem*
--------------------------------------------------------------------------
   113591707  74     F   Sr   Tu Univ      1200.00
   123456789  75     F   So   Tu Univ       235.00
   133680736  74     F   Jr   RB FGov       485.00  Outstanding senior
   142668782  74     F   Jr   Tu  Ind      1050.00  Award from
                                                    International
                                                    Business Systems of
                                                    Armonk, New York
   149514226  75     W   Jr   Ca  Ind       100.00  Grant from the Oily
                                                    Petroleum Co. to
                                                    study oil slicks
```

The following RestrictAndJoin command (RAJ is a synonym) joins the records from the Students2 and Fin-Aid datasets together, using SIN as a link field, to form a new Result set. Each record in the Result set contains the link field, followed by the non-link fields from Students2, followed by the non-link fields from Fin-Aid. There is one record in the Result set for each match between an SIN in Students2 and an SIN in Fin-Aid. Records from Students2 and Fin-Aid having a non-matching SIN value do not contribute to the result. In other words, only the students with financial aid (those who were in WithAid) appear in the Result set along with their financial aid information.

```
 Ready:
-RAJ Students2 by SIN with Fin-Aid
  25 records in RESULT

 Ready:
-P@abbr@count=4

        SIN  Na                          Race  Sex      Bday ClYr Year Term   CL Type  SOF          Amt  Rem*
---------------------------------------------------------------------------------------------------------------------
  113591707  Jones William               Wh    Ma     551010   77   74    F   Sr   Tu Univ      1200.00
  133680736  Lee Sue Jean                Or    Fe     560716   78   74    F   Jr   RB FGov       485.00  Outstanding se
  142668782  Cook Cynthia Anna           Bl    Fe     560908   78   74    F   Jr   Tu  Ind      1050.00  Award from
                                                                                                         International
                                                                                                         Business Syste
                                                                                                         Armonk, New Yo
  149514226  Scott Mary Grace            Wh    Fe     561230   78   75    W   Jr   Ca  Ind       100.00  Grant from the
                                                                                                         Petroleum Co.
                                                                                                         study oil slic

 Ready:
-Name It StudentsWithAid
```

Now we can make the comparisons mentioned before.

```
 Ready:
-Xtab in StudentsWithAid Sex, Race, ave Amt, tot Amt
  9 records in RESULT
  25 records represented

 Sex           Ethnicity  Ave.Amount   Tot.Amount       Count      Percent
---------------------------------------------------------------------------
 Female            White    195.7273      2153.00          11        44.00
 Female            Black    795.0000      1590.00           2         8.00
 Female          Spanish    525.0000       525.00           1         4.00
 Female         Oriental    485.0000       485.00           1         4.00
 Male              White    454.1667      2725.00           6        24.00
 Male        Amer-Indian    400.0000       400.00           1         4.00
 Male            Spanish    200.0000       200.00           1         4.00
 Male           Oriental    150.0000       150.00           1         4.00
 Male              Other    125.0000       125.00           1         4.00
```

Validating Data

We can also do some data checking. The expected relationship between Class-Level (from Fin-Aid) and Class-Year (from Students2) is shown in the following table.

Class-Level vs Class-Year

Class-Level	value	Class-Year
Freshman	1	80
Sophomore	2	79
Junior	3	78
Senior	4	77

(Keep in mind that these datasets were built in 1977.) We can test this relationship now that both fields are in the StudentsWithAid dataset. First, we select the fields of interest and calculate the expected class-year (EClass-Year) from the Class-Level field.

```
 Ready:
-Select in StudentsWithAid SIN, Class-Level, Class-Year
  25 records in RESULT

 Ready:
-Calc in It EClass-Year = 81 - Class-Level
  25 records in RESULT
  0 records in *Problems*
```

Then we find the records where Class-Year and EClass-Year are not the same. The Find command allows field-to-field comparisons.

```
 Ready:
-Find in It w Class-Year is not EClass-Year
  1 record found (4.000%)
  1 record in RESULT

 Ready:
-P

  Student-ID-Num Class-Level  Class-Year EClass-Year
 ---------------------------------------------------
       360984224   Sophomore          80          79
```

Replace Command: Making Permanent Changes to a Dataset

We can fix such an error using the Change command, which creates a Result set with the change made, followed by a Replace command, which makes the change permanent. Assuming that the value for Class-Year in Students2 is incorrect ...

```
 Ready:
-Change in Students2 w SIN = 360984224 st Class-Year = 79
  1 record changed (1.176%)
  85 records in RESULT

 Ready:
-Replace Students2 with It
 Students2 is to be replaced.
     replace this dataset? (Yes,No) [default = No] y
 Students2 has been replaced with RESULT.
```

This is a permanent operation, so Micro asks for confirmation before doing the Replace. The '`[default = No]`' means that giving a null reply accepts the default value, 'No'.

Join Command: Including Records with Non-Matching Links

In the RAJ command, records with non-matching links do not contribute to the Result. This means that our StudentsWithAid dataset does not represent all students, only those who have financial aid. To do the comparisons for all students, we must use the Join command.

The Join command works like the RAJ command except that records with non-matching links are included in the Result set and appropriate "fill" values are inserted to complete each record.

```
 Ready:
-Join Students2 by SIN with Fin-Aid
  94 records in RESULT

 Ready:
-P@abbr@c=9

        SIN  Na                          Race  Sex      Bday ClYr Year Term   CL Type  SOF          Amt  Rem*
-------------------------------------------------------------------------------------------------------------------
   13222340  Richardson Deborah          Bl    Fe     550310   77    0    0    0    0    0          .00
   95887328  Pond Stuart Leslie          Wh    Ma     560528   78    0    0    0    0    0          .00
  113591707  Jones William               Wh    Ma     551010   77   74    F   Sr   Tu Univ      1200.00
  122045525  Grace Kenneth R             Wh    Ma     580418   80    0    0    0    0    0          .00
  123456789                              Unk   Unk      Miss  Unk   75    F   So   Tu Univ       235.00
  133680736  Lee Sue Jean                Or    Fe     560716   78   74    F   Jr   RB FGov       485.00  Outstanding se
  139904799  Bailey Edith                Bl    Fe     570913   77    0    0    0    0    0          .00
  142668782  Cook Cynthia Anna           Bl    Fe     560908   78   74    F   Jr   Tu  Ind      1050.00  Award from
                                                                                                        International
                                                                                                        Business Syste
                                                                                                        Armonk, New Yo
  149514226  Scott Mary Grace            Wh    Fe     561230   78   75    W   Jr   Ca  Ind       100.00  Grant from the
                                                                                                        Petroleum Co.
                                                                                                        study oil slic
```

Note that the students without financial aid are included with zeros in place of the missing Fin-Aid information. However, those financial aid records with no matching student are also included. In order to include all of the Students records but only the matching Fin-Aid records we can use the '`all`' and '`matching`' keywords in the Join command.

```
 Ready:
-Join all Students2 by SIN with matching Fin-Aid
  85 records in RESULT
```

If these keywords are omitted, as is often the case, the RAJ command behaves as if they are both '`matching`' while the Join command behaves as if they are both '`all`'. So we could have omitted the '`all`' in this Join command.

```
 Ready:
-P@abbr@c=8

        SIN  Na                          Race  Sex      Bday ClYr Year Term   CL Type  SOF          Amt  Rem*
-------------------------------------------------------------------------------------------------------------------
   13222340  Richardson Deborah          Bl    Fe     550310   77    0    0    0    0    0          .00
   95887328  Pond Stuart Leslie          Wh    Ma     560528   78    0    0    0    0    0          .00
  113591707  Jones William               Wh    Ma     551010   77   74    F   Sr   Tu Univ      1200.00
  122045525  Grace Kenneth R             Wh    Ma     580418   80    0    0    0    0    0          .00
  133680736  Lee Sue Jean                Or    Fe     560716   78   74    F   Jr   RB FGov       485.00  Outstanding se
  139904799  Bailey Edith                Bl    Fe     570913   77    0    0    0    0    0          .00
  142668782  Cook Cynthia Anna           Bl    Fe     560908   78   74    F   Jr   Tu  Ind      1050.00  Award from
                                                                                                        International
                                                                                                        Business Syste
                                                                                                        Armonk, New Yo
  149514226  Scott Mary Grace            Wh    Fe     561230   78   75    W   Jr   Ca  Ind       100.00  Grant from the
                                                                                                        Petroleum Co.
                                                                                                        study oil slic
```

```
Ready:
-Name It Students+Aid
```

In the comparison results, the totals are the same for the Students+Aid dataset as they are for the StudentsWithAid dataset, but the averages, counts, and percentages are different due to the fill zeros. In fact, more records appear for the Sex and Race combinations having no financial aid at all.

```
Ready:
-Xtab in Students+Aid Sex, Race, ave Amt, tot Amt
 11 records in RESULT
 85 records represented

 Sex          Ethnicity  Ave.Amount  Tot.Amount     Count    Percent
---------------------------------------------------------------------
 Female           White     69.4516     2153.00        31      36.47
 Female           Black    144.5455     1590.00        11      12.94
 Female     Amer-Indian       .0000         .00         1       1.17
 Female         Spanish    262.5000      525.00         2       2.35
 Female        Oriental    485.0000      485.00         1       1.17
 Male             White    100.9259     2725.00        27      31.76
 Male             Black       .0000         .00         4       4.70
 Male       Amer-Indian    200.0000      400.00         2       2.35
 Male           Spanish    100.0000      200.00         2       2.35
 Male          Oriental     75.0000      150.00         2       2.35
 Male             Other     62.5000      125.00         2       2.35
```

Destroy Command: Destroying a Dataset

If we attempt to Destroy the Students2 dataset, Micro balks.

```
Ready:
-Destroy Students2
You are not allowed to destroy Students2.
DESTROY command cancelled.
```

When we saved Students2 in the Tutorial 1, we specified that it could not be destroyed. Micro honors this and cancels the command. If the destroy operation is allowed for a permanent dataset, Micro asks for confirmation before proceeding. For temporary datasets, the Destroy command is the same as a Release command.

Keys and Links: Relationships between Datasets

The interactions between keys and links are central to the design of a relational database. A lack of understanding often leads to unexpected and incorrect results.

To demonstrate this, we will use some existing datasets from another directory. The datasets are small and simple so we can easily see the relationships between them.

```
Ready:
-Get K59K
 51 datasets from K59K:Directory
 Ready:
-P Faculty

         ID  Name
 ------------------------
          1  George
          2  Ronald
          3  Daniel
          4  Jesse
          5  Dave

 Ready:
-P Salaries

         ID      Salary
 ------------------------
          1   200000.00
          2  1075000.00
          3    50000.00
          4   100000.00
          6    60000.00
```

When the links are also keys in both datasets, there is a *one-to-one* relationship. In this example the ID field is a key in both Faculty and Salaries.

```
 Ready:
-Join Faculty by ID with matching Salaries
  5 records in RESULT

 Ready:
-P

         ID  Name             Salary
 -------------------------------------
          1  George        200000.00
          2  Ronald       1075000.00
          3  Daniel         50000.00
          4  Jesse         100000.00
          5  Dave                .00
```

Note that there is a faculty record with no salary record (ID = 5). The Result of the Join includes a record for this faculty member with a salary value of zero. There is also a salary record with no faculty record (ID = 6), but the '`matching`' keyword insures that we include only those salary records that match a faculty record.

When the links are keys in only one dataset, there is a *one-to-many* relationship. In this example we have a more detailed picture of the salary data showing joint appointments. The ID field is not a key in Appointments.

```
 Ready:
-P Appointments

         ID        Unit       Title      Salary
 -------------------------------------------------
          1     PoliSci   Professor   200000.00
          2    FineArts   ArtistRes    25000.00
          2     PoliSci  ProfEmerit    50000.00
          2 JapanStudys    Lecturer  1000000.00
          3     PoliSci  AssistProf    50000.00
          4    FineArts      Critic   25000.00
          4     PoliSci   AssocProf   75000.00
          6       Psych   Professor   60000.00

 Ready:
-Join Faculty by ID with their Appointments
  8 records in RESULT
```

The keyword '`their`' is a synonym for '`matching`'. In many contexts this provides a more natural syntax.

```
 Ready:
-P

          ID  Name              Unit       Title      Salary
-------------------------------------------------------------
           1  George         PoliSci   Professor   200000.00
           2  Ronald        FineArts   ArtistRes    25000.00
           2  Ronald         PoliSci  ProfEmerit    50000.00
           2  Ronald     JapanStudys    Lecturer  1000000.00
           3  Daniel         PoliSci  AssistProf    50000.00
           4  Jesse         FineArts      Critic    25000.00
           4  Jesse          PoliSci   AssocProf    75000.00
           5  Dave                 0           0         .00
```

When the links are not keys in either dataset, there is a *many-to-many* relationship. In this example we have the BadFaculty dataset containing duplicate records. The ID field is not a key in BadFaculty.

```
 Ready:
-P BadFaculty

          ID  Name
-------------------------
           1  George
           2  Ronald
           2  Ronnie
           3  Daniel
           4  Jesse
           5  Dave
           5  David

 Ready:
-Join BadFaculty by ID with their Appointments
  12 records in RESULT

 Ready:
-P

          ID  Name              Unit       Title      Salary
-------------------------------------------------------------
           1  George         PoliSci   Professor   200000.00
           2  Ronald        FineArts   ArtistRes    25000.00
           2  Ronald         PoliSci  ProfEmerit    50000.00
           2  Ronald     JapanStudys    Lecturer  1000000.00
           2  Ronnie        FineArts   ArtistRes    25000.00
           2  Ronnie         PoliSci  ProfEmerit    50000.00
           2  Ronnie     JapanStudys    Lecturer  1000000.00
           3  Daniel         PoliSci  AssistProf    50000.00
           4  Jesse         FineArts      Critic    25000.00
           4  Jesse          PoliSci   AssocProf    75000.00
           5  Dave                 0           0         .00
           5  David                0           0         .00
```

A many-to-many relationship is usually an error. In this case, it causes the salaries to be inflated. If you expect a link to be a key, you should make sure it is before doing a join operation.

It is possible to join datasets using no link at all. The result is known as the *Cartesian product* of the datasets. It joins every record of one dataset with every record of the other.

```
 Ready:
-Join Faculty with Salaries
  25 records in RESULT

 Ready:
-P

         ID  Name              ID       Salary
 ---------------------------------------------
          1  George             1    200000.00
          1  George             2   1075000.00
          1  George             3     50000.00
          1  George             4    100000.00
          1  George             6     60000.00
          2  Ronald             1    200000.00
          2  Ronald             2   1075000.00
          2  Ronald             3     50000.00
          2  Ronald             4    100000.00
          2  Ronald             6     60000.00
          3  Daniel             1    200000.00
          3  Daniel             2   1075000.00
          3  Daniel             3     50000.00
          3  Daniel             4    100000.00
          3  Daniel             6     60000.00
          4  Jesse              1    200000.00
          4  Jesse              2   1075000.00
          4  Jesse              3     50000.00
          4  Jesse              4    100000.00
          4  Jesse              6     60000.00
          5  Dave               1    200000.00
          5  Dave               2   1075000.00
          5  Dave               3     50000.00
          5  Dave               4    100000.00
          5  Dave               6     60000.00
```

The Cartesian product can be used to append fields to a dataset. In this example we give everyone an across-the-board raise using a percentage increase stored in the single-record Raise dataset.

```
 Ready:
-P Raise

        Raise
 -------------
          .05

 Ready:
-Join Salaries with Raise
  5 records in RESULT

 Ready:
-Calc in it Salary = Salary * (1 + Raise)
  5 records in RESULT
  0 records in *Problems*

 Ready:
-P
```

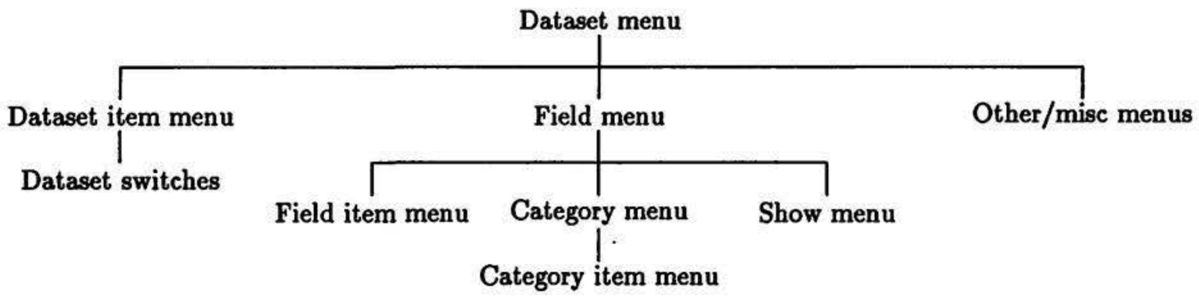
```
         ID      Salary       Raise
-----------------------------------
          1   210000.00         .05
          2  1128750.00         .05
          3    52500.00         .05
          4   105000.00         .05
          6    63000.00         .05

 Ready:
-Stop
   $.04,  $5.04M ($1.77S),  $13.78T
 Thu 5 Mar 1992  15:32:42
#Execution terminated   15:32:42  T=4.823
```

Database Definition

In the first two tutorials we used Micro to manipulate the Students and Fin-Aid datasets stored on the ILIR userid. In this tutorial, we will show you how these (or any other) datasets can be created from scratch.

The creation of a dataset is a separate process from entering data into the dataset (discussed in Tutorial 4). During the creation of the dataset, it is decided what fields (or variables) are to be included in the dataset, how they are to be represented, and what editing is to be performed during data entry.

To create a new dataset, just $Run ILIR:MicDef. MicDef is menu driven unless you request command mode. The following chart shows the organization of the menus.



MicDef allows three parameters. If you follow '`$Run ILIR:MicDef`' with '`PAR=NOMENU`', MicDef will run in command mode. Using '`PAR=DIR=`*userid*`:DIRECTORY`' allows you to edit the directory on another userid. Using '`PAR=TIME=`*n*' increases the time limit set by the program to *n* CPU seconds. However, you will usually want to run MicDef as demonstrated in the following example.

Running MicDef

```
#$Run ILIR:MicDef
#Execution begins   14:31:37
 ILIR:MicDef (7 Aug 91)
 Fri 6 Mar 1992  14:31:38
```

If this is the first database to be created on this userid, MicDef will offer some tips on getting started.

```
Would you like some help? (Yes,No) n
"K59F:DIRECTORY" has been created.

Dataset menu                                   Active dataset: "No.Active.Set"

       1. List existing datasets               5. Find datasets matching pattern
       2. Activate a set and edit fields       6. Exit menu (go to command mode)
       3. Change/describe dataset info         7. Help
       4. Other/misc commands                  8. Stop
```

This is the Dataset menu. It is the gateway to the other menus. Selections are made by Entering the number or first characters of your choice. Enter '2' (or 'A') to go to the Field menu. Enter '3' (or 'C') to go to the Dataset item menu. Enter '4' (or 'O') to go to the Other/misc menu.

Getting Help

Our first choice is to get more help. The Help selection will generate menus that are similar to the command menus. You can use the Help menus to preview and explain your options at any given point.

Items 1-7 are the choices available in the Dataset menu. You can select one of these items for an explanation of the choice and to preview the menus available from a selection. A null reply will return to the menu on the next higher level, in this case the Dataset menu.

```
              Enter command: help

Welcome to ILIR:MicDef.  This program will allow you to create and edit
dataset structures without using a command language. You will be
presented with a series of menus.  You simply choose a menu item and
MicDef will perform the requested operation.  It will prompt you for
everything it needs to know.  To select a menu item, just enter either
the capitalized letters associated with the desired item in the menu or
enter the number.  You can type ahead if you leave at least one blank
space between each menu selection.

You are currently in the dataset menu.  To get help with one of the
options in this menu, just enter a number from the following help menu.

-*-
 1. List existing datasets          4. Other/misc commands            7. Stop
 2. Activate a set and edit fields  5. Find datasets matching pattern
 3. Change/describe dataset info    6. Exit menu
-*-
Choose a Topic (or just Return to exit menu):

Dataset menu                                   Active dataset: "No.Active.Set"

       1. List existing datasets               5. Find datasets matching pattern
       2. Activate a set and edit fields       6. Exit menu (go to command mode)
       3. Change/describe dataset info         7. Help
       4. Other/misc commands                  8. Stop
```

Creating a Dataset in Menu Mode

We want to build a new dataset. To do this we make the appropriate menu selection.

```
          Enter command: activate
```

MicDef prompts for information needed to build a dataset.

```
Dataset name: Students
```

The dataset name is used to identify this dataset in Micro.

```
Is this a new dataset? (Yes,No) y
Description: Example Students dataset for the Micro Course -- Tutorial 3
Dictionary file "K59F:STUDENTS#" has been created.
  Dataset: Students           Example Students dataset for the Micro Course --
                              Tutorial 3 03-06-92

Dictionary file: K59F:STUDENTS#      Data file: K59F:STUDENTS$
    The dataset IS destroyable, it IS replaceable
```

MicDef displays all of the dataset items. Notice that many of them have default values supplied by MicDef. These will be discussed later.

Creating a Field

The dataset now exists but it has no fields, so MicDef asks for the name of the first field. The field name may be up to 16 characters long.

```
Field name: Student-ID-Num
```

The field abbreviation is a short form of the field name. It may be up to 4 characters long. A field name or abbreviation must not be the same as any other field name or abbreviation in the dataset. It must not be the same as any category name or abbreviation in the dataset.

```
Abbreviation: SIN
```

Micro allows integer, character, and categorical data types. The integer types are unsigned (U), and signed (S). The character types are fixed length (C) and variable length (E). The categorical types are unsigned (UC), signed (SC), and fixed length character (CC).

Since a student ID number is an unsigned integer, it's a type U.

```
Type: u
Maximum value: 999999999
Scaling function:
Data required? (Yes,No) Yes
Description: Student Identification Number

------------Field "Student-ID-Num" completed-----------
```

Entering categories would change the type from integer (U) to categorical (UC). A null reply means no categories.

```
Category name:
```

```
Field name: Name
```

If you enter a null reply, MicDef will try to take a default value. For a field abbreviation, the default is the first four characters of the field name. A field abbreviation can be identical to a field name if they refer to the same field.

```
Abbreviation:
```

Name is a fixed length character field, so it's a type C.

```
Type: c
Maximum number of characters (or name of field to overlap): 24
Data required? (Yes,No) y
Description: Student's Name (Last Middle First)
```

```
------------Field "Name" completed------------
```

MicDef allows you to anticipate the prompts.

```
Field name: Ethnicity Race
```

The abbreviation, Race, was not prompted for because it was already entered.

```
Type: uc
```

The number you enter next is used to calculate the amount of computer storage (number of bytes) to be allocated for the field. For an unsigned (U or UC) field the following table shows the number of bytes associated with each range.

Space Requirements for Unsigned Integer Data

Bytes	Minimum	Maximum
1	0	255
2	0	65535
3	0	167777215
4	0	2147483647

Here we enter a maximum value of 6. Micro needs only one byte to store this value. Micro can store any value from 0 to 255 in an unsigned, one byte field. When entering data, Micro will accept a value for an unsigned field if it falls within the range of minimum and maximum values.

```
Maximum value: 6
Scaling function:
Categories required? (Yes,No) y
Data required? (Yes,No)
Description:

------------Field "Ethnicity" completed------------
Category name:

Field name: Gender Sex cc
```

The prompts for abbreviation and type have been anticipated.

```
Maximum number of characters (or name of field to overlap): 1
Categories required? (Yes,No) y
Data required? (Yes,No) n
Description:

------------Field "Gender" completed------------
```

Creating a Category

Numeric fields, and character fields of length less than four, can have categories. A category is a name that is associated with a value. Within Micro you can refer to a value by using the category name. Here we associate the word '`Male`' with the value '`M`'. The value is all that will be stored in your data but '`Male`' will appear by default when you display this data point.

```
Category name: Male
Abbreviation: Ma
Category value: M
Description: Male

           Category  "Male" completed
```

You can anticipate the prompts for category information too.

```
Category name: Female Fe F Female

           Category  "Female" completed
Category name: Unknown Unk U Unknown

           Category  "Unknown" completed
Category name:

Field name: Class-Year ClYr SC
```

For a signed (S or SC) field the following table shows the number of bytes associated with each range.

Space Requirements for Signed Integer Data

Bytes	Minimum	Maximum
1	−128	127
2	−32768	32767
3	−8388608	8388607
4	−2147483648	2147483647

The Class-Year can range from −1 (missing) to 99. Entering 99 generates a one byte field. This will allow values between −128 and 127 to be stored. It is important to note that you must use the maximum absolute value for the field.

```
Maximum value: 99
Scaling function:
Categories required? (Yes,No)
Data required? (Yes,No)
Description:

------------Field "Class-Year" completed------------
Category name:
```

Entering a null reply to the field name prompt ends the field building loop and returns you to the Field menu. Now we have a dictionary structure that we could add data to. Rather than adding data, we're going to make some changes.

```
Field name:

Field menu                                    Active dataset: "Students"

     1. Append fields (terse)              6.  Modify/add categories
     2. Insert fields                      7.  Show documentation
     3. Redo a field                       8.  Help
     4. Delete fields                      9.  Back to dataset menu (return)
     5. Change field item
```

Adding a Category to an Existing Field

We can alter and display the structure of the active dataset, Students, from the Field menu. To add a category we select the Category menu by entering '`m`' (choice 6).

```
          Enter command: m

Category menu          Dataset: "Students"         Field: "No.active.field"

     1. Append categories                   6. Show categories
     2. Insert categories                   7. READ categories from a file
     3. Redo a category                     8. Edit another field
     4. Delete categories                   9. Help
     5. Change category item               10. Back to field menu (return)
```

Let's add the category.

```
          Enter command: 1
```

The top line of the above Category menu says there is no active field. When there is no active field the program prompts for a field to activate. This is the field that the category will be added to.

```
Field: Class-Year
Category name: Unknown
Abbreviation: Unk
```

Getting Help From a Prompt

What's a default category? To find out, we enter 'HELP'.

```
Default category? (Yes,No) help

The default category is used by the ENTERDATA and JOIN
commands, as well as Micro.add, to fill in values where data are
missing.

If you enter "y", then the value you enter at the "category
value" prompt will become the value for this field in records
that are missing data.  Missing datum is generated by a null reply
to a prompt in Micro.add or ENTERDATA, or when key fields have
no match in joined datasets.

If you enter a null reply or "n" then this category will not be
the default.

While a field may have more than one default category, only the
first will be used.
Default category? (Yes,No) n
Category value: -1
Description:

              Category  "Unknown" completed
```

More categories could be added now, but we enter a null reply to end the category building loop and return to the Category menu.

```
Category name:

Category menu          Dataset: "Students"         Field: "Class-Year"

     1. Append categories                    6. Show categories
     2. Insert categories                    7. READ categories from a file
     3. Redo a category                      8. Edit another field
     4. Delete categories                    9. Help
     5. Change category item                10. Back to field menu (return)
```

Adding Categories from a File

An MTS command can be entered at any prompt. Just enter a '$' in front of the command. Here we list a file containing previously entered category information.

```
           Enter command: $list k59f:cat.race

     1     Unknown        Unk    Y    0    Unknown
     2     White          Wh     n    1    White
     3     Black          Bl     n    2    Black
     4     Amer-Indian    AI     n    3    American Indian
     5     Spanish        Sp     n    4    Spanish-Surnamed American
     6     Oriental       Or     n    5    Oriental
     7     Other          Oth    n    6    Other
     8     <null>
```

Now we add categories to a field from the file. First select the field into which the categories are to be added.

```
        Enter command: e
Field: race

Category menu         Dataset: "Students"        Field: "Ethnicity"

     1. Append categories                      6. Show categories
     2. Insert categories                      7. READ categories from a file
     3. Redo a category                        8. Edit another field
     4. Delete categories                      9. Help
     5. Change category item                  10. Back to field menu (return)

        Enter command: read
```

Now we add the categories.

```
Append categories from which file? k59f:cat.race

Category menu         Dataset: "Students"        Field: "Ethnicity"

     1. Append categories                      6. Show categories
     2. Insert categories                      7. READ categories from a file
     3. Redo a category                        8. Edit another field
     4. Delete categories                      9. Help
     5. Change category item                  10. Back to field menu (return)
```

Showing the Field Structure

Did it work? Let's 'Show' categories to find out. Categories can be referred to by name, abbreviation, or sequence number. Here we refer to the range of the first category, 'C1', through the last category, 'CL'.

```
        Enter command: s
Enter a null reply (return) to show all categories or
Enter a pattern, category range, or category: c1 thru cl
---------------------------------------------------------------------------
  F# Field name      Abbr        Value Description
---------------------------------------------------------------------------
  F3 Ethnicity       Race              Ethnicity
          7 categories (required)
       Unknown       Unk             0 Unknown
       White         Wh              1 White
       Black         Bl              2 Black
       Amer-Indian   AI              3 American Indian
       Spanish       Sp              4 Spanish-Surnamed American
       Oriental      Or              5 Oriental
       Other         Oth             6 Other

Category menu         Dataset: "Students"        Field: "Ethnicity"

     1. Append categories                      6. Show categories
     2. Insert categories                      7. READ categories from a file
     3. Redo a category                        8. Edit another field
     4. Delete categories                      9. Help
     5. Change category item                  10. Back to field menu (return)
```

Inserting a New Field

Now we return to the Field menu to insert a new field after the field Sex.

```
        Enter command: b

Field menu                                   Active dataset: "Students"

    1. Append fields (terse)                6.  Modify/add categories
    2. Insert fields                        7.  Show documentation
    3. Redo a field                         8.  Help
    4. Delete fields                        9.  Back to dataset menu (return)
    5. Change field item

        Enter command: 2

Insert new fields after which field? sex
```

A null reply here would make the new field the first one in the dataset.

```
Field name: Birthdate
Abbreviation: Bday
Type: uc
Maximum value: 991231
Scaling function:
Categories required? (Yes,No)
Data required? (Yes,No)
Description: Date of Birth (yymmdd)

------------Field "Birthdate" completed------------
Category name: Missing
```

The default category abbreviation is the first four characters of the category name. Enter a null reply to use the default.

```
Abbreviation:
Default category? (Yes,No) y
Category value: 0
```

The default description is the category name. Enter a null reply to use the default.

```
Description:

          Category  "Missing" completed
```

End the add category loop by entering a null reply.

```
Category name:
```

End the add field loop by entering a null reply.

```
Field name:

Field menu                                   Active dataset: "Students"

    1. Append fields (terse)                6.  Modify/add categories
    2. Insert fields                        7.  Show documentation
    3. Redo a field                         8.  Help
    4. Delete fields                        9.  Back to dataset menu (return)
    5. Change field item
```

Showing Dataset Structure

The Show menu, choice 7 on the Field menu, displays options for documenting the structure of datasets.

```
          Enter command: s

Show menu                                  Active dataset: "Students"

     1. Fulldoc on screen                  6.  PF (Print Fulldoc on file
     2. Doc on screen                      7.  PD (Print Doc on file)
     3. Tdoc on screen                     8.  PT (Print Tdoc on file)
     4. DT (Doc Terse)                     9.  Help
     5. TT (Tdoc Terse)                   10.  Back to field menu (return)
```

Fulldoc displays all information about a dataset. Output is 132 columns wide so scrolling is necessary to see all the information on the screen. Doc displays non-technical information in an 80 column display. TDoc displays technical information in an 80 column display. Terse options (`'Fulldoc *'`) do not display categories.

```
          Enter command: fulldoc
Enter a null reply (return) to show all fields or
```

Fields are numbered F1, F2, ..., FL. Here SIN is the abbreviation of the first field. FL is the number of the last field.

```
Enter a pattern, field range, or field: sin thru fl
  Dataset: Students            Example Students dataset for the Micro Course --
                               Tutorial 3 03-06-92

Dictionary file: K59F:STUDENTS#      Data file: K59F:STUDENTS$
   The dataset IS destroyable, it IS replaceable, it is NOT scrambled
-------------------------------------------------------------------------------------------------------------
  F#  Field name      Abbr        Value  Type Length Scale Factor                Description
-------------------------------------------------------------------------------------------------------------
  F1  Student-ID-Num  SIN       DataReq   U      4                               Student Identification Number
  F2  Name            Name      DataReq   C     24                               Student's Name (Last Middle First)
  F3  Ethnicity       Race                UC     1                               Ethnicity
           7 categories (required)
        Unknown       Unk             0  Default                                 Unknown
        White         Wh              1                                          White
        Black         Bl              2                                          Black
        Amer-Indian   AI              3                                          American Indian
        Spanish       Sp              4                                          Spanish-Surnamed American
        Oriental      Or              5                                          Oriental
        Other         Oth             6                                          Other
  F4  Gender          Sex                 CC     1                               Gender
           3 categories (required)
        Male          Ma          M                                              Male
        Female        Fe          F                                              Female
        Unknown       Unk         U                                              Unknown
  F5  Birthdate       Bday                UC     3                               Date of Birth (yymmdd)
           1 category
        Missing       Miss            0  Default                                 Missing
  F6  Class-Year      ClYr                SC     1                               Class-Year
           1 category
        Unknown       Unk            -1                                          Unknown
```

```
Field menu                                    Active dataset: "Students"

     1. Append fields (terse)              6.  Modify/add categories
     2. Insert fields                      7.  Show documentation
     3. Redo a field                       8.  Help
     4. Delete fields                      9.  Back to dataset menu (return)
     5. Change field item
```

Editing a Field

The description for field Name is in error. To fix it we must go to the Change Field Item menu.

```
          Enter command: c

Field item menu      Dataset: "Students"       Field: "No.active.field"

     1. Name                               7.  Categories required
     2. Abbreviation                       8.  DATA required
     3. Length or max value                9.  View field
     4. Scale  function                   10.  Edit another field
     5. Factor                            11.  Help
     6. Description                       12.  Back to field menu (return)
```

A field is composed of the first 8 items listed above. To change the description we select the description item. Descriptions are echoed to the screen for visual editing.

```
          Enter command: D
Field: name
Current description:

Description: Students Name (Last First Middle)

Field item menu      Dataset: "Students"       Field: "Name"

     1. Name                               7.  Categories required
     2. Abbreviation                       8.  DATA required
     3. Length or max value                9.  View field
     4. Scale  function                   10.  Edit another field
     5. Factor                            11.  Help
     6. Description                       12.  Back to field menu (return)
```

Adding a Default Category

Now we want to add a default category to the field Gender. To do this we must go up a level to the Field menu and then down the Category menu branch to the Category Item menu.

```
          Enter command: 12

Field menu                                    Active dataset: "Students"

     1. Append fields (terse)              6.  Modify/add categories
     2. Insert fields                      7.  Show documentation
     3. Redo a field                       8.  Help
     4. Delete fields                      9.  Back to dataset menu (return)
     5. Change field item

          Enter command: m
```

```
Category menu          Dataset: "Students"        Field: "No.active.field"

     1. Append categories                       6. Show categories
     2. Insert categories                       7. READ categories from a file
     3. Redo a category                         8. Edit another field
     4. Delete categories                       9. Help
     5. Change category item                   10. Back to field menu (return)

          Enter command: c

Category item menu     Dataset: "Students"        Field: "No.active.field"

     1. Name                                   5.  Description
     2. Abbreviation                           6.  Edit another field
     3. Status (default)                       7.  Help
     4. Value                                  8.  Back to categorymenu (return)
```

From here we can change category items.

```
          Enter command: 3
Field: sex
Which category? unk
Is this a default category? (Yes,No) Y

Category item menu     Dataset: "Students"        Field: "Gender"

     1. Name                                   5.  Description
     2. Abbreviation                           6.  Edit another field
     3. Status (default)                       7.  Help
     4. Value                                  8.  Back to categorymenu (return)
```

Moving Between Menus

A null reply returns you up a level of the menu. Multiple null replies return you to the top menu level (the Dataset menu).

```
          Enter command:

Category menu          Dataset: "Students"        Field: "Gender"

     1. Append categories                       6. Show categories
     2. Insert categories                       7. READ categories from a file
     3. Redo a category                         8. Edit another field
     4. Delete categories                       9. Help
     5. Change category item                   10. Back to field menu (return)

          Enter command:

Field menu                                     Active dataset: "Students"

     1. Append fields (terse)                  6.  Modify/add categories
     2. Insert fields                          7.  Show documentation
     3. Redo a field                           8.  Help
     4. Delete fields                          9.  Back to dataset menu (return)
     5. Change field item

          Enter command:

Dataset menu                                   Active dataset: "Students"
```

```
      1. List existing datasets           5. Find datasets matching pattern
      2. Activate a set and edit fields   6. Exit menu (go to command mode)
      3. Change/describe dataset info     7. Help
      4. Other/misc commands              8. Stop
```

Saving a Dataset Definition

All of the work we have done so far is in computer memory. To save it we must put it in a file. This is done automatically when you enter stop. If you exit the program using the MTS command, or by turning off your terminal you will lose your work.

```
          Enter command: s
 "K59F:STUDENTS#" has been saved.
 Fri 6 Mar 1992  14:43:09
#Execution terminated   14:43:08  T=1.229
```

The message '"K59F:STUDENTS#" has been saved' tells us that our work has been made permanent. The file K59F:STUDENTS# is the dictionary file that holds the structure of the new Students dataset.

We can run MicDef again to see what it looks like.

```
#$Run ILIR:MicDef
#Execution begins   14:43:28
 ILIR:MicDef (7 Aug 91)
 Fri 6 Mar 1992  14:43:29


 Dataset menu                              Active dataset: "No.Active.Set"

      1. List existing datasets           5. Find datasets matching pattern
      2. Activate a set and edit fields   6. Exit menu (go to command mode)
      3. Change/describe dataset info     7. Help
      4. Other/misc commands              8. Stop
```

Students already exists so activating it does not cause MicDef to ask for field information.

```
          Enter command: a
 Dataset name: students

 Field menu                                Active dataset: "Students"

      1. Append fields (terse)           6.  Modify/add categories
      2. Insert fields                   7.  Show documentation
      3. Redo a field                    8.  Help
      4. Delete fields                   9.  Back to dataset menu (return)
      5. Change field item
```

From here we can make changes to the dictionary, or show what it looks like.

```
          Enter command: 7

 Show menu                                 Active dataset: "Students"

      1. Fulldoc on screen               6.  PF (Print Fulldoc on file
      2. Doc on screen                     7. PD (Print Doc on file)
      3. Tdoc on screen                    8. PT (Print Tdoc on file)
      4. DT (Doc Terse)                    9. Help
      5. TT (Tdoc Terse)                  10. Back to field menu (return)

          Enter command: pt
 Enter a null reply (return) to show all fields or
 Enter a pattern, field range, or field: f1 name race f4 f5 f1
 Enter a new output file or device: -tdoc

 Field menu                                Active dataset: "Students"

      1. Append fields (terse)             6. Modify/add categories
      2. Insert fields                     7. Show documentation
      3. Redo a field                      8. Help
      4. Delete fields                     9. Back to dataset menu (return)
      5. Change field item

          Enter command: $copy -tdoc
>-----------------------------------------------------------------------
> F#  Field name       Abbr        Value  Type Length Scale Factor Disp
>-----------------------------------------------------------------------
> F1  Student-ID-Num   SIN       DataReq  U       4                   0
> F2  Name             Name      DataReq  C      24                   4
> F3  Ethnicity        Race               UC      1                  28
>           7 categories (required)
>        Unknown       Unk             0  Default
>        White         Wh              1
>        Black         Bl              2
>        Amer-Indian   AI              3
>        Spanish       Sp              4
>        Oriental      Or              5
>        Other         Oth             6
> F4  Gender           Sex                CC      1                  29
>           3 categories (required)
>        Male          Ma         M
>        Female        Fe         F
>        Unknown       Unk        U       Default
> F5  Birthdate        Bday               UC      3                  30
>           1 category
>        Missing       Miss            0  Default
> F6  Class-Year       ClYr               SC      1                  33
>           1 category
>        Unknown       Unk            -1
```

The Students dataset is done. Next we'll build the Fin-Aid dataset in command mode. Enter an Attn, a Ctrl-C, or $Endfile to go to command mode.

```
          Enter command: $endfile
 The MENU command will return you to menu mode.
 Corrections:
```

Using Command Mode

The following commands may be used in MicDef's command mode. To see a command's syntax, examples, or command description, just select it from the help menu.

```
 -: help
 -*-
   1. Alter command       7. DOcument command  13. Menu command
   2. APPend command      8. EDit command      14. Permit command
   3. COpy command       9. Get command       15. Redo command
   4. DELete command    10. HELP command      16. Show command
   5. Describe command  11. Insert command    17. Write command
   6. DESTroy command   12. Move command
 -*-
 Choose a Topic (or just Return to exit menu):
```

Overriding Default Attributes

The Edit command activates an existing dataset or, as in this case, creates a new one. Specifying the FULL option allows you to override the MicDef default settings.

```
-: edit FULL Fin-Aid
Is this a new dataset? (Yes,No) y
Description: Example Fin-Aid dataset for the Micro Course -- Tutorial 3
```

This determines if Micro's Destroy command can destroy this dataset (the MTS $Destroy command can destroy the files that hold this set regardless of your answer here).

```
Can this dataset be Destroyed? (Yes,No) y
```

This determines if Micro's Replace command can replace this dataset. You can make permanent changes to records and data structures using Replace.

```
Can this dataset be Replaced? (Yes,No) y
```

Scrambled data can not be seen without entering a scramble key. Answering '**Yes**' here will cause Micro to prompt for the scramble key whenever the data for this dataset is first accessed.

```
Will this dataset be Scrambled? (Yes,No) n
```

This is the name of the file that Micro will store the data in. The default data file name is taken from the first 11 characters of the dataset name with a '**$**' appended. The data file is a sequential file usually created and maintained by Micro.

```
Micro Data file: fin-aid$
```

The conversion format file is used by Micro.Cnvrt to enter fixed format data. It is discussed in Tutorial 4. A null reply here means that no conversion format file will be associated with this dataset.

```
Conversion Format file:
```

This is the name of the file in which Micro will store the data structure we are about to build. The default dictionary file name is formed by taking the first 11 characters of the dataset name and appending '**#**'.

```
Dictionary file: fin-aid#
Dictionary file "fin-aid#" has been created.
  Dataset: Fin-Aid            Example Fin-Aid dataset for the Micro Course --
                              Tutorial 3 03-06-92

Dictionary file: K59F:FIN-AID#        Data file: K59F:FIN-AID$
    The dataset IS destroyable, it IS replaceable
```

Creating a Link Field

The Student-ID-Num field will be used as a *link field* in Micro to join Fin-Aid with Students. Micro does not require the names and abbreviations of link fields to be the same, but it is usually a good idea to do so. However, the types and lengths of link fields must agree.

```
Field name: Student-ID-Num
Abbreviation: SIN
Type: u
```

The length of Student-ID-Num in Students is four bytes. The length of Student-ID-Num in Fin-Aid must be four bytes too. It is possible to specify the length directly by using the letter 'B' (for byte) followed by the number of bytes. Entering 'b4' here will create a four byte field.

```
Maximum value: b4
Scaling function:
Data required? (Yes,No) y
Description: Student Identification Number

------------Field "Student-ID-Num" completed------------
Category name:

Field name: Year-Awarded Year u 99 0 y Year Awarded (yy)

------------Field "Year-Awarded" completed------------
```

Observe the scale function entered while anticipating the prompts is 0. This indicates no scaling.

```
Category name:

Field name: Term-Awarded Term uc b1 0
Categories required? (Yes,No) y y Term Awarded (YY)

------------Field "Term-Awarded" completed------------
Category name:

Field name:
Corrections:
```

Append Command: Appending Categories

The Append command can be used to add categories to a field.

```
-: append f3
Category name: fall
Abbreviation: F
Default category? (Yes,No)
Category value: 1
Description:

            Category  "fall" completed
```

If an error is made while anticipating prompts, only the error needs to be entered again. Here, 'Term-Awarded' (F3), has been defined to be a 1 byte type UC field. The first table shows that the maximum value of a 1 byte type UC field is 255. Since the category value 257 is greater than 255, MicDef asks for a replacement.

```
Category name: Winter W n 257 Winter
scaled values must be between 0 and 255.
Enter another value to replace "257"
    (or enter HELP, CANCEL): 2

               Category  "Winter" completed
Category name: Summer W n 3 Summer
"W" has already been used.
Enter another category abbreviation to replace "W"
    (or enter HELP, null reply, CANCEL): S

               Category  "Summer" completed
```

Category names and abbreviations must be distinct from all other category names and abbreviations in the same field, and from all field names and abbreviations in the same dataset. '`W`' was used as a category abbreviation in the previous category, so it must be replaced.

```
Category name:
Corrections:
```

Fields can be added quickly by accepting default values for certain field items. The default values are: data not required, categories not required, no scaling function, and no categories. To accept these defaults just add '`*`' to the Append command.

```
-: append *

Field name: Class-Level
Abbreviation: CL
Type: uc
Maximum value: 4
Description:

------------Field "Class-Level" completed-----------

Field name: Remark*
Abbreviation: Rem*
Type: e
```

Type E fields hold variable length character data. The character data is stored in an external data file, which is created and maintained by Micro. The line numbers indexing into the external data file are stored in the Micro data file.

```
External Data file: k59f:remark*
Description: Additional remarks

------------Field "Remark*" completed-----------

Field name:
Corrections:
```

Insert Command: Inserting Categories

The Insert command can insert fields or categories. The word '`in`' is used to insert categories. If a category name followed the field name, categories would be inserted after that category. Since there is no category named here, the first category inserted will be the first category.

```
-: insert in class-level
Category name: Freshman
Abbreviation: Fr
Default category? (Yes,No)
Category value: 1
Description:

            Category  "Freshman" completed
Category name: Sophomore So no 2 Sophomore

            Category  "Sophomore" completed
Category name: Junior Jr no 3 Junior

            Category  "Junior" completed
Category name: Senior Sr no 4 Senior

            Category  "Senior" completed
Category name:
Corrections:
```

Insert Command: Inserting a Field

Next is an example of the Insert command inserting fields after the field Class-Level (abbreviation 'cl').

```
-: ins cl

Field name: Type-of-Aid
Abbreviation: Type
Type: uc
Maximum value: b1
Scaling function:
Categories required? (Yes,No) y
Data required? (Yes,No)
Description: Type of aid

------------Field "Type-of-Aid" completed------------
Category name: Tuition Tu
Default category? (Yes,No) n 1 Tuition

            Category  "Tuition" completed
Category name: Room-board Rb n 2 room and Board

            Category  "Room-board" completed
Category name: Cash Ca n 3
Description:

            Category  "Cash" completed
Category name: Loan Lo n 4 Loan

            Category  "Loan" completed
Category name: Work-Study Ws n 5 Work Study

            Category  "Work-Study" completed
Category name: Other Oth n 6 Other

            Category  "Other" completed
Category name:

Field name: Source-of-funds SOF uc 1 0 y n Source of funds

------------Field "Source-of-funds" completed------------
Category name: University Univ n 1 University
```

```
            Category  "University" completed
Category name: State-Gov SGov n 2 State Government

            Category  "State-Gov" completed
Category name: Federal-Gov FGov n 3 Federal Government

            Category  "Federal-Gov" completed
Category name: Industry Ind n 4 Industry

            Category  "Industry" completed
Category name: Foundation
Abbreviation:
Default category? (Yes,No)
Category value: 5
Description:

            Category  "Foundation" completed
Category name: Other Oth n 6 Other

            Category  "Other" completed
Category name:

Field name:
Corrections:
```

Show Command: Showing the Field Structure

The Fdoc command (Show is a synonym) can display the entire dataset or, in this case, just one field.

```
-: show sof
----------------------------------------------------------------------------------------------------------
  F#  Field name       Abbr        Value  Type Length Scale Factor              Description
----------------------------------------------------------------------------------------------------------
  F6  Source-of-funds  SOF                UC      1                             Source of funds
          6 categories (required)
         University    Univ          1                                          University
         State-Gov     SGov          2                                          State Government
         Federal-Gov   FGov          3                                          Federal Government
         Industry      Ind           4                                          Industry
         Foundation    Foun          5                                          Foundation
         Other         Oth           6                                          Other
Corrections:
```

Redo Command: Redoing a Field

The Redo command is the only command that will change the field type. It will maintain the category list when it is logically possible.

```
-: redo sof
Field name: Source-of-Funds
Abbreviation: SOF
```

Changing the type from UC to SC will allow negative numbers to be used. Any changes in type must be done before data is entered. Otherwise the data may not agree with the structure and errors will occur.

```
Type: sc
Maximum value: b1
Scaling function:
Data required? (Yes,No)
Description:
Category name:
Corrections:
-: s sof
------------------------------------------------------------------------------------------------------------
  F#  Field name       Abbr        Value  Type Length Scale Factor          Description
------------------------------------------------------------------------------------------------------------
  F6  Source-of-Funds  SOF                 SC     1                         Source-of-Funds
           6 categories
        University     Univ          1                                      University
        State-Gov      SGov          2                                      State Government
        Federal-Gov    FGov          3                                      Federal Government
        Industry       Ind           4                                      Industry
        Foundation     Foun          5                                      Foundation
        Other          Oth           6                                      Other
Corrections:
```

Now we can add a category with a negative value to Source-of-Funds.

```
-: ins in sof
Category name: Unknown
Abbreviation: Unk
Default category? (Yes,No) n
Category value: -1
Description:

          Category  "Unknown" completed
Category name:
Corrections:
-: ins sof

Field name: Amount
Abbreviation: Amt
Type: u
```

Field Scaling

Micro supports real numbers by scaling integers. A number with two decimal places is treated as an integer divided by 100. The integer is stored in your data file and the scale function division ('/') and scale factor ('100') is part of the structure. The function and factor are applied to your data when you enter or retrieve it. The major shortcoming of this method is that the decimal places count against your maximum value. An unscaled integer can have a value no larger than 2,147,483,647. A field scaled by 100 can hold a value no larger than 21,474,836.47. It is important to enter the decimal places when entering the Maximum value. Entering 9999999.99 will produce a four byte field that can hold up to 21,474,836.47. Entering 1000000 will produce a three byte that can hold up to 1,677,772.15

```
Maximum value: 9999999.99
```

Valid scale functions are '/', '*', '-', '+', and '0', the default function, none. For scaling by multiplication and division, the scale factor must be a power of 10.

```
Scaling function: /
Scaling factor: 100
Data required? (Yes,No)
Description:

-----------Field "Amount" completed-----------
Category name:

Field name:
Corrections:
```

Patterns for Names

Patterns can be used in many commands. The following command displays all fields with the letter 'e' in their names.

```
-: show ?e?
--------------------------------------------------------------------------------------------------------
  F#  Field name       Abbr        Value  Type Length Scale Factor          Description
--------------------------------------------------------------------------------------------------------
  F1  Student-ID-Num   SIN       DataReq   U       4                         Student Identification Number
  F2  Year-Awarded     Year      DataReq   U       1                         Year Awarded (yy)
  F3  Term-Awarded     Term      DataReq   UC      1                         Term Awarded (YY)
            3 categories (required)
         fall          F               1                                     fall
         Winter        W               2                                     Winter
         Summer        S               3                                     Summer
  F4  Class-Level      CL                  UC      1                         Class-Level
            4 categories
         Freshman      Fr              1                                     Freshman
         Sophomore     So              2                                     Sophomore
         Junior        Jr              3                                     Junior
         Senior        Sr              4                                     Senior
  F5  Type-of-Aid      Type                UC      1                         Type of aid
            6 categories (required)
         Tuition       Tu              1                                     Tuition
         Room-board    Rb              2                                     room and Board
         Cash          Ca              3                                     Cash
         Loan          Lo              4                                     Loan
         Work-Study    Ws              5                                     Work Study
         Other         Oth             6                                     Other
  F6  Source-of-Funds  SOF                 SC      1                         Source-of-Funds
            7 categories
         Unknown       Unk            -1                                     Unknown
         University    Univ            1                                     University
         State-Gov     SGov            2                                     State Government
         Federal-Gov   FGov            3                                     Federal Government
         Industry      Ind             4                                     Industry
         Foundation    Foun            5                                     Foundation
         Other         Oth             6                                     Other
  F8  Remark*          Rem*                E       4                         Additional remarks
          External file = K59F:REMARK*
Corrections:
```

Only one field does not have an 'e'. We show it for completeness.

```
-: show amount *
--------------------------------------------------------------------------------------------------------
  F#  Field name       Abbr        Value  Type Length Scale Factor          Description
--------------------------------------------------------------------------------------------------------
  F7  Amount           Amt                 U       4  /        100           Amount
Corrections:
```

Change Command: Changing Field Items

The Change command can be used to change field items. The following command makes categories required.

```
-: ch sof catsreq to yes
Corrections:
-: show sof
------------------------------------------------------------------------------------------------------------------
  F#  Field name       Abbr        Value  Type Length Scale Factor              Description
------------------------------------------------------------------------------------------------------------------
  F6  Source-of-Funds  SOF                 SC      1                            Source-of-Funds
            7 categories (required)
        Unknown        Unk          -1                                          Unknown
        University     Univ          1                                          University
        State-Gov      SGov          2                                          State Government
        Federal-Gov    FGov          3                                          Federal Government
        Industry       Ind           4                                          Industry
        Foundation     Foun          5                                          Foundation
        Other          Oth           6                                          Other
Corrections:
```

Menu Mode

The Menu command returns us to the menu we exited.

```
-: menu

Field menu                                  Active dataset: "Fin-Aid"

     1. Append fields (terse)               6. Modify/add categories
     2. Insert fields                       7. Show documentation
     3. Redo a field                        8. Help
     4. Delete fields                       9. Back to dataset menu (return)
     5. Change field item

          Enter command:

Dataset menu                                Active dataset: "Fin-Aid"

     1. List existing datasets               5. Find datasets matching pattern
     2. Activate a set and edit fields       6. Exit menu (go to command mode)
     3. Change/describe dataset info         7. Help
     4. Other/misc commands                  8. Stop
```

A null reply returns us to the Dataset menu. Selecting List displays all the datasets available on this userid.

```
          Enter command: 1

Students
Fin-Aid
"Fin-Aid" is the currently active dataset.

Dataset menu                                Active dataset: "Fin-Aid"

     1. List existing datasets               5. Find datasets matching pattern
     2. Activate a set and edit fields       6. Exit menu (go to command mode)
     3. Change/describe dataset info         7. Help
     4. Other/misc commands                8. Stop
```

To change a dataset item, such as its destroy status, we select 3.

```
         Enter command: 3

Dataset item menu                          Active dataset: "Fin-Aid"

     1. List files in active set          7.  Edit the dataset description
     2. Pattern matching dataset names    8.  Set scramble, dest & rep swit
     3. DSN (dataset name) change         9.  Redo dataset info
     4. DICtionary file name change      10.  Help
     5. Micro data file name change      11.  Back to master menu (return)
     6. Format file name change
```

Items 4 through 8 are dataset items. To edit the description we select item 7. The current description is echoed.

```
         Enter command: e
Current description:
Example Fin-Aid dataset for the Micro Course -- Tutorial 3 03-06-92
Description: Example Financial Aid dataset for the Micro Course -- Tutorial 3

Dataset item menu                          Active dataset: "Fin-Aid"

     1. List files in active set          7.  Edit the dataset description
     2. Pattern matching dataset names    8.  Set scramble, dest & rep swit
     3. DSN (dataset name) change         9.  Redo dataset info
     4. DICtionary file name change      10.  Help
     5. Micro data file name change      11.  Back to master menu (return)
     6. Format file name change
```

A null reply returns to the next highest menu.

```
         Enter command:

Dataset menu                               Active dataset: "Fin-Aid"

     1. List existing datasets             5. Find datasets matching pattern
     2. Activate a set and edit fields     6. Exit menu (go to command mode)
     3. Change/describe dataset info       7. Help
     4. Other/misc commands                8. Stop

         Enter command: other

Miscellaneous menu                         Active dataset: "Fin-Aid"

     1. Transfer sets from another CCID    6. Master cmd trace file change
     2. Permit sets to another CCID        7. Write changes to file
     3. System master signon IDchange      8. Destroy dataset
     4. Runfile name change                9. Help
     5. Local cmd trace file change       10. Back to master menu (return)
```

Transferring a Dataset From One *userid* to Another

To transfer a dataset from one userid to another, you must first sign on to the userid holding the dataset, select the Miscellaneous menu, and permit the dataset to the userid you want to transfer it to. Then you sign on to the userid that is to receive the data, select the Miscellaneous menu, and select option 1, Transfer.

```
        Enter command: t
```

If the set is moved it will not exist on its previous userid. Copy will create a new copy.

```
Should the dataset be Copied or Moved? (C,M) c
Get datasets from which CCID? ilir
```

Many datasets can be copied or moved at once. Here we just make a copy of the Art dataset on ILIR.

```
Enter a pattern or dataset name: art
  Dataset: ART                ART ILIR/01-28-82

Dictionary file: K59F:ART#          Data file: K59F:ART
The dataset is NOT destroyable, it is NOT replaceable

Miscellaneous menu                          Active dataset: "Fin-Aid"

     1. Transfer sets from another CCID     6. Master cmd trace file change
     2. Permit sets to another CCID         7. Write changes to file
     3. System master signon IDchange       8. Destroy dataset
     4. Runfile name change                 9. Help
     5. Local cmd trace file change        10. Back to master menu (return)
```

To look at the new dataset we must activate it and go to the Field menu.

```
        Enter command: B

Dataset menu                                Active dataset: "Fin-Aid"

     1. List existing datasets              5. Find datasets matching pattern
     2. Activate a set and edit fields      6. Exit menu (go to command mode)
     3. Change/describe dataset info        7. Help
     4. Other/misc commands                 8. Stop

        Enter command: a
```

Fin-aid is still active until we activate a new set.

```
Dataset name: art
"K59F:FIN-AID#" has been saved.

Field menu                                  Active dataset: "ART"

     1. Append fields (terse)              6.  Modify/add categories
     2. Insert fields                      7.  Show documentation
     3. Redo a field                       8.  Help
     4. Delete fields                      9.  Back to dataset menu (return)
     5. Change field item
```

We enter the Show menu to look at Art.

```
        Enter command: s

Show menu                                Active dataset: "ART"

     1. Fulldoc on screen                 6.  PF (Print Fulldoc on file
     2. Doc on screen                     7.  PD (Print Doc on file)
     3. Tdoc on screen                    8.  PT (Print Tdoc on file)
     4. DT (Doc Terse)                    9.  Help
     5. TT (Tdoc Terse)                  10.  Back to field menu (return)
```

Entering 'd' gets us '`Doc on screen`'.

```
        Enter command: d
Enter a null reply (return) to show all fields or
Enter a pattern, field range, or field:

---------------------------------------------------------------------------
 F#  Field name      Abbr        Value  Description
---------------------------------------------------------------------------
 F1  YEAR            YR                 YEAR ACQUIRED
 F2  CATALOG#        CAT                CATALOG NUMBER (TIMES 1000)
 F3  MEDIUM          MED                MEDIUM
            6 categories
        PAINTING     PNTG           1   PAINTING
        PRINT        PRNT           2   PRINT
        SCULPTURE    SCLP           3   SCULPTURE
        CERAMIC      CRMC           4   CERAMIC
        GLASS        GLSS           5   GLASS
        DRAWING      DRWG           6   DRAWING
 F4  SCHOOL          SCH                SCHOOL
            5 categories
        FRENCH       FR             1   FRENCH
        AMERICAN     AM             2   AMERICAN
        CHINESE      CHNS           3   CHINESE
        INDIAN       IND            4   INDIAN
        SWISS        SWSS           5   SWISS
 F5  PRICE           COST               PRICE PAID
            1 category
        N/L          NL         99999   NOT LISTED
 F6  SOURCE          SO                 SOURCE OF ACQUISITION
            6 categories
        ARCH         A+D            1   COLLEGE OF ARCHITECTURE & DESIGN
        MIRVISH      MIRV           2   DAVID MIRVISH GALLERY
        LAKESIDE     LAKE           3   LAKESIDE STUDIO
        LANTERN      LNTR           4   LANTERN GALLERY
        PARKER       PARK           5   MARGARET WATSON PARKER COLLEGE
        ROSENBERG    ROSY           6   ROSENBERG & STEIBEL

Field menu                               Active dataset: "ART"

     1. Append fields (terse)             6.  Modify/add categories
     2. Insert fields                     7.  Show documentation
     3. Redo a field                      8.  Help
     4. Delete fields                     9.  Back to dataset menu (return)
     5. Change field item
```

A null reply returns us to the Dataset menu.

```
        Enter command:

Dataset menu                             Active dataset: "ART"
```

```
      1. List existing datasets           5. Find datasets matching pattern
      2. Activate a set and edit fields   6. Exit menu (go to command mode)
      3. Change/describe dataset info     7. Help
      4. Other/misc commands              8. Stop
```

And 'stop' gets us out of the program.

```
        Enter command: stop
 Fri 6 Mar 1992  14:55:07
#Execution terminated   14:55:07  T=1.683
```

Data Entry

Now that the new datasets, Students and Fin-Aid, have been defined, we can enter data into them. Data entry is the process of inserting a value for each field in each new record.

There are five data entry methods available in Micro:

- interactive
- free format
- fixed format
- custom programming
- visual data entry and editing

In this Micro Tutorial we will use the first three methods and briefly discuss the last two.

Interactive

The *interactive* method is the easiest. All you need to do is run ILIR:Micro.Add and let it prompt you for everything.

```
#$Run ILIR:Micro.Add
#Execution begins   15:53:16


 %%% MICRO Data Entry System  %%%

 15:53:17
 MAR  6, 1992

 STDS* Copyright 1988-90 by the University of Michigan.  All rights reserved.   [15 May 90]
```

First, Micro.Add asks some general questions, such as which dataset to use.

```
 MICRO data set name:  students

 Concerning K59F:STUDENTS$
 Data file does not exist.
```

Since we have not yet entered any data into Students, Micro.Add asks for confirmation before creating the Micro data file.

```
 Type "YES" to have it created:  y
```

The records we enter remain in memory until they are written to disk as a group. Micro.Add needs to know how many records to keep in memory before saving them.

```
After how many records should data be saved?  3
```

Micro.Add prompts for each data item using either the field name or abbreviation. The null reply given here means 'no', so Micro.Add will prompt with field names.

```
Abbreviated prompting? (Y,N)
```

Now we can enter the values for the fields in the first record. The Student-ID-Num field is an integer.

```
Student-ID-Num  : 13222340
```

The Name field is next. Primes are needed if a data value contains blanks.

```
Name            : 'Richardson Deborah'
```

We can use category names, abbreviations, or values for the Ethnicity and Gender fields.

```
Ethnicity       : black
Gender          : female
```

The Birthdate is in YYMMDD format.

```
Birthdate       : 550310
```

The Class-year field is a 2 digit integer (class of 77).

```
Class-Year      : 77
```

When we have entered values for all of the fields, Micro.Add goes into *Corrections mode.* At this point any of the following actions are possible:

- A null reply accepts the current record and begins the next record.
- A correction, in the form *field = value*, makes the change and goes back to Corrections.
- "Save" accepts the current record, saves this group of records to disk, and begins the next record.
- "Stop" accepts the current record, saves this group of records to disk, and stops the program.

A null reply begins the next record.

```
%Corrections, "STOP", or "SAVE":
%
 Student-ID-Num  : 095887328
 Name            : 'Pond Stuart Leslie'
```

This is what happens if Micro.Add detects an error. If you're not sure what to do, type a '?'. Micro.Add will display documentation for the current field.

```
 Ethnicity      : 20

 Concerning 20 for field Ethnicity
 Value must be a category name or category value.
 Enter a new value or return to skip this field:  ?


 -------------------------------------------------------------------------
 F(#)   FIELD NAME       ABBR       VALUE  DESCRIPTION
 -------------------------------------------------------------------------


 F(1)   Ethnicity        Race              Ethnicity
           CATEGORIES (ONLY)
            Unknown      Unk          0    Unknown
            White        Wh           1    White
            Black        Bl           2    Black
            Amer-Indian  AI           3    American Indian
            Spanish      Sp           4    Spanish-Surnamed American
            Oriental     Or           5    Oriental
            Other        Oth          6    Other
```

So we enter a value that Micro.Add will allow, and then complete the record.

```
 Enter a new value or return to skip this field:  white
 Gender          : male
 Birthdate       : 560528
 Class-Year      : 78
%Corrections, "STOP", or "SAVE":
%
```

It is possible to enter more than one field on the same line, anticipating the field prompts. Indeed, for datasets with few fields, the whole record can fit on one line.

```
 Student-ID-Num  : 113591707 'Jones Bill' wh male 551010 77
```

Here we make a correction before continuing.

```
%Corrections, "STOP", or "SAVE":
%name = 'Jones William'
%Corrections, "STOP", or "SAVE":
%

  3 records added.
  3 records in permanent set.
```

Micro.Add automatically saves this group of three records, as requested earlier, before beginning the next record.

```
 Student-ID-Num  : 122045525
 Name            : 'Grace Kenneth R'
 Ethnicity       : wh
```

What happens when you enter a null reply for a data item? If data is required for the field, Micro.Add just prompts again. Otherwise, it uses the default value for the field.

```
 Gender          :
```

A common mistake is to type 'o' instead of '0'.

```
Birthdate       : 58o418

Concerning 58o418 for field Birthdate
Value must be an integer or category name.
Enter a new value or return to skip this field:  580418
Class-Year      : 80
```

Corrections can also be in the form *field* : *value*.

```
%Corrections, "STOP", or "SAVE":
%sex: male
```

We're done for now. Micro.Add saves the current group of records (just one here) and stops.

```
%Corrections, "STOP", or "SAVE":
%stop

  1 records added.
  4 records in permanent set.

 Summary:
      4  records are being added
      4  records in permanent set
#Execution terminated   15:56:33  T=0.949
```

The interactive method is useful for entering a handful of records. It is also useful for training in the use of the free format method. Prompting for each field and providing immediate error correction allows you to learn about the fields in the dataset.

Free Format

The *free format* method is similar to the interactive but, instead of typing the data in response to the prompts, you enter the data into a file in such a way as to anticipate all of the prompting.

Here is a listing of an existing input file containing some sample free format data to be entered into the Students dataset.

```
#$List Students.Add

      1       133680736 'Lee Sue Jean' oriental fe 560716 78 /e
      2       139904799 'Bailey Edith' black fe 570913 77 /e
      3       142668782 'Cook Cynthia Anna' black fe 560908 78 /e
      4       149514226 'Scott Mary Grace' white fe 561230 78 /e
      5       173049236 'Crosby Jane Lillian' white fe 570510 79 /e
      6       179410240 'Taylor Vera M' black fe 501004 78 /e
      7       181703558 'Hale Robert Samuel' white ma 551011 77 /e
      8       186549615 'Cunningham Sarah' white fe 570805 79 /e
      9       199472144 'Almondizer Karen S' white fe 580215 80 /e
     10       200914081 Schiller Esther' white fe 570728 79 /e
```

The '/e's in the file signal the end of each record. These are needed since the data for a record can span several lines in the file.

In the next $Run command, Students.Add is the input file. Since this method is not interactive, errors can't be fixed immediately. If a record contains an error, the entire record is placed into the error file, –ERR1. The 'par=Freeform' tells Micro.Add that we're doing free format entry.

```
#$Run ILIR:Micro.Add input=Students.Add spunch=-Err1 par=Freeform
#Execution begins   15:57:53


 %%%  MICRO Data Entry System  %%%

 15:57:53
 MAR  6, 1992

 STDS* Copyright 1988-90 by the University of Michigan.  All rights reserved.   [15 May 90]
```

Micro.Add only needs to know what dataset to use.

```
 MICRO data set name:  students

 Concerning Esther' for field Ethnicity
 Value must be a category name or category value.
 Error occurred on line       10.000
 Record began on line         10.000
      10.000  200914081 Schiller Esther' white fe 570728 79 /e

  9 records added.
  13 records in permanent set.

 Summary:
      10  records were read in
       1  records were rejected
       0  records were duplicates
       9  records are being added
      13  records in permanent set
#Execution terminated   15:58:02  T=0.567
```

We can fix error records by editing the error file ...

```
#$Edit -Err1
:print /f
:    10       200914081 Schiller Esther' white fe 570728 79 /e
:alter  ;S;'S;
:    10       200914081 'Schiller Esther' white fe 570728 79 /e
:stop
```

...and then using the error file as the input file. Notice that if the dataset name is given in the $Run command par field, Micro.Add will not ask you for it.

```
#$Run ILIR:Micro.Add input=-Err1 spunch=-Err2 par=Freeform DSN=Students
#Execution begins   15:58:21


 %%%  MICRO Data Entry System  %%%

 15:58:21
 MAR  6, 1992

 STDS* Copyright 1988-90 by the University of Michigan.  All rights reserved.   [15 May 90]
```

```
  1 records added.
  14 records in permanent set.

 Summary:
      1  records were read in
      0  records were rejected
      0  records were duplicates
      1  records are being added
     14  records in permanent set
#Execution terminated   15:58:25  T=0.364
```

The free format method is best for entering lots of records that are not already machine readable (i.e., someone must type it in). You can use the MTS Editor to type the data into a file. If you have access to a microcomputer, you can enter data using a text-editor or word-processor (if it can generate a text-only file), then transfer the file to MTS.

Enterdata Command

The interactive and free format methods are available in ILIR:Micro via the Enterdata command. This is useful if you will be working with the data immediately after entering it.

```
#$Run ILIR:Micro
#Execution begins   15:59:03
 ILIR:Micro (15 Dec 91)
  University of Michigan
 Fri 6 Mar 1992  15:59:04

  3 datasets from Directory
```

Here is a listing of an input file containing some sample free format data to be entered into the Fin-Aid dataset.

```
 Ready:
-$List Fin-Aid.Add

      1       113591707 74 F SR TU UNIV 1200.00 '' /E
      2       123456789 75 F SO TU UNIV 235.00 '' /E
      3       133680736 74 F JR RB FGOV 485.00 'Outstanding senior' /E
      4       142668782 74 F JR TU IND 1050.00
      5         'Award from International Business Systems of Armonk, New York' /E
      6       149514226 75 W JR CA IND 100.00
      7         'Grant from the Oily Petroleum Co. to study oil slicks' /E
      8       173049236 75 F SO OTH UNIV 100.00 'Part-time employment grading papers' /E
      9       179410240 75 S JR TU UNIV 540.00 '' /E
     10       229644658 75 F SO WS UNIV 400.00 'One time award' /E
     11       251760359 73 F SR RB UNIV 500.00 '' /E
     12       259964164 76 F FR WS SGOV 150.00 'Work in office of Secretary of State' /E
     13       281482860 75 S JR WS FGOV 400.00
     14         'Federal work/study with the Department of Labor' /E
     15       304789983 75 F SO CA FGOV 128.00 'Outstanding senior' /E
     16       360984224 77 W SO RB UNIV 175.00 'One time award' /E
     17       408823168 76 F SO LO UNIV 225.00 '' /E
     18       409142875 75 W JR TU FGOV 550.00 '' /E
     19       463898163 74 S SR TU UNIV 125.00 '' /E
     20       480780530 76 F FR CA SGOV 375.00 'Governor''s Award' /E
     21       480780530 76 W FR CA SGOV 375.00 'Governor''s Award' /E
     22       498930942 73 F SR LO SGOV 820.00 '' /E
```

```
23      521491463 76 F SO OTH UNIV 200.00 'Part-time employment grading papers' /E
24      521690114 75 W JR RB SGOV 200.00 '' /E
25      556052771 74 F JR TU IND 150.00 'Scholarship from Amalgamated Industries,-
26       Inc.' /E
27      686850331 74 W SR RB UNIV 525.00 '' /E
28      733267655 74 W SR CA SGOV 50.00 'State grant to study automobile industry' /E
29      741997403 74 F JR TU UNIV 350.00 '' /E
30      787195848 75 F SO CA UNIV 75.00 '' /E
31      796731703 76 S SO TU SGOV 295.00 'Regents'' Award' /E
32      805219718 77 W FR LO UNIV 150.00 '' /E
33      854351704 76 F FR RB UNIV 250.00 '' /E
34      906922815 73 F JR RB SGOV 125.00 '' /E
35      913222340 74 W SR WS FGOV 350.00 'Outstanding senior' /E
36      945031192 74 W JR CA IND 325.00
37        'Grant from Belt Telephone Co. to study telepathy' /E
38      949705539 76 W FR RB SGOV 280.00 '' /E
39      1399094799 73 F SR LO FGOV 180.00 '' /E
```

As in Micro.Add, primes are needed around data items that contain blanks. Two primes together '''' indicate a null string. If a prime occurs in the text, as in line 31, it must be doubled.

When a data value is too long to fit on a line, as in line 25, enter a hyphen followed immediately by a return, and then continue the data item on the next line. This is better handled by starting a long data item on a new line, as in lines 36 and 37.

In the next Enterdata command, the phrase '`from Fin-Aid.Add with errors on -err3`' tells Micro to use the free format method: Fin-Aid.Add is the input file and –err3 is the error file. If the input file is omitted, Micro uses the interactive method.

The '`perm`' keyword tells Micro to enter the records permanently into Fin-Aid. If it is omitted, Micro puts the existing records plus the new records into the Result dataset.

```
 Ready:
-Enter perm into Fin-Aid from Fin-Aid.Add with errors on -err3
 Micro data file "K59F:FIN-AID$" doesn't exist.

 Concerning K59F:FIN-AID$
 Micro data file does not exist.
```

Since we have not yet entered any data into Fin-Aid, Micro asks for confirmation when creating the Micro data file and the external data file.

```
 Type "YES" to have it created:  y

 Concerning K59F:REMARK* for field Remark*
 External data file does not exist.
 Type "YES" to have it created:  y

  34 records added.
  34 records in permanent set.

 Summary:
      34  records were read in
       0  records were rejected
       0  records were duplicates
      34  records are being added
      34  records in permanent set
```

Here are a few of the new records from our datasets.

```
 Ready:
-Print in Fin-Aid count
  34 records in Fin-Aid

 Ready:
-Print@abbr@count=10 Fin-Aid

         SIN Year Term   CL Type  SOF          Amt  Rem*
--------------------------------------------------------------------------
   113591707   74    F   Sr   Tu Univ      1200.00
   123456789   75    F   So   Tu Univ       235.00
   133680736   74    F   Jr   Rb FGov       485.00  Outstanding senior
   142668782   74    F   Jr   Tu  Ind      1050.00  Award from
                                                    International
                                                    Business Systems of
                                                    Armonk, New York
   149514226   75    W   Jr   Ca  Ind       100.00  Grant from the Oily
                                                    Petroleum Co. to
                                                    study oil slicks
   173049236   75    F   So  Oth Univ       100.00  Part-time employment
                                                    grading papers
   179410240   75    S   Jr   Tu Univ       540.00
   229644658   75    F   So   Ws Univ       400.00  One time award
   251760359   73    F   Sr   Rb Univ       500.00
   259964164   76    F   Fr   Ws SGov       150.00  Work in office of
                                                    Secretary of State

 Ready:
-Print in Students count
  14 records in Students

 Ready:
-Print@abbr@count=10 Students

         SIN  Name                    Race  Sex     Bday ClYr
-------------------------------------------------------------
    13222340  Richardson Deborah       Bl   Fe    550310   77
    95887328  Pond Stuart Leslie       Wh   Ma    560528   78
   113591707  Jones William            Wh   Ma    551010   77
   122045525  Grace Kenneth R          Wh   Ma    580418   80
   133680736  Lee Sue Jean             Or   Fe    560716   78
   139904799  Bailey Edith             Bl   Fe    570913   77
   142668782  Cook Cynthia Anna        Bl   Fe    560908   78
   149514226  Scott Mary Grace         Wh   Fe    561230   78
   173049236  Crosby Jane Lillian      Wh   Fe    570510   79
   179410240  Taylor Vera M            Bl   Fe    501004   78

 Ready:
-Stop
   $.01,  $.75M ($.26S),  $5.02T
 Fri 6 Mar 1992  16:00:28
#Execution terminated   16:00:28  T=0.795
```

Fixed Format

The *fixed format* method allows you to read data that is formatted in column ranges. Data formatted in this way is usually generated by a computer program or by keypunching. Large amounts of such data are often stored on tape.

Here are some records from a fixed format input file with a "ruler" to indicate the columns.

```
#$List Ruler+Students.Cvt(1,*L,10)

     1            10        20        30        40        50        60        70        80
     2      ....,....|....,....|....,....|....,....|....,....|....,....|....,....|....,....|
     3
     1      207920096Conners Suzanne L                1F571013 79
    11      264872649Foley Donald C                   2M570210 79
    21      430150047Fitzgerald Joan                  2F581015 80
    31      514215872Swift Sally Jane                 2F581130 80
    41      613835160Knapp Frederick Harold           1M550118 77
    51      752155359Erickson Sheila Mary             3F581010 -1
    61      816115339Gould Harry Edward               2M560313 78
    71      967397398Mendez Jose Roberto              4M560708 78
```

Micro's fixed format program must know the format of the input file, so a *conversion format file* must be built before the data is actually entered. This is done via the interactive program ILIR:Micro.Form.

```
#$Run ILIR:Micro.Form
#Execution begins   16:01:21


 %%%  MICRO Data Entry System  %%%

 16:01:22
 MAR  6, 1992
```

First, Micro.Form asks some general questions, such as which dataset to use.

```
MICRO data set name:  Students
```

It also needs to know the name of the conversion format file (unless one was specified when the dataset was defined).

```
 Conversion format file name:  Students%
 Conversion format file STUDENTS% does not exist.
 Type "YES" to have it created:  y
```

Micro.Form next asks for information about the input file as a whole. There are 55 columns per record in this input file. The term *card* here refers to one line in the input file.

```
 Maximum total number of columns on all cards:  55
```

There is only one record per line.

```
 Maximum number of cards in one record:  1
```

A null reply to the next question indicates that we want to enter all of the records.

```
 Do you wish to include or exclude only certain records ("E" or "I" or RETURN to ignore)?
```

Another null reply disables the record sampling facility.

```
Record sampling string of zeros and ones (0=skip 1=keep) or RETURN to ignore:
```

```
field Student-ID-Num
```

Now Micro.Form asks about the position and content of each field in an input record. Often this information is specified in a format document accompanying a tape or diskette that contains the input file. If not, you must inspect the input file.

Micro.Form displays the field name for each field, then asks for the *starting column*. This is the column in the input record where the data value for this field begins. The Student-ID-Num value starts in column 1. Note: a discussion of *subroutine file* is beyond the scope of this tutorial.

```
Starting column (or subroutine file):  1
```

The *input type* is the general format of the data value in the input record. Here '`i`' means integer.

```
Input type:  i
```

The *length* is the number of columns that the data value spans in the input record. The Student-ID-Num value occupies 9 columns.

```
Length:  9
```

The *missing data value* is the value to use if the column range for this field in the input record contains all blanks or hyphens. By default this is zero for a numeric field and blanks for a character field. Most folks give a null reply here.

```
Missing data value:
```

The *recode type* allows you to modify data during data entry. It is discussed below. A null reply means no recode.

```
Recode type:
```

```
field Name
```

On to the Name field, a 24 character name starting in column 10. Note: for a character field, the Input type is assumed to be character.

```
Starting column (or subroutine file):  10
Character input assumed.
Length:  24
Missing data value:
```

```
Recode type:
```

```
field Ethnicity
```

The Ethnicity field is a single digit integer in column 44 of the input record. We have a recode for this field. You can read this recode to mean "for input values greater than 6, enter 0". Micro.Form will ask for more recodes if there are more possible input values.

```
Starting column (or subroutine file):  44
Input type:  i
Length:  1
Missing data value:

Recode type:  gt
Input value:  6
New value:  0

Recode type:

field Gender
```

The Gender field is a 1 character value in column 45.

```
Starting column (or subroutine file):  45
Character input assumed.
Length:  1
Missing data value:

Recode type:

field Birthdate
```

The Birthdate field is a 6 digit integer starting in column 46. We have made a deliberate error (the length) to be fixed later.

```
Starting column (or subroutine file):  46
Input type:  i
Length:  4
Missing data value:

Recode type:

field Class-Year
```

The Class-year field is a 2 digit integer starting in column 53. If we make a mistake and enter a starting column larger than the maximum, Micro.Form will detect this error and ask for a replacement. The recode here will translate all negative input values to -1.

```
Starting column (or subroutine file):  56

Concerning 56 for field Class-Year input column
Value exceeds number of columns on a card.
Maximum column number is       55
Starting column (or subroutine file):  53
Input type:  i
Length:  2
Missing data value:

Recode type:  lt 0
New value:  -1

Recode type:
The conversion format is complete.
```

At this point Micro.Form has all of the information it needs, so it goes into *Corrections mode.* You can see what the conversion information looks like using the Document command.

```
%Corrections:
%doc

 NUMBER OF CARDS =           1
 COLUMNS PER CARD =          55
 TOTAL INPUT RECORD LENGTH =          55

 FIELD           TYPE CARD COLUMN LENGTH   MISSING-DATA
 -----------------------------------------------------

 Student-ID-Num    I    1      1      9             0

 Name              C    1     10     24           ' '

 Ethnicity         I    1     44      1             0
 RECODES....
   1          7 THROUGH   2147483647 BECOMES              0

 Gender            C    1     45      1           ' '

 Birthdate         I    1     46      4             0

 Class-Year        I    1     53      2             0
 RECODES....
   1  -2147483648 THROUGH           -1 BECOMES           -1
```

Using the Redo command we can correct the error made in the Birthdate field format (there is no command to change individual items).

```
%Corrections:
%redo birthdate

 field Birthdate
 Starting column (or subroutine file):  46
 Input type:  i
 Length:  6
 Missing data value:
```

You can add recodes for a field using the Recode command.

```
%Corrections:
%recode birthdate

 Recode type:  lt 0
 New value:  0

 Recode type:
```

Then Show the field to see the new recode.

```
%Corrections:
%show birthdate

 FIELD           TYPE CARD COLUMN LENGTH   MISSING-DATA
 -----------------------------------------------------

 Birthdate         I    1     46      6             0
 RECODES....
   1  -2147483648 THROUGH           -1 BECOMES            0
```

The Stop command saves the conversion format file and terminates Micro.Form.

```
%Corrections:
%stop
#Execution terminated   16:05:59  T=0.412
```

To enter the records, run ILIR:Micro.Cnvrt. The $Run command is similar to that for the free format method using Micro.Add. Students.Cvt is the input file, –err4 is the error file, and Students is the dataset name. The conversion format file is Students%.

```
#$Run ILIR:Micro.Cnvrt input=Students.Cvt spunch=-err4 par=DSN=Students CFF=Students%
#Execution begins   16:06:22


 %%% MICRO Data Entry System  %%%
 16:06:23
 MAR  6, 1992

 *** Conversion begins ***
 *** Conversion completed -- converted data is in -RESTDS ***

 Loading STDS -- please do not interrupt.
 *** STDS operations begin ***
 STDS* Copyright 1988-89 by the University of Michigan.  All rights reserved.   [08/31/89]
 *** STDS operations completed -- old + new data is in -LMIS ***
              71 records read (from SCARDS)
 -             0 records rejected (to SPUNCH)
 =            71 records converted
 +            14 records from the original data set
 -             0 records weeded (duplicated)
 =            85 records in the final data set
#Execution terminated   16:06:30  T=0.683
```

There are more features available in the fixed format method; we have seen only those that are most heavily used. Micro.Form and Micro.Cnvrt are fully documented in the reference manual.

Custom Programming

The *custom programming* method is the original data entry method. As the name implies, you write your own programs tailored to your data entry needs. You can write programs to be efficient, friendly, "smart", or whatever your application requires. They must produce files that can be read using one of the methods discussed below or using the Micro Read Array command.

The custom programming method gives you complete control over data entry, so you can take advantage of available computing resources. For example, on MTS you can use the MTS Screen Support routines to build custom data entry screens and menus. On a microcomputer you can use database programs, spreadsheets, or other tools to do data entry off-line and then transfer the files to MTS.

Custom programming is best suited for use with a well-established database that will not require frequent structural changes (e.g., adding new fields). Such changes will probably require changes to the programs. Since it can require a considerable programming effort, you should only consider custom programming if the benefits are worth it.

Visual Data Entry and Editing

Visual data entry and editing are provided through ILIR:MicEdit. You can display and edit existing records and enter new records in visual (full-screen) mode. You can let MicEdit use a default screen layout or you can design you own. MicEdit is fully documented in Part I of the Micro manual.

Miscellaneous Topics

This is the fifth Micro Tutorial. We will cover the following topics:

- Updating
- Command files and Runfiles
- Micro Programming Facility

Updating

Updating involves making changes to existing records for any fields in a dataset using any fields to identify which records to change.

Interactive

The *interactive* method is the easiest. All you need to do is run ILIR:Micro.Up and let it prompt you for everything. In this example we use SIN (Student-ID-Num) as the *identifying field*, the field used to identify which records to update in Students. The Name and Birthdate are the *update fields*, the fields to be updated.

```
#$Run ILIR:Micro.Up
#Execution begins   16:12:30


   @@@  MICRO Update System  @@@
 16:12:31
 MAR  6, 1992
 STDS* Copyright 1988-90 by the University of Michigan.  All rights reserved.   [15 May 90]

 MICRO data set name:  students
 Names of fields used to identify records:  sin
 Names of fields to be updated (or "STOP"):  name birthdate
 Abbreviated prompting? (Y,N) y
 SIN  : 854351704
 Name : 'Rosen Lawrence Z' 580510
%Corrections, "STOP", or "SAVE":
%stop

  1 records added.
 Names of fields to be updated (or "STOP"):  stop
#Execution terminated   16:12:35  T=0.199
```

Notice that this is similar to the interactive method of data entry in the fourth Tutorial. Indeed, the same program is used to enter the data, but only for the identifying and update fields.

Free format

The *free format* method reads the data corresponding to the interactive prompts from a file, such as this one to update the Students dataset.

```
#$List Students.Up

     1      139904799 570914 /e
     2      173049236 570410 /e
     3      186549615 570508 /e
     4      210999051 560809 /e
     5      281482860 560112 /e
```

In this example we update the Birthdate field in Students using SIN to identify the records.

```
#$Run ILIR:Micro.Up  input=Students.Up  spunch=-Err1  par=Freeform
#Execution begins   16:13:14


   @@@  MICRO Update System  @@@
 16:13:15
 MAR  6, 1992
 STDS* Copyright 1988-90 by the University of Michigan.  All rights reserved.   [15 May 90]

 MICRO data set name:  students
 Names of fields used to identify records:  sin
 Names of fields to be updated (or "STOP"):  birthdate

  5 records added.

 Summary:
       5  records were read in
       0  records were rejected
       0  records were duplicates
       5  records are being added
#Execution terminated   16:13:19  T=0.294
```

Notice that this is similar to the free format method of data entry in the fourth Tutorial.

Update Command

The interactive and free format methods are available in ILIR:Micro via the Update command.

```
#$Run ILIR:Micro
#Execution begins   16:13:36
 ILIR:Micro (15 Dec 91)
  University of Michigan
 Fri 6 Mar 1992  16:13:37

  3 datasets from Directory
 Ready:
-Update Fin-Aid  perm  id=SOF  update=SOF
 Abbreviated prompting? (Y,N) y
 SOF  : federal-gov
 SOF  : state-gov
%Corrections, "STOP", or "SAVE":
%stop

  1 records added.

  1 update record entered
  6 records updated
 Fin-Aid has been updated.
```

The phrase '`id=sof`' tells Micro to use SOF (Source-of-Funds) as the identifying field. Multiple identifying fields must be separated by commas. If this phrase is omitted, Micro asks for the identifying fields.

The phrase '`update=sof`' tells Micro to update SOF (it is possible to update an identifying field). Multiple update fields must be separated by commas. If this phrase is omitted, Micro asks for the update fields.

The '`perm`' keyword tells Micro to update the records permanently in Fin-Aid. If it is omitted, Micro generates a Result dataset containing the records with the updates applied.

If a phrase such as '`from Fin-Aid.Up with errors on -err3`' were given, it would tell Micro to use the Free format method, where Fin-Aid.Up is the input file and –err3 is the error file. Since this is omitted, Micro uses the interactive method.

Runfiles

It is often useful to execute a sequence of commands just after getting into Micro. Suppose that, among other things, we want to always suppress the '`Ready:`' prompt so that Micro only prompts with the hyphen. Of course we could type Settings commands each time we run Micro, but that would be tedious. Fortunately, there's a better way. You can build a *runfile*, a file that contains Micro commands to be executed at the start of each Micro run. Here's a sample runfile.

```
 Ready:
-$List Runfile

      1      Set  Command: Ready off;  -
      2             Itab: % off, Print off
      3      Message  "0" ; -
      4      "0--- Current settings ---" ; -
      5      "0 Command: Ready off" ;  -
      6      "     Itab: % off, Print off" ;  -
      7      " "
```

It is not necessary to use the name RUNFILE but most people do. The following command tells Micro to use RUNFILE as a runfile the next time we run Micro.

```
 Ready:
-Set  User: Runfile = "Runfile"
```

```
 Ready:
-stop
   $.00,  $.69M ($.24S),  $10.87T
 Fri 6 Mar 1992  16:15:11
#Execution terminated   16:15:11  T=0.686
```

Now when we run Micro again, the runfile will execute automatically.

```
#$Run ILIR:Micro
#Execution begins   16:15:24
 ILIR:Micro (15 Dec 91)
  University of Michigan
 Fri 6 Mar 1992  16:15:25

  3 datasets from Directory


 --- Current settings ---

  Command: Ready off
     Xtab: % off, Print off
```

You must build the runfile using a text editor. Although you can put any Micro commands into a runfile, the Settings and Message commands are used most often to customize settings and to display messages.

Command Files

A runfile is a special case of a *command file*. You can put any sequence of Micro commands into a file and use it as a command file. This is a simple way to automate procedures that are performed often. Here is an example command file.

```
-$List CalcAge.Cmd

      1      *   This Micro command file calculates a new Age field for each record in
      2      *   the Students dataset from the Birthdate field.
      3      Calculate in Students Age@scale=0 = int((date() - from_yymmdd(Birthdate)) / 365.25)
      4      Release *Problems*
```

To invoke a command file, use the Settings command.

```
-Set  Cmd: Input = "CalcAge.Cmd"
```

This Micro command file calculates a new Age field for each record in the Students dataset from the Birthdate field.

```
-Calculate in Students Age@scale=0 = int((date() - from_yymmdd(Birthdate)) / 365.25)
  85 records in RESULT
  0 records in *Problems*
-Release *Problems*
 End of command file.
```

Let's check its work.

```
-P@Count=10 in It SIN, Birthdate@date, Age

 Student-ID-Num  Birthdate        Age
----------------------------------------
       13222340   03-10-55         36
       95887328   05-28-56         35
      113591707   10-10-55         36
      122045525   04-18-58         33
      133680736   07-16-56         35
      139904799   09-14-57         34
      142668782   09-08-56         35
      149514226   12-30-56         35
      173049236   04-10-57         34
      179410240   10-04-50         41

-stop
  $.00,  $.39M ($.14S),  $11.69T
 Fri 6 Mar 1992  16:16:02
#Execution terminated   16:16:01  T=0.418
```

Micro Programming Facility

The commands in CALCAGE.CMD work as long as the dataset is Students and the field is Birthdate. Of course we can modify CALCAGE.CMD to use other datasets and fields, but it would be better to specify the dataset and field without having to change the file. In other words, we would like to treat the dataset and field names as parameters or arguments that can be passed to the Calculate command.

The Micro Programming Facility (MPF) makes this possible. It allows us to write programs that can be used just like a Micro command. Such programs, called *MPF Micro commands*, can be invoked at the Micro command prompt. The first word of the command is the name of the program; the remaining words are the parameters.

MPF programs can generate and execute Micro commands and MPF Micro commands. The parameters can be manipulated and inserted into the generated commands. So we can do the age calculation for any dataset using any birthdate field. Here is an example.

```
#$List CalcAge.MPF

     /*
     ------------------------------------------------------------------------------

                                  CalcAge

          This MPF program generates a Result dataset containing a new field,
          Age, generated from a birthdate field.  The syntax is

               CALCAGE  dataset  [birthdate_field]

          dataset is the name of the dataset containing the birthdate_field.
          If omitted, CalcAge will prompt for it.  If dataset = "?", CalcAge
          displays a usage line.

          birthdate_field is the name of the birthdate field from which to
          derive the age.  It must be a numeric field containing dates in
```

```
17          yymmdd or yyyymmdd format (e.g., 910228 or 19411207).  If omitted,
18          it defaults to "Birthdate".
19
20     ----------------------------------------------------------------------------
21     */
22
23     micro  CalcAge  -
24     calc_age(char *Dataset, char *BDay=Birthdate) {
25          /*
26               Dataset        name of the dataset
27               BDay           (optional) name of the birthdate field
28          */
29
30          if ( ! strcmp (Dataset, "?") ) {
31               printf ("Usage:\n"
32                       "  CALCAGE  dataset  [birthdate_field]\n") ;
33          }
34          else {
35               calculate in {Dataset}
36               :    Age@scale=0 = int((date() - from_yymmdd({BDay})) / 365.25)
37                        execute
38          }
39     }
```

Everything between the '/*' and the '*/' are comments. On line 23 '`micro`' declares this as a MPF Micro command; it can be invoked from the Micro command prompt. '`CalcAge`' is the MPF Micro command name, the name used at the Micro command prompt. The hyphen says that the declaration will continue on the next line. '`calc_age`' is the C name of the function. The parameters are the name of the dataset, '`Dataset`', and the name of the birthdate field, '`BDay`'. '`=Birthdate`' is the default value to be used if the BDay parameter is omitted on the command line.

If you type '`CalcAge ?`' at the Micro prompt, CalcAge displays a brief usage line showing what parameters it expects. Otherwise it substitutes the Dataset and BDay parameters into the Calculate command on line 35 and executes it.

Before we can use CalcAge, we must compile it. This involves running the MPF Preprocessor (ILIR:MPF), the C compiler (*C87), and the object editor (*ObjUtil). Fortunately, there is an MTS macro (in ILIR:MPF.MAC) that automates this process. First we activate the macro (MTS macros are discussed in MTS Volume 21). Then we can use it to compile MPF programs.

```
#$Source ILIR:MPF.Mac
#MPF CalcAge
  1 Micro routine saved in K59F:MPFLib
  Function calc_age (254 code, 172 const, 2 grs, 0 frs, 96 stack)
  Total code and constants size 428
  No compilation errors.
 Creating the object library, K59F:MPF.Lib.
 OBJUTIL VERSION(CT311) 16:18:38 03-06-92
 ADDED:
  CALC_AGE
```

'`MPF`' is the name of the macro and '`CalcAge`' is the name of the file (without the '`.MPF`' extension) containing the MPF source code.

The MPF macro inserts an entry for the CalcAge program into the *MPF library file.* It contains the information necessary for Micro to recognize "CalcAge" as a MPF Micro command and to handle its parameters. The default MPF library name is "MPFLIB". The MPF macro also inserts the MPF program itself (the object code) into the *MPF object library file.* The default MPF object library name is "MPF.LIB".

Let's run Micro and try it out.

```
#$Run ILIR:Micro
#Execution begins   16:19:18
 ILIR:Micro (15 Dec 91)
  University of Michigan
 Fri 6 Mar 1992  16:19:19

  3 datasets from Directory


 --- Current settings ---

  Command: Ready off
     Xtab: % off, Print off
```

Here's the usage information.

```
-CalcAge ?
 Usage:
    CALCAGE  dataset  [birthdate_field]
```

First we try it using the Birthdate field in Students.

```
-CalcAge Students Birthdate
  85 records in RESULT
  0 records in *Problems*
```

This time we let it assume the default Birthdate field.

```
-CalcAge Students
  85 records in RESULT
  0 records in *Problems*
```

Here are the results.

```
-Sort it by Birthdate
  85 records in RESULT

-Print@c=10 in it Birthdate@date, Age

    Birthdate         Age
 -------------------------
     01-11-47          45
     11-05-48          43
     10-04-50          41
     12-05-50          41
     06-08-52          39
     01-02-55          37
     01-18-55          37
     01-23-55          37
     03-10-55          36
     03-14-55          36

-Stop
   $.00,  $.53M ($.18S),  $13.78T
 Fri 6 Mar 1992  16:19:30
#Execution terminated   16:19:30  T=0.584
```

The source code for another MPF program is in ILIR:DEMO.MPF. This is a more complex program that demonstrates most of the features of the MPF. First we compile it using the MPF macro.

```
#MPF ILIR:Demo
  6 Micro routines saved in K59F:MPFLib
  Function echocmd (46 code, 32 const, 2 grs, 0 frs, 72 stack)
  Function printcnt (132 code, 44 const, 2 grs, 0 frs, 96 stack)
  Function dsinfo (1456 code, 737 const, 3 grs, 0 frs, 2408 stack)
  Function printpars (98 code, 65 const, 2 grs, 0 frs, 96 stack)
  Function pardemo (706 code, 574 const, 2 grs, 0 frs, 80 stack)
  Function mpfdemo (1048 code, 784 const, 2 grs, 0 frs, 208 stack)
  Function Handler (60 code, 8 const, 2 grs, 0 frs, 72 stack)
  Total code and constants size 5812
  No compilation errors.
 OBJUTIL VERSION(CT311) 16:19:48 03-06-92
 ADDED:
  ECHOCMD  PRINTCNT DSINFO   PRINTPAR PARDEMO  MPFDEMO  HANDLER

 CPU time = 0.06 seconds.
```

Then we run Micro to try it out. We will list parts of the source code, execute the programs, and discuss them as we go.

```
#$Run ILIR:Micro
#Execution begins   16:20:06
 ILIR:Micro (15 Dec 91)
  University of Michigan
 Fri 6 Mar 1992  16:20:07

  3 datasets from Directory


 --- Current settings ---

  Command: Ready off
     Xtab: % off, Print off

-$List ILIR:Demo.MPF(53,68)

     53      /*  Note:  ilir:mpf.h is automatically included by the MPF preprocessor  */
     54
     55                                          /*  things for printf formats:  */
     56      #define  NL           "\n"          /*  newline string  */
     57      #define  InQuotes(s)  "\"" #s "\""  /*  usually used as InQuotes(%s)  */
     58
     59
     60      /*
     61      -------------------------------------------------------------------------
     62           Echo the command string (contained in the global buffer _microcmd).
     63      -------------------------------------------------------------------------
     64      */
     65      void  echocmd (void) {
     66
     67           printf ("    EchoCmd:  %s" NL, _microcmd) ;
     68      }
```

Lines 1 through 52 in ILIR:Demo.MPF are comments. '`NL`' is used in `printf` statements to generate a newline. '`InQuotes`' is a convenient way to generate quoted strings in `printf` statements.

'echocmd' is a regular C function. It displays (echos) a Micro command that has been generated. We will use it later.

```
-$List ILIR:Demo.MPF(71,91)
```

```
/*
---------------------------------[ Count ]-----------------------------------

     Count prints the number of records in a dataset.

          Usage:    COUNT dataset

     This is a simple example of building a Micro command.

     Notes:
     -  The '-' after "Count" is the continuation character.
     -  I indent the "execute"s only for readability.

-----------------------------------------------------------------------------
*/
micro  Count  -
printcnt(char *Dataset) {

     Print in {Dataset} Count
               execute
}
```

'Count' is a MPF Micro command. It prints the number of records in a dataset.

```
-Count Fin-Aid
  34 records in Fin-Aid
```

Micro Interface Functions

The MPF Micro command 'DSinfo' prints information about a dataset, similar to that produced by the Micro Document command, by using many of the *Micro Interface Functions*.

```
-$List ILIR:Demo.MPF(94,129)
```

```
/*
---------------------------------[ DSinfo ]----------------------------------

     DSinfo prints some dataset information to stdout.

          Usage:    DSINFO dataset

     This demonstrates various functions in the MPF Library.

-----------------------------------------------------------------------------
*/
micro  DSinfo  -
dsinfo(char *Dataset) {

     /*
          The "magic numbers", special types, and functions used here are
          defined or declared in the header files.
     */

     DATASET        dse ;
     FIELD          fld ;
     CATEGORY       cat ;
```

```
116         FIELD_TYPE    type ;
117         int           ncats, dict_fdnb ;
118         int           hadattn = 0 ;
119         char
120             dsname[DSNAME_SIZE],         extfile[FILENAME_SIZE],
121             dictfile[FILENAME_SIZE],     datafile[FILENAME_SIZE],
122             fldname[FLDNAME_SIZE],       fldabbr[FLDABBR_SIZE],
123             catname[CATNAME_SIZE],       catabbr[CATABBR_SIZE],
124             catvalue[5],                 text_buf[2000],
125             *types[] = { "O", "U", "C", "UC", "CC", "S", "E", "SC" },
126             *dashes = "----------" "----------" "----------" "----------"
127                       "----------" "----------" "----------" "---------"
128         ;
129         jmp_buf       env ;     /* setjmp/longjmp environment (see setjump)  */
```

The Micro Interface Functions access information in the Micro internal data structures, identifying them with variables declared as type **DATASET**, **FIELD**, and **CATEGORY**. These are C typedefs and are automatically included in the C code generated by the MPF preprocessor. The type **FIELD_TYPE** is another C typedef restricted by an enumeration to be one of the legal Micro field types: **U**, **UC**, **S**, **SC**, **C**, **CC**, or **E**. Other terms in all caps are used to define the sizes of strings needed to hold dataset names (**DSNAME_SIZE**), field names (**FLDNAME_SIZE**), etc.

```
-$List ILIR:Demo.MPF(131,162)

131         #define   Prog      "DSinfo: "        /*  used in error messages  */
132
133
134         if ( (dse = _ds_lookup (Dataset)) == NULL ) {
135             printf (Prog InQuotes(%s) " is not a dataset!" NL, Dataset) ;
136             return ;
137         }
138
139                       /*  Print general dataset information  */
140
141         printf (NL
142             "%s" NL
143             "%s  --  %s" NL
144             "       %s Destroyable    %s Replaceable    %s Scrambled" NL
145             "    %i fields in %s     %i records in %s" NL
146             "%s" NL,
147             dashes,
148             _ds_name (dse, dsname),  _ds_desc (dse, text_buf),
149             ( _can_destroy (dse) )  ?  ""  :  "Not",
150             ( _can_replace (dse) )  ?  ""  :  "Not",
151             ( _is_scrambled (dse) )  ?  ""  :  "Not",
152             _fld_cnt (dse),  _dict_file (dse, dictfile),
153             _recd_cnt (dse),  _data_file (dse, datafile),
154             dashes
155         ) ;
156
157         setjump (hadattn, env) ;      /*  this traps attns (see Handler below)  */
158         if ( hadattn ) {
159             printf (NL Prog "Attn!" NL) ;
160             _micrortn = MPF_ATTN ;
161             goto  Done ;
162         }
```

The function **_ds_lookup** takes a dataset name and returns a variable of type **DATASET**, which can be used to obtain attributes of the dataset. If **_ds_lookup** returns **NULL**, the name is not that of an available dataset.

The dataset attributes are produced by one `printf` with a long format. The '`NL`'s produce newline characters. The '`%s`' in line 142 corresponds to the '`dashes`' string in line 147. The '`%s`'s in line 143 correspond to the strings returned by the functions `_ds_name` and `_ds_desc` in line 148. These return the dataset name and description, respectively.

The '`%s`'s in line 144 correspond to the string expressions in lines 149, 150, and 151. The expression '`_can_destroy(dse) ? "" : "Not"`' has the value after the '`?`' if the expression before it is true, else it has the value after the '`:`'. The function `_can_destroy` returns 1 (non-zero is "true" in C) if the dataset `dse` can be destroyed, else 0. So if the dataset is destroyable, the expression produces a null string, '`""`', otherwise '`Not`', which precedes '`Destroyable`' in the format. The same is done for replaceable status returned by `_is_replaceable` and scrambled status returned by `_is_scrambled`.

The '`%i`'s in line 145 correspond to the integer number of fields returned by the function `_fld_cnt` and the integer number of records returned by `_recd_cnt`. The '`%s`'s in line 145 correspond to the dictionary file name returned by `_dict_file` and the Micro data file name returned by `_data_file`.

The code in lines 157 through 162 traps attentions. This allows DSinfo to clean up (see line 232 below) before stopping.

```
-$List ILIR:Demo.MPF(164,178)

    164         findfdub (dictfile, &dict_fdub) ;       /* get fdub for dictionary */
    165
    166                     /* Print field and category information */
    167
    168         for ( fld = _frst_fld (dse) ; fld != NULL ; fld = _next_fld (fld) ) {
    169
    170                         /* General field information */
    171
    172             type = _fld_type (fld) ;
    173
    174             printf (NL "%-16s  %-4s  %-2s  %8s   %s" NL,
    175                 _fld_name (fld, fldname), _fld_abbr (fld, fldabbr),
    176                 types[type], ( _data_reqd (fld) ) ? "DataReqd" : "",
    177                 _fld_desc (fld, dict_fdub, text_buf)
    178             ) ;
```

The next block begins with a call to `findfdub`, to get the MTS FDUB for the dictionary file. Field and category descriptions are stored in that file.

The clauses of the `for` loop show how to traverse the fields in a dataset. The function `_frst_fld` takes a dataset identifier and returns the identifier for the first field, or NULL if there are no fields. The function `_next_fld` takes a field identifier and returns the identifier for the next field, or NULL when there are no more fields.

General field information is printed with the help of functions returning (in the order in the `printf` call) the name of the field (`_fld_name`), the abbreviation (`_fld_abbr`), the type (returned by the call to **_fld_type** before the **printf** call), the data required switch (**_data_reqd**), and the field description (**_fld_desc**).

```
-$List ILIR:Demo.MPF(180,202)

    180                         /*  Field information specific to each type  */
    181
    182             switch ( type ) {
    183
    184             case U:   case UC:
    185             case S:   case SC:
    186                  printf ("        Range:  [%i, %i]" NL,
    187                       _min_val (fld), _max_val (fld)
    188                  ) ;
    189                  break ;
    190
    191             case C:   case CC:
    192                  printf ("        Length = %i" NL,
    193                       _fld_len (fld)
    194                  ) ;
    195                  break ;
    196
    197             case E:
    198                  printf ("        External file = %s" NL,
    199                       _ext_filename (fld, extfile)
    200                  ) ;
    201                  break ;
    202             }
```

The **switch** statement on the field type calls functions to print attributes appropriate to the different field type cases: minimum and maximum values for integer fields (**_min_val** and **_max_val**), length for character fields (**_fld_len**), and the name of the external data file for external fields (**_ext_filename**).

```
-$List ILIR:Demo.MPF(204,236)

    204                         /*  Number of categories  */
    205
    206             if ( (ncats = _cat_cnt (fld)) > 0 ) {
    207                  printf (NL
    208                       "  %i categor%s %s" NL,
    209                       ncats,  ( ncats == 1 )  ?  "y"  :  "ies",
    210                       ( _cat_reqd (fld) )  ?  "(required)"  :  ""
    211                  ) ;
    212             }
    213
    214                         /*  Print category information  */
    215
    216             for ( cat = _frst_cat (fld) ;  cat != NULL ;  cat = _next_cat (cat) ) {
    217
    218                  printf ("    %-12s   %-4s",
    219                       _cat_name (cat, catname),  _cat_abbr (cat, catabbr) ) ;
    220
    221                  if ( type == CC )
    222                       printf ("   %-10s", _cat_cval (cat, catvalue, fld) ) ;
    223                  else
    224                       printf ("   %-10i", _cat_ival (cat, fld) ) ;
    225
    226                  printf ("  %-7s  %s" NL,
    227                       ( _dfalt_cat (cat) )  ?  "Default"  :  "",
    228                       _cat_desc (cat, dict_fdub, text_buf) ) ;
```

```
                }
            }

    Done:
            losefdub (dict_fdub) ;         /* release the dictionary fdub  */

            printf (NL "%s" NL, dashes) ;
    }
```

The next block prints category information, if there are categories (i.e., `_cat_cnt` returns a positive value). The number of categories is printed along with the categories required switch (`_cat_reqd`).

The `for` loop shows how to traverse categories in a field. `_frst_cat` returns the first category in a field, or NULL if there are none; `_next_cat` returns the next category, or NULL if there are no more.

The first `printf` statement prints the category name (`_cat_name`) and abbreviation (`_cat_abbr`). The category value for a CC field is a string returned by `_cat_cval`; the category value for a UC or SC field is an integer returned by `_cat_ival`. The final `printf` statement prints the default category switch (`_dfalt_cat`) and description (`_cat_desc`)

The last lines close the loops, release the FDUB, and end the display with a line of dashes. Here's how it works.

```
-DSInfo Fin-Aid

 ------------------------------------------------------------------------------
 Fin-Aid  --  Example Financial Aid dataset for the Micro Course -- Tutorial 3 03-06-92
         Destroyable      Replaceable     Not Scrambled
      8 fields in K59F:FIN-AID$      34 records in K59F:FIN-AID$
 ------------------------------------------------------------------------------

 Student-ID-Num      SIN     U    DataReqd     Student Identification Number
         Range:  [0, 2147483647]

 Year-Awarded        Year    U    DataReqd     Year Awarded (yy)
         Range:  [0, 255]

 Term-Awarded        Term    UC   DataReqd     Term Awarded (YY)
         Range:  [0, 255]

   3 categories (required)
     fall             F      1                  fall
     Winter           W      2                  Winter
     Summer           S      3                  Summer

 Class-Level         CL      UC                Class-Level
         Range:  [0, 255]

   4 categories
     Freshman         Fr     1                  Freshman
     Sophomore        So     2                  Sophomore
     Junior           Jr     3                  Junior
     Senior           Sr     4                  Senior

 Type-of-Aid         Type    UC                Type of aid
         Range:  [0, 255]
   6 categories (required)
     Tuition        Tu     1             Tuition
     Room-board     Rb     2             room and Board
     Cash           Ca     3             Cash
     Loan           Lo     4             Loan
     Work-Study     Ws     5             Work Study
     Other          Oth    6             Other

 Source-of-Funds    SOF    SC          Source-of-Funds
        Range:  [-128, 127]

   7 categories (required)
     Unknown        Unk    -1            Unknown
     University     Univ   1             University
     State-Gov      SGov   2             State Government
     Federal-Gov    FGov   3             Federal Government
     Industry       Ind    4             Industry
     Foundation     Foun   5             Foundation
     Other          Oth    6             Other

 Amount             Amt    U           Amount
        Range:  [0, 2147483647]

 Remark*            Rem*   E           Additional remarks
        External file = K59F:REMARK*

 ---------------------------------------------------------------------------
```

Parameter Passing

Parameters to MPF Micro commands can be declared as required or optional, and optional parameters can be supplied with an initial value that is used if the parameter is not given in the command. The '`PrintPars`' program shows each.

```
-$List ILIR:Demo.MPF(239,253)

   239    /*
   240    ---------------------------------[ PrintPars ]----------------------------------
   241
   242         Print the values for the pars passed.
   243
   244              Usage:    PRINTPARS [ReqdPar=]value1 [OptPar=value2] [InitPar=value3]
   245
   246    ---------------------------------------------------------------------------------
   247    */
   248    micro  PrintPars  -
   249    printpars(char *ReqdPar, char *OptPar=, char *InitPar="the default") {
   250
   251         printf ("    PrintPars:  ReqdPar = %s, OptPar = %s, InitPar = %s" NL,
   252                                  ReqdPar, OptPar, InitPar) ;
   253    }
```

The `ReqdPar` parameter is required: if it is omitted in the command, Micro prompts for it. `OptPar` is optional: if it is omitted, the null string is passed to PrintPars. `InitPar` is optional with an initial value: if it is omitted, the string `"the default"` is used.

Parameters are assigned by combining the information given in the command with that in the declaration. If '`OptPar = value`' is given in the command, `OptPar` is assigned the '`value`'. If just '`value`' is given in the command, it is assigned to the next unassigned parameter in the list. Here are some examples

```
-printpars  1st 2nd 3rd
    PrintPars:  ReqdPar = 1st, OptPar = 2nd, InitPar = 3rd
```

In this command the parameters are assigned left to right from the values given in the command.

```
-printpars  par1
    PrintPars:  ReqdPar = par1, OptPar = , InitPar = the default
```

Here '`ReqdPar`' is assigned `par1`, `OptPar` has no value, and `InitPar` has its default value.

```
-printpars
 reqdpar? reply
    PrintPars:  ReqdPar = reply, OptPar = , InitPar = the default
```

No values are given in this command. Micro prompts for `ReqdPar`, `OptPar` has no value, and `InitPar` has its default value.

```
-printpars OptPar="OptPar value" "ReqdPar value" "InitPar value"
    PrintPars:  ReqdPar = ReqdPar value, OptPar = OptPar value, InitPar = InitPar value
```

Here `OptPar` is assigned '`OptPar value`'. `ReqdPar`, the first unassigned parameter, is assigned '`ReqdPar value`'. `InitPar`, the next unassigned parameter, is assigned '`InitPar value`'.

Generating and Executing Micro and MPF Micro commands

ILIR:Demo.MPF contains a Demo MPF Micro command, which prints a menu and allows you to select choices to execute features discussed in this tutorial. It executes Micro commands and other MPF Micro commands in ILIR:Demo.MPF, and prompts the user for a Micro command.

```
-$List ILIR:Demo.MPF(307,439)

    307     /*
    308     ----------------------------------[ Demo ]----------------------------------
    309
    310          This is the "main" demo program.  It presents a menu of demonstrations.
    311
    312               Usage:    DEMO
    313
    314          Notes:
    315          -  Demo takes no parameters.
    316          -  By changing the word "micro" to "fep" in the line following this
    317             comment and letting the MPF preprocessor compile this source file
    318             into the MPF library named MPFLIB on the master id (the default),
    319             this Demo program becomes a Front End Processor (FEP).  As such,
    320             Demo would automatically take control when you run ILIR:Micro.
    321
    322     ----------------------------------------------------------------------------
    323     */
    324     micro Demo  -
    325     mpfdemo(void) {
    326
    327          #define   REPLYLEN  81
    328
```

```
        int  percent = 1 ;              /*  this is used as a switch  */
        char
             *fin_aid = "Fin-Aid",
             *class_level = "Class-Level",
             *amount = "Amount",
             reply[REPLYLEN]
        ;

                     /*  Datasets from ILIR are used below  */

        Get ILIR
                echocmd () ;
                execute

                     /*  Menu choice loop  */

        for ( ; ; ) {

             printf (NL
                  "The following choices are available:" NL
                  "  1. Count    3. ParDemo        5. Micro commands   7. Menu help" NL
                  "  2. DSinfo   4. Micro prompt   6. Command file" NL
             ) ;
             if ( query ("Pick one (or just Return to exit the menu): ",
                             reply, REPLYLEN) != 0 || *reply == '\0' )
                  break ;

             switch ( atoi (reply) ) {

             case 1:   /*           Invoke the Count program
                            This invokes the Count MPF program using the "build"
                            and "execute" MPF keywords.  The "build" keyword must
                            be used since "Count" is not a Micro command.  Count
                            will ask for a dataset name since one is not given as
                            a parameter.
                       */
                  build  Count
                            execute

                  break ;

             case 2:   /*           Invoke the DSinfo program
                            This invokes the DSinfo MPF program which uses the
                            MPF library functions to access dataset information.
                            Like Count above, DSinfo will ask for a dataset name.
                       */
                  build  DSinfo
                            execute
                  break ;

             case 3:   /*           Invoke the ParDemo program
                            This invokes the ParDemo MPF program which exercises
                            MPF Micro command line parameter passing.
                       */
                  build  ParDemo
                            execute
                  break ;

             case 4:   /*           Go to the Micro Ready prompt
                            This goes to the Micro Ready prompt, allowing ad hoc
                            commands.  Note: you don't need "execute" here.
                       */
                  prompt
                  break ;

             case 5:   /*           Execute embedded Micro commands
```

```
394                          This builds and executes Micro commands.  Note the use
395                          of string expressions.  The MPF does not echo commands
396                          itself so the echocmd() function (defined above) is
397                          used to echo commands before executing them.
398                     */
399
400                     /*  The continuation char '-' wraps long command lines  */
401                Xtab@{ percent ? "" : "no" }%@noprint in {fin_aid}  -
402                     {class_level}, tot {amount}
403                          echocmd () ;
404                          execute
405
406                     /*  The append char ':' appends a line to the command  */
407                Print in Result {class_level}@lj, Tot.{amount}@$, Count
408                if ( percent ) {
409                     :, Percent
410                }
411                          echocmd () ;
412                          execute
413                break ;
414
415           case 6:   /*           Invoke a Micro command file
416                          This invokes a command file.  Note: you can't invoke
417                          a MPF program from a Micro command file.
418                     */
419                Set Command: Input = ILIR:MPFDemo.Cmd
420                          echocmd () ;
421                          execute
422                break ;
423
424           case 7:   /*           Help with menu choices
425                          This displays simple help information for Demo.
426                     */

427                printf (NL
428                     "    1. Count:  display the number of records in a dataset" NL
429                     "    2. DSinfo:  display (dictionary) information for a dataset" NL
430                     "    3. ParDemo:  MPF parameter passing demo" NL
431                     "    4. Have Micro prompt for a command" NL
432                     "    5. Execute imbedded Micro commands" NL
433                     "    6. Invoke a Micro command file" NL
434                     "    7. Display this" NL
435                ) ;
436                break ;
437           }
438      }
439 }
```

Note the '**build**' directive in line 365. This tells the MPF preprocessor to start building a command. Micro command names such as `Xtab` and `Print` tell the preprocessor to start building a Micro command, but '`build`' is necessary to start building a MPF Micro command.

The '`prompt`' macro in line 390 calls the Micro command interpreter to prompt for a Micro (or MPF Micro) command.

Note the expression in braces in the Xtab command in line 401, '`percent ? "" : "no"`'. Since it produces a character string, it is acceptable. This governs calculation of the Percent field for the Result set, depending on the value of the `percent` variable. The continuation character, '-', at the end of that line causes the next line to be appended to the command.

Note also the colon ':' notation following the Print command in line 409. Lines in MPF code beginning with a colon are appended to the existing Micro command, allowing conditional command construction. In this case the Percent field is printed if the `percent` variable is not zero.

A Simple MPF Handler

There are several exceptional conditions that can occur even in a relatively simple program like this demo. Besides attention interrupts, there may be errors in MPF programs (or even Micro!) that cause program interrupts, or the MPF program may generate a Micro command that is not correct. You can test for these conditions throughout the program but a better way is to use a *handler* program.

```
-$List ILIR:Demo.MPF(442,467)

/*
---------------------------------[ Handler ]-----------------------------------

     Exception handler

     This is called:
     -  After each Micro command generated by a MPF program
     -  After each MPF Micro command
     -  If any exception occurs (like an attention interrupt)

-------------------------------------------------------------------------------
*/
handler   Handler(int sig) {

     /*
          If sig is anything other than MPF_SUCCESS (i.e., if anything goes
          wrong), we jump to the previous setjump call or, if no setjump is
          active, we return to the Micro Ready prompt.  The value of sig is
          passed through unchanged.
     */

     if ( sig != MPF_SUCCESS )
          longjump (sig) ;

     return (sig) ;
}
```

As explained in the comments, this handler "quits" after any exceptional condition. In the DSinfo program, control returns from the `setjump` call in line 157 with a non-zero value for `hadattn` which allows DSinfo to release the dictionary FDUB before stopping. In all other cases, control returns to the Micro prompt.

Handlers for complex applications are usually a bit more involved than this simple example. A full discussion of MPF handlers can be found in Part IV.

Using the Demo MPF Micro command

Entering '**demo**' invokes the Demo MPF Micro command. The menu is printed with **printf**. The prompt is printed and the reply read with the MTS C function **query**.

```
-demo
    EchoCmd:  Get ILIR
 8 datasets from ILIR:Directory

 The following choices are available:
   1. Count    3. ParDemo        5. Micro commands   7. Menu help
   2. DSinfo   4. Micro prompt   6. Command file
 Pick one (or just Return to exit the menu): 7

    1. Count:  display the number of records in a dataset
    2. DSinfo:  display (dictionary) information for a dataset
    3. ParDemo:  MPF parameter passing demo
    4. Have Micro prompt for a command
    5. Execute imbedded Micro commands
    6. Invoke a Micro command file
    7. Display this

 The following choices are available:
   1. Count    3. ParDemo        5. Micro commands   7. Menu help
   2. DSinfo   4. Micro prompt   6. Command file
 Pick one (or just Return to exit the menu): 1
 dataset? students
  85 records in Students

 The following choices are available:
   1. Count    3. ParDemo        5. Micro commands   7. Menu help
   2. DSinfo   4. Micro prompt   6. Command file
 Pick one (or just Return to exit the menu): 3

 Entering the ParDemo program ...

   All pars given in order with left-hand sides (lhs):
     EchoCmd:  PrintPars  ReqdPar = 1st  OptPar = 2nd  InitPar = 3rd
     PrintPars:  ReqdPar = 1st, OptPar = 2nd, InitPar = 3rd

   All pars given in order without lhs:
     EchoCmd:  PrintPars  1st  2nd  3rd
     PrintPars:  ReqdPar = 1st, OptPar = 2nd, InitPar = 3rd

   No pars given (prompts for ReqdPar):
     EchoCmd:  PrintPars
 reqdpar? required value
     PrintPars:  ReqdPar = required value, OptPar = , InitPar = the default

   Only OptPar given (prompts for ReqdPar):
     EchoCmd:  PrintPars  OptPar = 1st
 reqdpar? required again
     PrintPars:  ReqdPar = required again, OptPar = 1st, InitPar = the default

   All pars given out of order with lhs:
     EchoCmd:  PrintPars  InitPar = 1st  OptPar = 2nd  ReqdPar = 3rd
     PrintPars:  ReqdPar = 3rd, OptPar = 2nd, InitPar = 1st

   All pars given out of order, one with lhs:
     EchoCmd:  PrintPars  OptPar = 1st  2nd  3rd
     PrintPars:  ReqdPar = 2nd, OptPar = 1st, InitPar = 3rd

 Exiting the ParDemo program.
 The following choices are available:
   1. Count    3. ParDemo        5. Micro commands   7. Menu help
   2. DSinfo   4. Micro prompt   6. Command file
 Pick one (or just Return to exit the menu): 4
-count fin-aid
  34 records in Fin-Aid

 The following choices are available:
   1. Count    3. ParDemo        5. Micro commands   7. Menu help
   2. DSinfo   4. Micro prompt   6. Command file
 Pick one (or just Return to exit the menu): 5
     EchoCmd:  Itab@%@noprint in Fin-Aid  Class-Level, tot Amount
  4 records in RESULT
  34 records represented
     EchoCmd:  Print in Result Class-Level@lj, Tot.Amount@$, Count, Percent

  Class-Level   Tot.Amount      Count      Percent
 -------------------------------------------------
  Freshman        $1580.00          6        17.64
  Sophomore       $1833.00          9        26.47
  Junior          $4275.00         11        32.35
  Senior          $3750.00          8        23.52

 The following choices are available:
   1. Count    3. ParDemo        5. Micro commands   7. Menu help
   2. DSinfo   4. Micro prompt   6. Command file
 Pick one (or just Return to exit the menu): 6
     EchoCmd:  Set Command: Input = ILIR:MPFDemo.Cmd

*    This is the Micro command file, ILIR:MPFDemo.Cmd, used in the MPF      -
*    Demo program (source in ILIR:Demo.MPF).  It contains only this         -
*    comment command.                                                       -
*
 End of command file.

 The following choices are available:
   1. Count    3. ParDemo        5. Micro commands   7. Menu help
   2. DSinfo   4. Micro prompt   6. Command file
 Pick one (or just Return to exit the menu):
-Stop
   $.02,  $1.29M ($.45S),  $17.06T
 Fri 6 Mar 1992  16:29:39
#Execution terminated   16:29:39  T=1.108
```

In the declaration of Demo (or any MPF Micro command), if '`micro`' is replaced with '`fep`', that routine is designated as a *Front End Processor.* It assumes control when the MPF library containing it is activated. If that library is named MPFLIB, it is activated automatically when Micro is $Run; the Micro prompt never appears and the FEP controls the interaction with the user.

FEPs and all aspects of the Micro Programming Facility are discussed in Part IV.

Part III

Command Reference

ILIR:Micro

ILIR:Micro Parameters

ILIR:Micro is run by:

$Run ILIR:Micro [`input=`*cmdfile*] [`print=`*outfile*] [`sercom=`*errfile*]
[`par=time=`{*n*|`OFF`} `dir=`*userid* `runfile=`{*runfile*|`OFF`} `cmt=`{`ON`|*mastercmtfile*}]

Micro normally reads commands from the terminal, but a command file can be assigned to `input`. Micro normally writes to the terminal. Assigning a file to `print` routes command output, such as '`10 records in Result`', to that file. Assigning a file to `sercom` routes error output, such as '`"stuents" is not a dataset`', to that file.

The `time` parameter sets a CPU time limit in seconds for individual Micro commands. The default is 10. `OFF` disables this time limit.

The `dir` parameter specifies a userid whose directory is to be read instead of the user's directory. The datasets in that directory are available, the ones in the user's directory are not.

The `runfile` parameter specifies a command file that is processed before a command prompt is issued. A runfile specified in this way overrides any runfile specified in the directory. If a runfile is specified in the directory, `OFF` prevents it from being processed.

The `cmt` parameter specifies a master command trace file. If a master command trace file specified in this way, or if `ON` is given, the command trace facility is activated. If a file is given, it overrides any file specified in the directory.

Example of Command Reference

Purpose: The Purpose section briefly states what the command does.

Synonyms: The Synonyms section lists synonyms, if any, for the command name, in minimum abbreviation form. In this form, the minimum abbreviation is capitalized, and optional letters are in lower case, as illustrated in the Usage section below.

Effect on the Result: This section discusses what effect the command has on the Result dataset. The Micro data manipulation commands produce a Result set; utility and miscellaneous commands produce none. The nature of the Result set is often elaborated on in the Description section.

Usage

The Usage section contains the prototype or syntax for the command, like the following example.

`Select IN` *dataset* [`ALL BUT`] <fields> [`,` <fields>]...

where

<fields> can be any of:

field
pattern
field `THRU` *field*

Any text appearing in `typewriter` is required in the command. The upper case portion of typewriter text denotes the minimum abbreviation for a word. If a typewriter word is entirely capitalized, it must be entered completely (in upper or lower case); any letters in lowercase may be omitted. In the above example, '`S`' denotes the minimum acceptable abbreviation for the Select command. '`s`', '`se`', and '`sel`' all work. The word '`IN`' has no abbreviation, and must appear entirely. Punctuation, such as '`,`', is required.

Generic terms are in *italic*. A generic must be replaced in a command with the appropriate name, number, etc. In this example, *dataset*, *field*, and *pattern* are generics. Here is a list of generic terms:

Syntax Generic Terms

Generic	Replace with ...
category	a category name or abbreviation
command	a Micro command
dataset	a dataset name
field	a field name or abbreviation
file	an MTS file or device name
item	an item to be set from one of the Micro Settings groups
number	a decimal number
pattern	a "wildcard" name, abbreviation, etc. (e.g., `A?`)
string	any string of characters, optionally enclosed in quotes or primes; if the string contains blanks, enclosure is required
unit	a dimension (preceded by a number)—points, inches, centimeters, millimeters, or twips (twentieths of a point): `1in`, `72pt`, `2.54cm`, `25.4mm`, `1440twip`
userid	an MTS userid (e.g., ILIR)

The generic terms also include minor variations on the above, such as '*input_file*', used when more than one *file* can appear on a command. The meaning of these is obvious from context.

Syntactic units are enclosed in angle brackets (< >). They are defined later in the prototype in terms of keywords, generics, phrases, or other syntactic units. In the example, '<fields>' is a syntactic unit.

Material appearing in brackets ([ ]) is optional. In the example, '`ALL BUT`' is an optional phrase.

Material followed by an ellipsis (...) can be repeated. In the example, '<fields> [`,` <fields>]...' indicates that one or more <fields> may be specified with commas separating them. At least one <fields> must be specified, the rest are optional.

Symbols enclosed in braces and separated by vertical bars are *alternatives*. Some common alternatives are '{`YES` | `NO`}' and '{`ON` | `OFF`}'.

Examples

The Examples section contains several typical examples of the command along with some comments. The intent is to demonstrate the various command forms and also some common command usage.

Settings

The Settings section lists the global settings, if any, that affect the command. All settings have default values initially. Settings are changed with the Settings command, and default values are restored with the Resettings command.

Modifiers

The Modifiers section lists modifiers, if any, that affect the command. A modifier allows you to override the default behavior of a command. Usually the default behavior is governed by a global setting.

Description

The Description section discusses in detail the command's effects and workings, what restrictions apply, etc.

CALCULATE Micro Command

Purpose: To create a Result dataset containing a copy of a given dataset with the result of arithmetic calculations placed into new or existing fields.

Synonyms: None.

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

`CALCulate IN` *dataset field* = <expression> [; *field* = <expression>] ...

where

<expression> can be (recursively) any of:

<operand>
<expression> <arith_op> <expression>
function ([<expression> [, <expression>] ...])
(<expression>)

<operand> can be any of:

field
category
number

<arith_op> can be any of:

```
+
-
*
/
^             **
mod           modulo
```

In plain English, <expression> can be a normal arithmetic expression with the arithmetic operators listed above and the functions listed below. Normal operator precedence can be overridden with parentheses.

Note: Currently there must be at least one space on each side of all <arith_op>s. This is because '+', '-', '*', and '/' are legal in Micro field names. For example, the expression '`A/B`' would be ambiguous if `A`, `B`, and `A/B` were all fields.

Examples

```
Calculate in Budget  Salary = Salary * 1.05
```

This example gives everyone in the Result set a 5% raise, multiplying the field Salary in dataset Budget by 1.05.

```
Calculate in Budget  NewSalary@Like=Salary = Salary * 1.05
```

This example puts the salary calculation of the previous example into a new field, NewSalary. The '`@Like=Salary`' creates the new field with all the attributes of Salary. Note that this modifier follows the name of the new field, not the command. Any of the modifiers that set attributes of the new field, as discussed below, must follow the name of the new field.

```
Calculate in It  YYMMDD = Year * 10000 + Month * 100 + Day
```

This example assigns to the field YYMMDD in the Result set a *packed date*, a six-digit integer which contains the year in the two leftmost digits (assuming a two-digit year), the month in the two middle digits, and the day in the two rightmost digits.

```
Calculate in It  Days = date() - from_y_m_d(Year, Month, Day)
```

This example assigns to the field Days in the Result set an integer, the number of days between the current date and the Year, Month, and Day contained in the record. The functions `date` and `from_y_m_d` return an internal date, the number of days since 1 March 0000.

```
Calculate in Students  JulBDay = julian(from_yymmdd(BDay))
```

This example assigns to the field JulBDay in the Result set a Julian date. The function `from_yymmdd` converts a six-digit packed date in YYMMDD form to an internal date. The function `julian` returns a Julian date (the number of days since 1 March 1900) from an internal date.

```
Calculate in It  RunMinutes = (time() - runtime()) / 60
```

This example assigns to field RunMinutes in the Result set the time of the Micro session so far. The function `time` returns the current time in internal form, the number of seconds since midnight; the function `runtime` returns in internal form the time ILIR:Micro was run to start the session. Dividing by 60 gives the minute in the session.

```
Calculate in Math  Shouldbe1 = sqrt(sin(Angle) ^ 2 + cos(Angle) ^ 2)

Calculate in Math  Shouldbe1 = (sin(Angle) ^ 2 + cos(Angle) ^ 2) ** .5
```

These examples assign a basic mathematical identity to the field Shouldbe1 in the Result set. The two forms are equivalent, obtaining the square root through the `sqrt` function in the first example, by exponentiation to the .5 power in the second.

```
Calculate in Schedule  Datestamp = yymmdd(date())
```

This example assigns to field Datestamp in the Result set a packed date, provided by function `yymmdd` from the internal date returned by the `date` function. The command '`Datestamp Schedule`' has the same effect.

```
Calculate in It  Key = recnum()
```

This example assigns to field Key in the Result set an integer which is the position of the record in the dataset. The command '`Key It`' has the same effect.

```
Calc in Students Byr = Bday / 10000 ; -
                 Bmon = Bday mod 10000 / 100 ; -
                 Bdate = Bday mod 100
```

This example assigns values to three fields; the expressions are separated by semi-colons. Field Bday is in packed date form, and integers for year, month and date are calculated from the expressions shown.

Settings

One setting affects the Calculate command. It determines whether the Result set includes all records, including any "problem records" resulting from the expression, such as division by zero. The initial setting puts only '`Goodrecords`' in the Result, the alternate puts in '`Allrecords`'.

The following settings affect the Calculate command.

Calculate Settings

Group	Item	Allowable values	Initially
`Calc:`	`Include`	`Goodrecords` or `Allrecords`	`Goodrecords`

Modifiers

These modifiers must follow the new field name in the command, not the command name. They are ignored if the field already exists.

`@TYPE={U|UC|S|SC|E}`	This modifier supplies the type for a new field. A type E field value is stored in the data file as the number of the line in the Micro external data file containing the actual (character) value. This line number value can be calculated and assigned to a type E field.

`@SCALE=[+|-|*|/]`*factor*	This modifier provides a scaling function and factor for a new field. For multiplication and division scaling, the scale factor must be a power of 10. The scaling factor and function are applied by Micro before the data are stored, and before they are output. A dollar amount would be scaled by division by 100. Data would be entered in decimal amounts, 125.76, stored by Micro as 12576, and printed on output as 125.76. `SCALE` also takes expressions like '`.00`'. This is equivalent to '`/100`', division by 100. If the scaling function is omitted, Micro assumes division. '`@SCALE=100`' is the same as '`SCALE=/100`'. If the factor is 0, scaling is disabled regardless of the function.
`@MAX=`{*value*|`B`*i*}	This modifier specifies the size of the field in bytes (`B`*i*, where *i* is the number of bytes) or gives a maximum *value*, from which Micro sets the field size. In the latter case, scaling is taken into account. The expression '`@max=99@scale=.00`' gives a field size of two bytes, needed to hold 9900, the internal form of 99.00. If the calculated value does not fit into the field, the record goes into the *Problems* dataset produced by the Calculate command, as discussed in the Description.
`@CATS=`*field*	This modifier applies the category structure of the specified *field*, which must be in the dataset given on the Calculate command.
`@LIKE=`*field*	This modifier applies all the attributes of the named *field* to the field being calculated. Attributes can be selectively overridden with other modifiers. If field Amt is scaled '`/100`', '`Newamt@LIKE=amt@scale=/1000`' assigns Newamt the value of Amt, but gives Newamt 3 decimal places instead of 2.

Description

The Calculate command assigns to a field or fields in a dataset the value of an arithmetic expression. This expression can include the usual arithmetic operations, represented by '+', '-', '*', '/', '^' or '**' for exponentiation, and '`mod`' for modulus. It can also include calls to functions, such as logarithmic functions, trigonometric functions, general functions such as average, absolute value and square root, functions for manipulating date and time, and other purposes. The expression can refer to other fields in the dataset and, if the field being assigned already exists, a category in that field.

The value of the expression can be placed in a new field or an old one. If a new field is created, it is by default a four-byte type S field, a signed fullword integer without categories. The scaling is according to the largest number of decimal places in the operands of the expression; if the expression requires three decimal places, the new field is scaled by division by 1000. A new field is the last field in the Result set.

If the expression is assigned to an existing field, the field type, length and scaling are not changed. The categories, if any, are kept.

If the expression is assigned to a new field, four modifiers, `@SCALE`, `@MAX`, `@CATS` and `@LIKE` can be used to set the scale factor and function; to give a size in bytes, or a maximum value from which a size is calculated; to use the categories from another field; or to use all the attributes of another field. These modifiers must appear after the name of the new field, not after the command name.

If an error condition occurs in a record, such as dividing by zero, the record is put into a special temporary dataset, named *Problems*. It contains all the fields in the Result set, plus a field named Calc.Problem at the end. The values for this field are:

Values for Field Calc.Problem in *Problems* Dataset

Value	Meaning
1	value can't fit into the field
2	division by zero (undefined)
4	numerator and denominator both zero (indeterminate)
8	date error, e.g., `from_y_m_d(`*yr*`,` *month*`, 32)`
16	time error, e.g., `from_h_m_d(`*hour*`,` *minute*`, 63)`

The *Problems* dataset is always created; if there are no problems, it is empty.

Whether "problem" records are included in the Result dataset is determined by the `CALC: INCLUDE` setting. The initial setting is:

```
Set Calc: Include Goodrecords
```

which includes valid records only in the Result. This can be overridden by:

```
Set Calc: Include Allrecords
```

which forces all records into the Result. However, the Result dataset does not contain the Calc.Problem field, even if there are problem records. That field is only in *Problems*.

There are a few restrictions in the Calculate command. Binary operators must be delimited by blanks, since '+', '-', '*', and '/' are allowed in field names. Functions can be nested up to 20 levels deep. When calculating more than one field, a new field cannot be used in subsequent expressions in the same command. For example, if Today is not a field in dataset A, then

```
Calc in A  Today = date();  Back90Days = Today - 90
```

will not work. However, it will work if Today is already a field in A.

The Usage section gives the syntax for expressions, and the Functions section describes the functions in detail. Most of the mathematical functions are from the Elementary Function Library, denoted 'EFL'. Further information on EFL functions is in MTS Volume 6.

Functions

Log Functions

These are the logarithmic functions:

`e()`	`e` returns the constant *e*, the base of natural logarithms.
`ln(`*x*`)`	`ln` returns the logarithm (base *e*) of the absolute value of *x*. (EFL)
`log10(`*x*`)`	`log10` returns the logarithm (base 10) of the absolute value of *x*. (EFL)
`exp(`*x*`)`	`exp` returns the exponential of *x*. If $x < -180.2182\ldots$, `exp` returns 0. If $x > 174.6730\ldots$, `exp` returns machine infinity. (EFL)

Trig Functions

These are the trigonometric functions:

`sin(`*x*`)`	`sin` returns the sine of *x*. (EFL)
`cos(`*x*`)`	`cos` returns the cosine of *x*. (EFL)
`tan(`*x*`)`	`tan` returns the tangent of *x*. (EFL)
`cot(`*x*`)`	`cot` returns the cotangent of *x*. (EFL)
`arcsin(`*x*`)`	`arcsin` returns the angle whose sine is *x*. (EFL)
`arccos(`*x*`)`	`arccos` returns the angle whose cosine is *x*. (EFL)
`arctan(`*x*`)`	`arctan` returns the angle whose tangent is *x*. (EFL)
`pi()`	`pi` returns the constant *pi*.

Arithmetic Functions

These are other arithmetic functions:

`int(`x`)`	`int` returns the integer part of x.
`abs(`x`)`	`abs` returns the absolute value of x.
`sign(`x`)`	`sign` returns the sign of x as follows: If $x < 0$, `sign` returns -1. If $x = 0$, `sign` returns 0. If $x > 0$, `sign` returns 1.
`min(`$x1$ [, $x2$] ...`)`	`min` returns the smallest of $x1$ [, $x2$] ... .
`max(`$x1$ [, $x2$] ...`)`	`max` returns the largest of $x1$ [, $x2$] ... .
`sum(`$x1$ [, $x2$] ...`)`	`sum` returns the sum of $x1$ [, $x2$] ... .
`avg(`$x1$ [, $x2$] ...`)`	`avg` returns the average of $x1$ [, $x2$] ... .
`sqrt(`x`)`	`sqrt` returns the square root of the absolute value of x. Note: You can get any root using a fractional exponent (e.g., `x ^ 0.5`). (EFL)

Date Functions

The Date functions are based on an *internal date* representing the number of days beginning with 1 March 0000.

There are functions to extract the year, month, day, and weekday from an internal date, to convert between an internal date and a *packed date* (yymmdd or yyyymmdd), and to convert between an internal date and a *Julian date* (number of days beginning with 1 March 1900).

These are the Date functions:

`date()`	`date` returns the internal date at the start of the current command. This remains constant throughout the command (it does not vary from record to record).
`rundate()`	`rundate` returns the internal date at the start of the Micro run. This remains constant throughout the Micro run.
`sigdate()`	`sigdate` returns the internal date at the start of the MTS signon. This remains constant throughout the signon.

year(*idate*)	**year** returns the 4-digit year [0000, ...] from an internal date, *idate*. If *idate* <= 0, **year** returns −1.
yr(*idate*)	**yr** returns the 2-digit year [00, 99] (current century) from an internal date, *idate*. If *idate* is not in the current century, **yr** returns −1.
month(*idate*)	**month** returns the month [1=Jan, 12=Dec] from an internal date, *idate*. If *idate* <= 0, **month** returns −1.
day(*idate*)	**day** returns the day [1, #days in the month] from an internal date, *idate*. If *idate* <= 0, **day** returns −1.
weekday(*idate*)	**weekday** returns the weekday [1=Sun, 7=Sat] from an internal date, *idate*. If *idate* <= 0, **weekday** returns −1.
julian(*idate*)	**julian** returns the Julian date from an internal date, *idate*. If *idate* is before 1-Mar-1900, **julian** returns −1.
yymmdd(*idate*)	**yymmdd** returns the packed date [000101, 991231] from an internal date, *idate*. If *idate* is not in the current century, **yymmdd** returns −1.
yyyymmdd(*idate*)	**yyyymmdd** returns the packed date [00000301, ...] from an internal date, *idate*. If *idate* <= 0, **yyyymmdd** returns −1.
from_julian(*jdate*)	**from_julian** returns the internal date from a Julian date, *jdate*. If *jdate* <= 0, **from_julian** returns −1.
from_yymmdd(*yymmdd*)	**from_yymmdd** returns the internal date from a packed date, *yymmdd*. If *yymmdd* <= 991231, the current century is assumed. If *yymmdd* is invalid, **from_yymmdd** returns −1.

`from_y_m_d(`*yy*`,`*mm*`,` *dd*`)`	`from_y_m_d` returns the internal date from a 3-part date: *yy*, *mm*, and *dd*. If *yy* <= 99, the current century is assumed. If *yy*, *mm*, or *dd* are invalid, `from_y_m_d` returns −1.

Time Functions

The Time functions are based on an *internal time* representing the number of seconds since midnight in one day. The internal time values range from 0 (12:00:00 midnight) through 86399 (11:59:59 pm). If an internal time parameter is > 86399, the time used is modulo 86400.

There are functions to extract the hour, minute, second, and am or pm from an internal time, and to convert between an internal time and a *packed time* (hhmmss).

These are the Time functions:

`time()`	`time` returns the internal time at the start of the current command. This remains constant throughout the command (it does not vary from record to record).
`runtime()`	`runtime` returns the internal time at the start of the Micro run. This remains constant throughout the Micro run.
`sigtime()`	`sigtime` returns the internal time at the start of the MTS signon. This remains constant throughout the signon.
`hour(`*itime*`)`	`hour` returns the hour [00, 23] from an internal time, *itime*. If *itime* < 0, `hour` returns −1.
`hour12(`*itime*`)`	`hour12` returns the hour [01, 12] from an internal time, *itime*. If *itime* < 0, `hour12` returns −1.
`minute(`*itime*`)`	`minute` returns the minute [00, 59] from an internal time, *itime*. If *itime* < 0, `minute` returns −1.
`second(`*itime*`)`	`second` returns the second [00, 59] from an internal time, *itime*. If *itime* < 0, `second` returns −1.
`pm(`*itime*`)`	`pm` returns 0 if *itime* is AM (ante meridian, twelve hours before 12 noon), 1 if *itime* is PM (post meridian, twelve hours after 12 noon). If *itime* < 0, `pm` returns −1.

`hhmmss(`*itime*`)`	**hhmmss** returns the packed time [000000, 235959] from an internal time, *itime*. If *itime* < 0, **hhmmss** returns -1.
`hhmmss12(`*itime*`)`	**hhmmss12** returns the packed time [010000, 125959] from an internal time, *itime*. If *itime* < 0, **hhmmss12** returns -1.
`from_hhmmss(`*hhmmss*`)`	**from_hhmmss** returns the internal time from a packed time, *hhmmss* [000000, 235959]. If *hhmmss* is invalid, **from_hhmmss** returns -1.
`from_h_m_s(`*hh*`,`*mm*`,`*ss*`)`	**from_h_m_s** returns the internal time from a 3-part time: *hh* [00, 23], *mm*, *ss*. If *hh*, *mm*, or *ss* are invalid, **from_h_m_s** returns -1.

Misc Functions

These are the miscellaneous functions:

`recnum()`	**recnum** returns the number of the current record.
`rand()`	**rand** returns a random number in the range from 0 to 1. **rand** uses the MTS **urand** function (see MTS Volume 3, "System Subroutine Descriptions"). The initial call to **urand** is made with a "seed" value of 0. This causes **urand** to generate an initial seed based on the time of day. Each subsequent call to **urand** uses the seed generated by the previous call. Thus, it is nearly impossible to get the same sequence of random numbers in any two commands.

CHANGE Micro Command

Purpose: To create a Result dataset containing a copy of a given dataset, with specified changes made to certain fields.

Synonyms: Alter

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

```
Change IN dataset [WHERE <condition> [<log_op> <condition>] ...]
                  SUCH THAT field = <value> [AND field = <value>] ...
```

where

<log_op> can be any of:

&	AND
|	OR
;	ALSO WHERE

<condition> can be any of:

```
field [IS] <rel_op> <right_side>
field [IS NOT] <rel_op> <right_side>
field = <value> [OR <value>] ...
field IS BETWEEN <value> AND <value>
field IS FROM <value> TO <value>
```

<rel_op> can be any of:

<	LESS THAN
<=	LESS THAN OR EQUAL TO
=	EQUAL TO
<>	IS NOT EQUAL TO
MAT	MATCHES
>=	GREATER THAN OR EQUAL TO
>	GREATER THAN

<right_side> can be any of:

field
<value>

<value> can be any of:

category
number
string

Examples

```
Change in Students where SIN = 123456789 such that BDay = 550317
```

The Result set contains all the fields and records from the Students dataset, but with the BDay field changed to 550317 for all records having SIN = 1234546789.

```
Change in A where F1 < 0 to 0
```

The Result set contains all the fields and records from dataset A, but with all negative values for the first field (F1) changed to zero.

Settings

None.

Modifiers

None.

Description

The Change command combines the data retrieval capability of the Find command with the ability to alter one or more fields in the Result set. The data retrieval syntax listed in the Usage section is discussed in the Description section of the Find command. Only the data alteration syntax is discussed here.

A single field change is denoted by '`SUCH THAT` *field* = <value>'. If *field* is character, and <value> is longer than the field length, the string is truncated. If *field* is numeric, and the <value> is too large for the field, a message is printed giving the acceptable range, and a new value prompted for. Multiple fields can be changed by appending one or more '`AND` *field* = <value>' to the first change expression.

If the changes made to a dataset result in duplicate records, the Result set will have fewer records than the original, since duplicates are weeded. The Key command can be used to make records unique, if weeding is not desired.

The line number stored with an external field can be altered with Change, but an external field value itself can be changed only through the Update command.

The cost of making a series of modifications to a large dataset can be reduced if the dataset can be divided into two temporary sets. The Find or Remove commands can produce from the original dataset a Result set to be worked on, and another Result (which can be renamed to a temporary set) to be left alone.

```
find in largeset where salary>500000
rename it wealthy
remove wealthy from largeset
rename it therestofus
change in wealthy where ...
change in it ...
...
combine therestofus with it
replace largeset with it
```

As in the above command sequence, the changes can be made to the working set, and the final Result combined with the unchanged set, using the Combine command. This Result can replace the original dataset, using the Replace command. The Replace must be done anyway, since Change produces a Result set containing the desired changes, rather than making them to the operand dataset.

COMBINE Micro Command

Purpose: To create a Result dataset containing copies of the records from one or more datasets which have the same number, type and size of fields.

Synonyms: None.

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

`COmbine` <datasets> [, <datasets>] ...

where

<datasets> can be any of:

dataset
pattern

Examples

```
Combine Fin-Aid84, Fin-Aid85, Fin-Aid86
```

The Result set contains copies of the Fin-Aid84, Fin-Aid85, and Fin-Aid86 datasets, assuming that the datasets have identical structure.

```
Combine Fin-Aid?
```

The Result set contains copies of the records from all datasets whose names begin with "Fin-Aid", assuming that the datasets have identical structure.

```
Combine Abcd
```

The Result set is a copy of dataset Abcd.

Settings

None.

Modifiers

None.

Description

The Combine command combines one or more datasets which have identical record structures: the same number, type and size of fields. If the sets are compatible, the data are merged and redundant records are weeded out. (If only one dataset is "combined", the Result set is simply a copy of that dataset.) The field information (name, abbreviation, categories, descriptions) from the first dataset specified is used for the Result set.

If the datasets are crosstabulated (contain Count and optionally Percent fields, as created by the Crosstabulate command) the data are merged, and records having the same values for all fields (except the Count field) are reduced to one record having those values. The Count field is the sum of the Count fields for the individual records. Combining crosstabulated sets preserves the aggregation achieved with Xtab.

COMMENT Micro Command

Purpose: To place a comment in a Micro session.

Synonyms: *

Effect on the Result: None.

Usage

`COMMent` *string*

Examples

```
Comment   This is a comment command
```

This just displays '`Comment   This is a comment command`'.

```
************************************************************
*                                                          *
*  The next few commands create a dataset, named WithAid,  *
*  containing only those students who have financial aid.  *
*                                                          *
************************************************************
```

These six comment commands display the "box" and its contents.

Settings

None.

Modifiers

None.

Description

The Comment command simply appears wherever Micro commands are echoed; it produces no other output. The primary use of the Comment command is to document command files. If command tracing is active, a Comment command is traced just like any other command.

CROSSTABULATE Micro Command

Purpose: To create a Result set containing a frequency count and (optionally) descriptive statistics for specified fields. These are generated by aggregating records in a dataset having the same values for the specified fields. The information in the Result is equivalent to that in an N-way table.

Synonyms: Tabulate, Xtab

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

```
CRosstab IN dataset [ALL BUT] [[<stat_op>] <fields>] [, [<stat_op>] <fields> ] ...
CRosstab dataset BY [ALL BUT] [[<stat_op>] <fields>] [, [<stat_op>] <fields> ] ...
```

where

<stat_op> can be any of:

```
TOT        TOTAL      SUM
AVE        AVERAGE    MEAN
STD        STD.DEV    SD
MIN        MINIMUM
MAX        MAXIMUM
MED        MEDIAN
```

<fields> can be any of:

field
pattern
field THRU *field*

Examples

`Crosstabulate in Students sex, race`

This example crosstabulates dataset Students by fields Sex and Race. The Result set contains those fields, and fields Count and Percent.

```
Xtab@no% Fin-Aid by cl, tot amt, ave amt
```

This example crosstabulates dataset Fin-Aid by field Class-level, calculates a total and average for the Amount field for each value of Class-level, and suppresses calculation of the Percent field. The Result set contains fields Class-level, Tot.amount, Ave.amount, and a Count field.

```
X A by f1, f2, f3, tot f9
```

This example crosstabulates dataset A by fields 1-3, and calculates a total for field 9. The Result set contains fields 1-3, a total for field 9, and Count and Percent fields.

```
X in Budget dept, tot ?$?
```

This example crosstabulates dataset Budget by field Department, and computes the total, within departments, for any fields whose names or abbreviations contain a dollar sign. The Result set contains field Department, and the total of any fields whose names or abbreviations contain a dollar sign, and Count and Percent fields.

```
X A by F2 thru F5
```

This example crosstabulates dataset A by fields 2 through 5. The Result set contains field 2-5, and Count and Percent fields.

```
xtab fin-aid
```

The Result set has one record, containing a count field holding the number of records in the set, and a percent field holding 100%.

```
xtab fin-aid by tot amt
```

The Result set has one record, containing the total for field Amount over all records, a count field holding the number of records in the set, and a percent field holding 100%.

Settings

The following settings affect the Crosstabulate (Xtab) command. In addition, the Print settings are used if the Result dataset is printed, as it is by default.

Crosstabulate Settings

Group	Item	Allowable values	Initially
XTAB:	AVE FACTOR	power of 10 or OFF	100
XTAB:	STD FACTOR	power of 10 or OFF	100
XTAB:	% FACTOR	power of 10 or OFF	100
XTAB:	PERCENT %	ON or OFF	ON
XTAB:	PRINT	ON or OFF	ON

`AVE FACTOR {n|OFF}`	The **AVE FACTOR** affects the number of decimal places in any average calculated. A value of 100 causes two decimal places to be added to the answer. Any existing scale factor for the field is multiplied by the **AVE FACTOR**; a field with a scale factor of 100 has four decimal places in the answer if **AVE FACTOR** is 100. (default: 100)
`STD FACTOR {n|OFF}`	The **STD FACTOR** is the the number of decimal places in any standard deviation calculated. A value of 100 causes two decimal places to be added to the answer. Any existing scale factor for the field is multiplied by the **STD FACTOR**; a field with a scale factor of 100 has four decimal places in the answer if **STD FACTOR** is 100. (default: 100)
`% FACTOR {n|OFF}`	The **% FACTOR** is the the number of decimal places in the percentage if it is calculated. The default value of 100 causes two decimal places to appear in the answer.

`PERCENT {ON|OFF}`
`%`
	The **PERCENT** setting affects calculation of the Percent field, the percentage of records in the dataset falling into each of the crosstabulated groups. (default: **ON**)

`PRINT {ON|OFF}`	The **PRINT** setting controls the printing of the Result set. If **ON**, the Xtab command prints the set automatically. (default: **ON**)

Modifiers

The following modifiers can be used to override some of the Xtab settings.

`@PRint`	Print the Result dataset.
`@NOPRint`	Don't print the Result dataset.
`@PERCent` `@%`	Calculate the Percent field in the Result.
`@NOPERCENT` `@NO%`	Don't calculate the Percent field in the Result.

In addition, the Print command modifiers can be used to override the Print settings if the Result dataset is automatically printed. Any Print field modifiers are ignored.

Description

The Crosstabulate command groups (aggregates) records that have the same values over the fields specified in the command, counts the records in each group, and optionally calculates descriptive statistics (e.g., average) on other fields in the group. For example, the command

```
Xtab in Students Sex
 3 records in RESULT
 85 records represented

 Sex              Count      Percent
------------------------------------
 Female              45        52.94
 Male                39        45.88
 Unknown              1         1.17
```

groups the records from Students by Sex and counts the number of records in each group. The Result contains Sex, the field "xtabbed over", and Count, the number of the records in each group. The field abbreviation for Count is 'Cnt'.

By default, the Xtab command calculates a Percent field showing what percentage of the records are in each group. You can suppress this using the **XTAB: PERCENT** setting or the **@NOPERCENT** command modifier. The field abbreviation for Percent is '%'.

You can generate the equivalent of an N-way table by xtabbing over N fields. For example, the command

```
xtab students by sex, class-year
 10 records in RESULT
 85 records represented

 Sex           Class-Year       Count      Percent
--------------------------------------------------
 Female                77          13        15.29
 Female                78          10        11.76
 Female                79          12        14.11
 Female                80           9        10.58
 Female           Unknown           1         1.17
 Male                  77           8         9.41
 Male                  78          12        14.11
 Male                  79          10        11.76
 Male                  80           9        10.58
 Unknown               79           1         1.17
```

yields the same information found in a 2-way table. Each record in the Result represents a unique combination of the values for Sex and Class-Year found in Students. Note that only the combinations actually found in the dataset are represented. Other possible combinations, such as Sex=Male and

Class-Year=Unknown, would have Count=0 and are not included in the Result. These are like the empty cells in a 2-way table.

Micro notes that a Result set is the result of an Xtab command. If you Xtab the result of a previous Xtab command, Micro totals the values for the Count field in each new group. For example (using the previous Result), the command

```
xtab it by sex
 3 records in RESULT
 85 records represented

 Sex              Count      Percent
-------------------------------------
 Female              45        52.94
 Male                39        45.88
 Unknown              1         1.17
```

yields the same result as did '`Xtab Students by Sex`'. Instead of counting the records in the Female group (for example) from the previous Result and getting a new Count of 5, Micro totals the Count field for those 5 records.

You can use the Xtab command to calculate simple descriptive statistics on the values for fields in a group of records. For example, the command

```
xtab in fin-aid by cl, tot amt, ave amt
 4 records in RESULT
 34 records represented

 Class-Level   Tot.Amount   Ave.Amount        Count       Percent
------------------------------------------------------------------
    Freshman      1580.00     263.3333            6         17.64
   Sophomore      1833.00     203.6667            9         26.47
      Junior      4275.00     388.6364           11         32.35
      Senior      3750.00     468.7500            8         23.52
```

groups the records from Fin-Aid by Class-Level, counts the number of records in each group, and generates the total and average of the values for Amount found in each group. The Ave.Amount is equal to Tot.Amount divided by Count. By default, averages are given 2 additional decimal places. You can change this using the `XTAB: AVE FACTOR` setting.

Micro creates a new field name for each descriptive statistic field in the Result. The new name has the form *stat.field* where *stat* indicates what statistic was calculated and *field* is the first twelve characters of the specified field name. The field abbreviation is carried over to the Result unchanged. The descriptive statistics available in the Xtab command and their field prefixes are:

Descriptive Statistics From Crosstabulate Command

Operation	Field Prefix
Total	`Tot.`
Average	`Ave.`
Standard deviation	`Std.`
Minimum	`Min.`
Maximum	`Max.`
Median	`Med.`

If a field used in a statistical operation has categories, they are carried over to the Result only for the minimum, maximum, and median operations.

Fields used in statistical operations should be 4-byte signed integers (types S and SC) or unsigned integers of any length (U and UC). Statistical operations on signed integers less than 4-bytes will yield an incorrect value if any of the numbers contributing to that value are negative. Micro displays a warning message in this case. If you specify a statistical operation on a character field (type C or CC), Micro displays an error message and skips that statistical operation. Any field created by a statistical operation is a 4-byte signed integer, fullword-aligned in the Result.

You can generate descriptive statistics over all of the records in a dataset by not specifying any fields to xtab over. This treats the entire dataset as one group. For example, the command

```
Xtab in Fin-Aid min Amt, max Amt, tot Amt, ave Amt, std Amt, med Amt
 1 record in RESULT
 34 records represented

  Min.Amount   Max.Amount   Tot.Amount   Ave.Amount   Std.Amount   Med.Amount      Count
------------------------------------------------------------------------------------------
       50.00      1200.00     11438.00     336.4118     263.6519       265.00         34
```

creates a Result holding one record with all of the statistical operations performed on the Amount field over all the records in Fin-Aid.

DATESTAMP Micro Command

Purpose: To create a Result dataset containing a copy of a given dataset, with a new or existing field containing the current date in integer "packed date" (YYMMDD) or Julian (number of days since 1 January 1900) form.

Synonyms: None.

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

`DAtestamp` *dataset* [{`AT BEGINNING` | `AT END`}] [`INTO` *field*] [`USING` <datefmt>]

where

<datefmt> can be one of

```
YYMMDD
YYYYMMDD
JULIAN
```

Examples

`Datestamp Fin-Aid`

The Result set is a copy of the Fin-Aid dataset, containing a new Datestamp field at the end.

`Datestamp A into Date`

The Result set is a copy of dataset A, with the current date inserted into field Date.

`Datestamp Fin-Aid into Today using yyyymmdd`

The Result set is a copy of Fin-Aid, with the current date in YYYYMMDD format in the field Today.

Settings

None.

Modifiers

`@NV`	This suppresses confirmation if datestamping into an existing field not specified on the command.

Description

If the Datestamp command creates a field, and the name is not given on the command, the field is named Datestamp, with abbreviation D.S. The field is placed at the end of the Result set, unless '`AT BEGINNING`' is specified. If '`INTO` *field*' is specified on the command, that name is used.

If the field to contain the date exists, it must be numeric and large enough to hold the value. If there are problems, Micro asks for a replacement field. If the field does not exist, it is created as a type U (unsigned integer), three or four bytes, depending on the date format.

Date format `YYMMDD` is the default format (two digits each for year, month and date), and creates a 3-byte type U (unsigned integer) field. The other two, `YYYYMMDD` (four digits for year, two each for month and date) and `JULIAN` (the number of days since 1 March 1900), create 4-byte type U fields. YYYYMMDD or Julian formats are selected by specifying '`USING YYYYMMDD`' or '`USING JULIAN`' on the command.

DESCRIBE Micro Command

Purpose: To print the descriptions for datasets, fields, or categories.

Synonyms: None.

Effect on the Result: None.

Usage

`DEscribe` [`ON` *file*] {`ALL DATASETS` | `*`}

`DEscribe` [`ON` *file*] [`ALL BUT`] <datasets> [, <datasets>] ...

`DEscribe` [`ON` *file*] `IN` *dataset* `FIELDS`

`DEscribe` [`ON` *file*] `IN` *dataset* [`ALL BUT`] <fields> [, <fields>] ...

`DEscribe` [`ON` *file*] `IN` *dataset* `CATEGORIES OF` *field*

`DEscribe` [`ON` *file*] `IN` *dataset* *field* [`ALL BUT`] <categories> [, <categories>] ...

where

<datasets> can be any of:

dataset
pattern

<fields> can be any of:

field
pattern
field `THRU` *field*

<categories> can be any of:

category
pattern
category `THRU` *category*

Examples

```
Describe Budget?, Students, Fin-Aid
```

This prints descriptions for all datasets having names beginning with "Budget" and for the Students and Fin-Aid datasets.

```
Desc in Fin-Aid term
```

This prints descriptions for the field Term in the Fin-Aid dataset.

```
Desc in Fin-Aid term ?
```

This prints descriptions for all categories of the field Term in the Fin-Aid dataset.

Settings

None.

Modifiers

None.

Description

The Describe command prints descriptive attributes of dataset, fields and categories. The command

```
   desc students
```

produces

```
 Students -- Example data set -- Demographic information on students
              09-11-81/ILIR
```

the name of the dataset, its description, creation date and owner userid. The command

```
   desc in students fields
```

produces

```
F1  Student-ID-Num   SIN      DataReq  Student Identification Number

F2  Name             Na       DataReq  Name of Student (Last First Middle
                                        format)

F3  Ethnicity        Race     DataReq  Ethnicity
           7 categories (required)

F4  Sex              Sex      DataReq  Sex
           3 categories (required)

F5  Birthdate        Bday     DataReq  Birthdate (YYMMDD format)
           1 category

F6  Class-Year       ClYr              Class Year (YY format)
           1 category
```

the number, name, abbreviation, data required status, description, and category number and required status. The command

```
desc in students categories of sex
```

produces

```
      Male          Ma          M    Male

      Female        Fe          F    Female

      Unknown       Unk         U    Unknown
```

the category name, abbreviation, value and description.

DESTROY Micro Command

Purpose: To destroy a Micro dataset and its dictionary and data files.

Synonyms: None.

Effect on the Result: The Result set is affected only if it is destroyed.

Usage

```
Destroy [ALL BUT] <datasets> [{OK|ALLOK|PROMPT}]
```

where

<datasets> can be any of:

dataset
pattern

Examples

```
dest a
```

This example destroys dataset A.

```
dest a?, b?
```

This example destroys all datasets beginning with 'a' and 'b', prompting once for confirmation for all the datasets.

```
dest a?, b? prompt
```

This example destroys all datasets beginning with 'a' and 'b', prompting for confirmation for each dataset.

```
dest all but a?, b?  allok
```

This example destroys all datasets except those beginning with 'a' and 'b', without any confirmation prompting.

Settings

None.

Modifiers

None.

Description

The Destroy command destroys the dictionary and data files for the specified datasets, if the datasets are destroyable. If the dictionary file is used by a dataset not specified in the command, it is not destroyed. External data files are not affected, and must be destroyed with the MTS $Destroy command.

When destroying a single dataset, Micro asks for confirmation. '`OK`' or '`ALLOK`' must be specified on the command to suppress this prompt.

When destroying multiple datasets, Micro lists the sets and asks for confirmation to destroy them all. '`ALLOK`' can be specified to suppress this prompt. If '`PROMPT`' is specified Micro asks for confirmation for each dataset. If a combination of names and patterns would cause all destroyable datasets to be destroyed, Micro prompts for confirmation even if '`OK`' or '`ALLOK`' was specified on the command.

The MTS *Restore facility can be used to restore a file until about six weeks after it has been destroyed. The MTS *FS facility allows users to make their own file save tapes. *Restore and *FS are documented in MTS Volume 2.

DISPLAY Micro Command

Purpose: To display the cost, time and date, or virtual memory allocation.

Synonyms: None.

Effect on the Result: None.

Usage

`Display` <item> [, <item>] ...

where

<item> can be any of:

```
COST          $
TIME          DATE
VM            VMSIZE
ALL
```

Examples

`Display time`

This displays the current time and date.

`Disp $`

This displays current values for the cost figures that are printed when the **`COST`** setting is **`ON`**, or when a Micro session is ended, which include the amount spent in the command, the amount spent in the Micro session, the surcharge on the Micro cost in parentheses, and the cost of the MTS session.

```
d all
  $.00,  $.06M ($.02S),  $.52T
Mon 17 Sep 1990  17:23:24
VM size = 243 pages
```

The '**`all`**' option produces the above.

Settings

None.

Modifiers

None.

Description

The Display command prints one or all of three informational items about the Micro session. `COST` includes the cost of the (Display) command, the cost of the Micro session, the Micro surcharge (in parentheses) and the total cost of the MTS session so far. Both `DATE` and `TIME` print the date and time. `VM` prints the virtual memory allocation in pages; each MTS page is 4096 bytes. `COST` is also a setting, in the `COMMAND` group; when set to `ON` with the Set command (`set cmd:  cost on`) cost figures are automatically printed after each command.

DOCUMENT Micro Command

Purpose: To print the general documentation for a dataset, fields in a dataset, or categories of a field.

Synonyms: None.

Effect on the Result: None.

Usage

`DOcument` [`ALL`] `DATASETS` [<option>] ...

`DOcument` [`ALL BUT`] <datasets> [, <datasets>] ... [<option>] ...

`DOcument IN` *dataset* [<option>] ...

`DOcument IN` *dataset* [`ALL BUT`] <fields> [, <fields>] ... [<option>] ...

`DOcument IN` *dataset* `CATEGORYS OF` *field* [`ON` *file*]

`DOcument IN` *dataset field* [`ALL BUT`] <categories> [, <categories>] ... [`ON` *file*]

where

<datasets> can be any of:

dataset
pattern

<fields> can be any of:

field
pattern
field `THRU` *field*

<categories> can be any of:

category
pattern
category `THRU` *category*

<option> can be any of:

`NO CATEGORIES`   `NOCATS`   `FIELDS ONLY`   `*`
`ON` *file*

Examples

`Document Students on -doc`

This command documents dataset Students on file –DOC.

`Doc in Fin-Aid *`

This command documents all fields in dataset Fin-aid, without category information.

`Doc in A f1 thru f5, f10`

This command documents fields 1-5 and 10 in dataset A.

`Doc cats of Race in Students`

This command documents the categories of field Race in dataset Students.

Settings

None.

Modifiers

None.

Description

The Document command produces general documentation for one or more datasets, or fields within datasets. When a complete dataset is documented, a header like the following is printed:

```
Fin-Aid -- Example data set -- Financial aid information   09-11-81/ILIR
         Not Scrambled      Not Destroyable      Not Replaceable
  Dictionary file = ILIR:FIN-AID#       Micro Data file = ILIR:FIN-AID
```

The header consists of the dataset name, description, creation date and owner; the scrambled, destroyable and replaceable status; and the names of associated files. When '`IN`' appears on the command, the dataset header is omitted, and only fields and/or categories are documented. In any case, they are documented as shown below.

```
--------------------------------------------------------------------------
 F#  Field name       Abbr        Value Description
--------------------------------------------------------------------------

 F1  Student-ID-Num   SIN       DataReq Student Identification Number

 F2  Year-Awarded     Year      DataReq Year Awarded (YY format)

 F3  Term-Awarded     Term      DataReq Term Awarded
            3 categories (required)
        Fall          F               1 Fall Term
        Winter        W               2 Winter Term
        Summer        S               3 Summer Term

 F4  Class-Level      CL        DataReq Class Level
            4 categories (required)
        Freshman      Fr              1 Freshman
        Sophomore     So              2 Sophomore
        Junior        Jr              3 Junior
        Senior        Sr              4 Senior

 F5  Type-of-Aid      Type      DataReq Type of Aid
            6 categories (required)
        Tuition       Tu              1 Tuition
        Room-Board    RB              2 Room and Board
        Cash          Ca              3 Cash
        Loan          Lo              4 Loan
        Work-Study    WS              5 Work Study Program
        Other         Oth             6 Other

 F6  Source-of-Funds  SOF       DataReq Source of Funds
            6 categories (required)
        University    Univ            1 University
        State-Gov     SGov            2 State Government
        Federal-Gov   FGov            3 Federal Government
        Industry      Ind             4 Industry
        Foundation    Foun            5 Foundation
        Other         Oth             6 Other

 F7  Amount           Amt       DataReq Amount of Award

 F8  Remark*          Rem*              Additional Comments
         External file = ILIR:REMARKS*
```

For fields, the `F#` through `Abbr` columns, and the `Description` are self-explanatory. The `Value` is the data required status. If a field has categories, the number and the categories required switch are shown. For categories, the `F#` column is blank, `Name`, `Abbr` and `Description` are self-explanatory, and `Value` is the category value. If `NOCATS` is specified, the individual category information is omitted.

ENTERDATA Micro Command

Purpose: To enter data in free format into a dataset.

Synonyms: None.

Effect on the Result: If `PERMANENTLY` is specified then the Result dataset is not affected. Otherwise new data and existing data are entered into a new Result set, replacing the existing Result.

Usage

`Enterdata INTO` *dataset* [{`PERMANENTLY`| `PERM`}] [`FROM` *input_file* `WITH ERRORS ON` *error_file*]

Examples

```
Enter into it
```

This enters data into the Result set, prompting for data and notifying and querying for error correction interactively.

```
Enterdata into Fin-Aid perm from FA.IN with errors on -err
```

This enters data permanently into dataset Fin-aid, reading from file FA.IN, and writing invalid input records on file -ERR.

Settings

None.

Modifiers

None.

Description

If `PERM` or `PERMANENTLY` is given, data is entered directly into the dataset. (Permanent data entry may not be specified for a temporary dataset). If '`FROM` *input_file* `WITH ERRORS ON` *error_file*' is given, data is read from *input_file*, and invalid records written on *output_file*. Otherwise, the program prompts for data. Whether data is read from a file, or entered interactively, the behavior of the Enterdata command is the same as the stand-alone program ILIR:Micro.Add.

For details of prompting, error analysis and correction, and format specification, see the sections on Interactive and Free-format in the chapter on Data Entry.

EXPORT Micro Command

Purpose: To write a dataset and/or commands on file(s) in ASCII, OSIRIS, SAS, SPSSPC, SPSSX or WK1 format, for use on MTS or downloading to a microcomputer.[2]

Synonyms: None.

Effect on the Result: None.

Usage

`EXPOrt FOR` <format> `IN` *dataset field* [, *field*] ... `ON` *file* [{PCFILE|HANDLE|SASNAME}="*string*"]

`EXPOrt FOR` <format> *dataset* `ON` *file* [{PCFILE|HANDLE|SASNAME}="*string*"]

where

<format> is one of

```
ASCII
OSIRIS
SAS
SPSSPC
SPSSX
WK1
```

Examples

```
export for ascii on -asc students
```

This command exports the entire dataset Students in ASCII format on file –ASC.

```
Export for wk1 in it f1@ssn, f2, f3@cat=abbr on -abc
```

This command exports, in WK1 format, Field 1 as a social security number, Field 2 as its default type, and Field 3 as abbreviations of the underlying categories, on file –ABC.

[2]SAS is a registered trademark of the SAS Institute, Inc. SPSS-X and SPSS/PC+ are registered trademarks of SPSS, Inc. Microsoft Excel is a registered trademark of Microsoft Corporation. Lotus 1-2-3 is a registered trademark of Lotus Development Corporation.

```
Export for spsspc in it f1, f2, f3 on -stu pcfile="c:\1992\stu.d"
```

This command exports, for SPSS/PC+, Fields 1, 2 and 3 on files -STU.C and -STU.D. The command file -STU.C names C:\1992\STU.D as the data file on the Data List statement. Thus, -STU.D must be downloaded to C:\1992\STU.D on the microcomputer.

```
Export for spssx in it f1@date=internal@dd-mmm-yyyy, -
   f2@date=internal@mm-dd-yyyy on -date handle="timetest"
```

This command exports, for SPSSX, Field 1 in the `dd-mmm-yyyy` format, and Field 2 in the `mm-dd-yyyy` format. The `handle` is given as '`timetest`', and appears on the SPSSX File Handle command in -DATE.C. The `@internal` modifiers tell Micro what form the date is in so it can be converted properly to character.

Settings

None.

Modifiers

Modifiers on the Export command may affect the entire command, or single fields (or both). The following tables summarize modifiers that affect both command and fields and notes which formats each are valid for. A modifier that is invalid for a particular format does not cause an error, but is simply ignored.

Export Command and Field Modifiers

Modifier	Function	Command	Field	Format
@#CHars=*n*	length of char cell	×	×	ASCII, WK1
@CATegory=*form*	cats as name, abbr, desc or value	×	×	all
@CENtered	col heads centered	×	×	WK1
@Count	# records exported	×		all
@DECimalplaces=*n*	# decimal places in real numbers	×	×	WK1
@DELIMiter=*c*	field separator	×		ASCII
@FIELD=*form*	var names as field name, abbr, or desc	×	×	all
@FIrst=*i*	first record exported	×		all
@Forward	records exported in ascending order	×		all
@[~]HEADings	column headings	×	×	ASCII, WK1
@HEADjustification=*just*	col heads left, right, or cent	×	×	WK1
@Increment=*i*	incr between records	×		all
@Justify=*just*	char fields left, right or cent	×	×	WK1
@LEFTJustify	col heads, char fields left	×	×	WK1
@MISSing=*string*	missing data value	×	×	OSIRIS, SPSSPC, SPSSX
@[~]PROTected	cells unalterable in spreadsheet	×	×	WK1
@Reverse	records exported in descending order	×		all
@RIGHTJustify	col heads, char fields right	×	×	WK1

@#CHaracters=*n*
@LENgth

This modifier determines the length of a character field for WK1 format. A character field can result from a field of type C, or from categorical data, if category names or abbreviations are used instead of values. The default is the length of the field in Micro, or the maximum length of a category name (11), or abbreviation (4), or maximum length over all descriptions for the category, if the field is printed that way.

@CATegory={NAME|ABBR|VALUE|DESCription}

This modifier controls the appearance of categorical data, as the category name, or abbreviation, or value, or description, for WK1 and ASCII formats. For formats that support categorical data, the value is always exported, so this modifier affects the contents of the category string; **VALUE** results in **NAME**.

@CENTered

This modifier centers character data in their columns for WK1.

@Count=*n*

This modifier specifies the maximum number of records to be exported. **@C=25** exports at most 25 records from the dataset.

@DECimalplaces=*n* **@DP=*n***	This modifier controls the number of decimal places in numeric fields. **@DP=2** causes 2 decimal places to appear for WK1.
@DELIMiter=*c*	This is the delimiter for field values in ASCII format. (default: '**,**')
@FIELD={NAME|ABBReviation|DESCription} **@FLD**	This modifier affects the format of column headings for WK1 and ASCII formats, causing field names, abbreviations, or descriptions to be used. For all other formats, it affects the variable names, except that only **NAME** and **ABBR** are recognized; **DESC** results in **NAME**. (default: **NAME**)
@FIrst=*i*	This modifier specifies the first record to be exported; exporting does not have to begin with the first record in the dataset.
@Forward	This modifier causes records to be exported in ascending order, from the first to the last in the dataset. This is the default.
@[~]HEADings	This modifier controls the appearance of column headings (field names, abbreviations or descriptions). The default is for field names to appear. **@~HEAD** omits the headings.
@HEADJustification={LEFT|RIGHT|CENTER} **@HEADJustify**	This modifier controls column heading alignment, left, right or center. (default: **LEFT**)
@Increment=*i*	This modifier controls the increment between exported records. An increment of 2 would export every other record after the first. (default: 1)
@Justify={LEFT|RIGHT|CENTER} **@Justification**	This modifier controls the alignment of WK1 character values in fields, left, right or center. (default: **LEFT**)
@LEFTJustify **@LJustify**	This modifier causes all WK1 character fields to be left-aligned.
@MISSing=["]*string*[,*string*]... ["]	This modifier provides missing data values for SPSS mainframe and micro versions, and for OSIRIS. For SPSS, the value is emitted on the Missing statement; for OSIRIS, the value appears on the variable descriptor. Multiple values may be given, separated by commas, with the entire string enclosed in quotes, as the optional elements of the syntax indicate.

`@[~]PROTected`	This modifier causes a field to be protected (unchangeable) in WK1 format. (default: `@~PROTected`)
`@Reverse`	This modifier causes records to be exported in descending order, from the first exported record to the first record in the dataset. (Forward is the default).
`@RIGHTJustify` `@RJustify`	This modifier causes field headings and character values in WK1 fields to be right-justified.

Field modifiers affect the appearance of individual fields on the worksheet screen, and in some cases the underlying storage of the data. For example, a number representing a date could appear in several different ways, depending on the date modifier used. The Export command date modifiers corresponds to WK1 date formats. This format can be changed easily in the spreadsheet program; the underlying data type is more difficult to change. Normally, users are concerned only with the on-screen data format, not the underlying WK1 data type, which Micro selects based on the requirements of the field. That data type can be overridden when necessary, as explained below. The following table summarizes the field modifiers; command modifiers that also modify fields are in the previous table.

Export Field Modifiers

Modifier	Function	Format
@+/-	WK +/- format	WK1
@$	currency fmt	ASCII, WK1
@%	percent format	WK1
@COMMA	comma format	WK1
@DATE=*form*	date packed, internal, Julian	ASCII, SPSSX, WK1
@DD-MMM	date format	ASCII, WK1
@DD-MMM-YY	date format	ASCII, SAS, SPSSX, WK1
@DD-MMM-YYYY	date format	ASCII, SAS, SPSSX
@12HH:MM	time format	ASCII, WK1
@12HH:MM:SS	time format	ASCII, WK1
@24HH:MM	Short Intl time format	ASCII, SAS, SPSSX, WK1
@24HH:MM:SS	Long Intl time format	ASCII, SAS, SPSSX, WK1
@INTeger	integer type	WK1
@LABEL	char data type	ASCII, WK1
@MM/DD	Short Intl date format	ASCII, WK1
@MM/DD/YY	Long Intl date format	ASCII, SAS, SPSSX, WK1
@MM/DD/YYYY	extra-Long Intl date format	ASCII, SAS, SPSSX
@MMM-YY	date format	ASCII, WK1
@MMM-YYYY	date format	ASCII
@REAL	real data type	WK1
@SCIentificnotation	WK format	WK1
@SSn	social security number	ASCII, WK1
@TIME=*form*	time internal or packed	ASCII, WK1
@ZIP	zip code	ASCII, WK1
@ZIP+4	zip+4 code	ASCII, WK1

@+/-
This modifier causes the appearance of plusses or minuses according to the value of the positive or negative integer field, as provided for in WK1 format. A value of 3 causes three plus signs to appear.

@$
@CURRency
This modifier is the WK1 currency format, which prefixes values with a dollar sign, and causes two decimal places to appear.

@%
@PERCent
@PCT
This modifier is the WK1 percent format, which causes a percent sign suffix to appear. The number of decimal places defaults to 2.

@COMMA
This modifier is the WK1 comma format, which inserts a comma every three (integer) digits in a numeric value.

@DATE={PACKED|INTERNAL|JULIAN}	This modifier indicates what format is used for fields containing date values in the dataset being exported. Micro must convert the value appropriately for use with date formats. (default: **PACKED**)
@DD-MMM	This date format prints the day in two digits, followed by a dash, and the month in three letters.
@DD-MMM-YY	This date format prints the day in two digits, the month in three letters, and the year in two digits, all separated by dashes.
@DD-MMM-YYYY	This date format prints the day in two digits, the month in three letters, and the year in four digits, all separated by dashes.
@12HH:MM	This time format prints the hour from 1 to 12, a colon, the minutes, and AM or PM.
@12HH:MM:SS	This time format prints the hour from 1 to 12, the minutes, and the seconds, separated by colons, and AM or PM.
@24HH:MM	This is the Short International time format, the hour from 0 to 23, a colon, and the minutes.
@24HH:MM:SS	This is the Long International time format, the hour from 0 to 23, and the minutes and seconds, separated by colons.
@INTeger	This modifier causes a number to appear in integer format, and to be stored as an integer, if it is small enough ($-32768 <= i <= 32767$). This is the default for numerical fields that do not have scaling functions, and are small enough.
@LABEL **@TEXT**	This modifier causes a field to be exported as a WK1 label (character) type, or as an ASCII character value.
@MM/DD	This is the Short International date format, the month as two digits, followed by a slash, followed by the day as two digits.
@MM/DD/YY	This is the Long International date format, the month, day and year as two digits, separated by slashes.
@MM/DD/YYYY	This is the "extra-Long" International date format, the month, and day as two digits, and the year as four, separated by slashes.
@MMM-YY	This date format prints the month as three letters, followed by a dash, followed by the year as two digits.

`@MMM-YYYY`	This date format prints the month as three letters, followed by a dash, followed by the year as four digits.
`@REAL`	This modifier causes a number to appear in the WK1 general real format, and to be stored as a real number. Both the screen appearance and the underlying data type are affected. Any numeric field with a division scaling function is by default a real WK1 data type.
`@SCIentificnotation`	This is the WK1 scientific notation format. Numbers appear normalized between 1 and 10, the exponent printed as '`E`' followed by a plus or minus sign, and two digits.
`@SSn`	This modifier prints a social security number in the form '`ddd-dd-dddd`', where each '`d`' is a digit, and leading zeroes are retained. The WK1 data type is a label, not a number, since there is no WK1 format corresponding to this.
`@TIME={PACKED|INTERNAL}`	This modifier indicates what format is used for fields containing time values in the dataset being exported. Micro must convert the value appropriately for use with time formats. (default: `PACKED`)
`@ZIP`	This modifier causes value to appear as a 5-digit zip code. The WK1 data type is a label, not an integer, since there is no WK1 format corresponding to this.
`@ZIP+4` `@ZIPPLUS4`	This modifier causes a value to appear as a ZIP-plus-4 code. The WK1 data type is a label, not an integer, since there is no WK1 format corresponding to this.

Description

The Export command writes selected fields from a dataset, or an entire dataset, to a file in one of several formats.

ASCII format means the field values are produced as text, one record per line, separated by a delimiter (by default a comma), with non-numeric strings enclosed in quotes. This is a standard definition for ASCII files imported by spreadsheet and other programs. The ASCII file is generated as a text file on MTS (EBCDIC) and should be downloaded as a text file, which causes conversion to ASCII.

OSIRIS format produces an OSIRIS type 5 dictionary file and a "rectangular" data file, one line with all variables for one case or record, for use with the OSIRIS IV program maintained by the

Institute for Social Research at the University of Michigan. The dictionary file is generated as *file*.i, the data file as *file*.a, where *file* is given on the Export command.

Data are binary for numeric fields. Numeric fields with division scaling are generated as double precision OSIRIS type F variables; other numeric fields are produced as type I variables of 1 to 4 bytes. One-byte type I variables in OSIRIS are unsigned; larger ones are signed. The Export command generates a 1-byte signed Micro field as a 2-byte OSIRIS type I variable. Micro 2- and 3-byte unsigned fields are generated as 3- and 4-byte type I variables, respectively. The sign bit is not used in Micro 4-byte unsigned fields (the largest value allowed is the same as the largest positive signed value).

Character fields are generated as OSIRIS type A variables, in EBCDIC. The maximum length of such a variable in OSIRIS is 999, which exceeds the Micro length, except for external fields. Such fields are generated as type A variables with length equal to the minimum of the length of the longest line in the external data file, and 999.

For Micro integer and character fields with categories, OSIRIS labels and values are generated. The `@MISSing` value is used for numeric data.

SAS[3] format generates two files, for commands and data, for the Statistical Analysis System Release 6.03. The files are named *file*.c and *file*.d, where *file* is the name given on the Export command. These files work on all versions. The commands create a temporary SAS dataset. The command file can be included in an interactive session with the SAS command `%Include '`*file*`.c'`. It should be downloaded as a text file.

The Micro dataset name is used by default, but another one may be specified with the **SASNAME=** element of the syntax. Micro names for datasets and variables are checked against SAS name syntax; a name may be shortened, and certain characters changed. A message is printed when this is done.

Micro categories are generated as SAS formats, with the name of the field used as the format name. Numeric data are generated in IBM binary, characters in EBCDIC. These formats are universal for SAS; variables are translated into native format as necessary by the local SAS version. The data file should therefore be downloaded as a binary file.

SPSSX and SPSSPC formats generate two files, for commands and data, for the Statistical Package for the Social Sciences program. As for SAS, the files are named *file*.c and *file*.d, where *file* is the name given on the Export command. SPSSX is for the IBM mainframe SPSS-X version that runs on MTS, Version 4.1. SPSSPC is for Version 4.0 of the IBM-compatible microcomputer version SPSS/PC+. The command file for both formats contains an SPSS Data List statement indicating

[3]SAS is a registered trademark of the SAS Institute, Inc. SPSS-X and SPSS/PC+ are registered trademarks of SPSS, Inc. Lotus 1-2-3 is a registered trademark of Lotus Development Corporation. Microsoft Excel is a registered trademark of Microsoft Corporation.

the data types and formats. Micro field descriptions are carried over as SPSS variable labels. Micro categories and descriptions are carried over as SPSS value labels.

In SPSS-X the Include command is used to read the command file. An SPSS-X File Handle command associates the data file with an SPSS-X handle, by default the first eight characters of the dataset name. This can be changed by using `HANDLE="`*string*`"` as shown in the Usage section. The handle may be no longer than 8 characters. In the data file, numeric data are generated in binary, of the appropriate type, and character in EBCDIC characters.

In SPSS/PC+ the Include menu, accessed from the Session Control and Info menu, is used to read the command file. The command file is an MTS text (EBCDIC) file. The data file is also a text file, with data in fixed format. Both the command and data files should be downloaded as text files, which causes conversion to ASCII. The name of the microcomputer data file is given on the Data List statement. This file defaults to the name of the MTS data file, less the dash prefix if it is temporary. A microcomputer file can also be named with the optional syntax shown, `PCFILE="`*string*`"`.

Micro field names are used for SPSS variable names, after being checked against the SPSS name syntax. Some characters may be changed, and the name may be shortened. A message is printed when this is done.

WK1 format was invented by Lotus Development Corporation, and is imported by Lotus 1-2-3, Microsoft Excel and most other microcomputer spreadsheet programs. Field values can be produced as real numbers, two-byte integers ($-32768 <= i <= 32767$), or character strings (WK1 "labels"). In practice, Micro selects the WK1 data type depending on the requirements of the field. The representation of the data type on the screen can be controlled by Export command modifiers that correspond to WK1 formats. A number representing a date or time can appear in one of several formats, for instance. A WK1 file should be downloaded as a binary file.

Times and dates can be represented in Micro in packed and internal formats. Dates can also be represented in Julian format. The Export command converts these numbers to WK1 serial date and time values, but it must know the format of the Micro value. The modifier `@DATE=` (or `@TIME=`) can be used to indicate `PACKED`, `INTERNAL` or `JULIAN`.

Micro selects the WK1 data type automatically depending on the field type and other information. Integers ($-32768 <= i <= 32767$) export as integers, other numerics as reals, and type C or type CC fields as WK1 "labels" (character strings). Labels are also produced for type UC and SC fields, if categories are required, and if `@CAT= NAME` or `ABBR` or `DESCription` is given. The `@SSN`, `@ZIP` and `@ZIP+4` modifiers cause fields to export as labels, since there are no corresponding WK1 formats. The `@INTEGER`, `@REAL` and `@LABEL` modifiers can also control the underlying data type when desired.

WK1 format does not attempt to change the default column width for the screen, which requires managing the entire screen. Labels that are too wide are displayed partially. Numeric fields that are too wide, including some date and time formats that display as character, display as asterisks. Changing the column width in the spreadsheet program will display the columns properly.

FIND Micro Command

Purpose: To create a Result set containing the records from a given dataset with certain fields having specified values.

Synonyms: None.

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

`Find IN` *dataset* `WHERE` <condition> [<log_op> <condition>] ...

where

<log_op> can be any of:

`&`	`AND`
`|`	`OR`
`;`	`ALSO WHERE`

<condition> can be any of:

```
field [IS] <rel_op> <right_side>
field [IS NOT] <rel_op> <right_side>
field = <value> [OR <value>] ...
field IS BETWEEN <value> AND <value>
field IS FROM <value> TO <value>
```

<rel_op> can be any of:

`<`	`LESS THAN`
`<=`	`LESS THAN OR EQUAL TO`
`=`	`EQUAL TO`
`<>`	`IS NOT EQUAL TO`
`MAT`	`MATCHES`
`>=`	`GREATER THAN OR EQUAL TO`
`>`	`GREATER THAN`

<right_side> can be any of:

field
<value>

<value> can be any of:

category
number
string

Examples

```
Find in Students where SIN = 123456789
```

The Result set contains the record in dataset Students where field SIN has the value 123456789.

```
Find in A where new$ > old$
```

The Result set contains the records in dataset A where the field New$ has a value greater than the field Old$.

```
find in A where state = Michigan | Illinois | Ohio
```

The Result set contains the records in dataset A where the field State has the value Michigan, Illinois, or Ohio.

```
Find in Fin-Aid where CL is Jr and Amount>300
```

The Result set contains the records in Fin-Aid where field CL has the value '`Jr`' and field Amount has a value greater than 300. Both conditions must be true for a record to be in the Result.

```
Find in Fin-Aid where CL is Jr or Amount>300
```

The Result set contains the records in Fin-Aid where field CL has the value '`Jr`' or field Amount has a value greater than 300. At least one condition must be true for a record to be in the Result.

```
F in it w name matches Jones and amt is from 100 to 500
```

The Result set contains the records (from the old Result set, `it`) where the field Name has the value Jones, and field Amt is greater than or equal to 100 and less than or equal to 500.

Settings

None.

Modifiers

None.

Description

The Find command is the data retrieval operation in Micro. It creates a Result set with records from a given dataset in which one or more field values meets a logical condition. Fields can be tested for equality, inequality, and greater than or less than relations, against given values, or against each other. Fields can be tested for equality against more than one value, and tested within a range of values. The following simple example tests a single field for a numeric value; the Result holds the record where the field has that value.

```
find in students where sin = 123456789
```

When comparing numeric values, the phrase '`IS FROM` <value> `TO` <value>' *includes* the endpoints of the interval. The data value is found if greater than or equal to the lower limit and less than or equal to upper limit. The following example finds all students born in the years 1960-1962.

```
find in students where birthdate is from 600101 to 621231
```

The phrase '`IS BETWEEN` <value> `AND` <value>' *excludes* the endpoints of the interval. The data value is found only if greater than the lower and less than the upper limit. The following example is another way of finding students born in 1960-1962.

```
find in students where birthdate is between 591231 and 630101
```

A single field can be tested against multiple values. The vertical bar in the following example means *OR*; the field name (Sin) and the relation (equality) need not be repeated.

```
find in students where sin=364075234 | 011236745 | 270982309
```

When comparing character values, strings containing embedded blanks must be enclosed in quotes, e.g., `"Jones, Reginald C."`. Capitalization in the search string must match the data value. If name data is entered as '`JONES`', finding '`Jones`' fails. Character data can be tested for equality to a string, or "matched" against a root string, as shown in the following examples.

```
find in students where name is "Hersh"
```

This command finds only records where the Name field is exactly '`Hersh`'.

```
find in students where name matches "Hersh"
```

This command finds all records where the Name field begins with '`Hersh`', including '`Hersh`', '`Hershey`', '`Hershfield`', etc., assuming the Name field is five characters or longer. In this case it does not find '`Hers`'. If the match string is longer than the field length, it is truncated to that length before the search begins. If the Name field is four characters long, and the match string is '`Hersh`', '`Hers`' is found.

When two character fields (as opposed to a character field and a string on the Find command) are being tested for equality or being `MATCH`ed, the shorter of the two field lengths is used. If field Name1 is four characters long, and field Name2 is eight long, and Name1 contains '`Hers`' and Name2 contains '`Hersh`', the command matches on four characters and puts the record in the Result set; if Name1 is twelve characters long, the comparison is done on eight, and fails. In either case, the following are equivalent:

```
find in it where name1 matches name2
find in it where name1 is name2
```

Unsigned fields are compared logically, from right to left, byte by byte, through the width of the shorter field, with the remaining bytes of the longer field compared with zeroes. Signed data are compared treating the leftmost bit of each field as a sign bit. External fields are compared like signed fields. Note that external data are not compared; the four-byte line number value stored as the external index is compared. Generally, when two fields are compared they must have the same basic type: unsigned, signed, character or external, though the lengths of the first three may differ. Signed and unsigned may be compared if both are length 4.

More than one field can be tested, by using two or more search conditions, operated on by '`AND`', '`OR`', or '`ALSO WHERE`'. Each search condition generates an intermediate Result set (Iset) to which subsequent conditions are applied. Exactly how the Isets are created and used depends on the logical operators used and their order in the command.

The following command contains two search conditions connected by an `OR`. The Result set contains all of the records from the given dataset that satisfy either condition.

`Find in` *dataset* `where` <cond1> `or` <cond2>

The *dataset* is the initial Iset, Iset0. <cond1> is applied to Iset0, yielding Iset1; <cond2> is also applied to Iset0, yielding Iset2; then Iset1 and Iset2 are combined, yielding Iset3. There are no more conditions so Iset3 becomes the Result.

Logically, `OR` maintains the scope of the search by applying additional conditions to the same Iset. The condition following an `OR` is applied to the Iset generated before the previous `AND` if there was one, otherwise it is applied to the initial Iset.

The following command contains two search conditions connected by an `AND`. The Result set contains all of the records from the given dataset that satisfy both conditions.

`Find in` *dataset* `where` <cond1> `and` <cond2>

The *dataset* is the initial Iset, Iset0. <cond1> is applied to Iset0, yielding Iset1; then <cond2> is applied to Iset1, yielding Iset2. There are no more conditions so Iset2 becomes the Result.

Logically, `AND` narrows the scope of the search by applying additional conditions to the most recent Iset. The condition following an `AND` is applied to the Iset generated by the condition preceeding the `AND`.

Care should be taken when mixing `AND`s and `OR`s in multiple condition commands. The order of operations is from left to right; parentheses are not available to affect the order. The following command shows an `OR` followed by an `AND`.

`Find in` *dataset* `where` <cond1> `or` <cond2> `and` <cond3>

The *dataset* is the initial Iset, Iset0. <cond1> is applied to Iset0, yielding Iset1; <cond2> is also applied to Iset0, yielding Iset2; then Iset1 and Iset2 are combined, yielding Iset3. <cond3> is then applied to Iset3, yielding Iset4. There are no more conditions so Iset4 becomes the Result.

The following command shows an `AND` followed by an `OR`.

`Find in` *dataset* `where` <cond1> `and` <cond2> `or` <cond3>

The *dataset* is the initial Iset, Iset0. <cond1> is applied to Iset0, yielding Iset1; then <cond2> is applied to Iset1, yielding Iset2. <cond3> is applied to Iset1 (the Iset generated before the previous `AND`), yielding Iset3; then Iset2 and Iset3 are combined, yielding Iset4. There are no more conditions so Iset4 becomes the Result.

Once an `AND` has been encountered, a subsequent `OR` can never be applied to the initial Iset unless the `ALSO WHERE` operator is used. The following command shows an `AND` followed by an `ALSO WHERE`.

`Find in` *dataset* `where` <cond1> `and` <cond2> `also where` <cond3>

The *dataset* is the initial Iset, Iset0. <cond1> is applied to Iset0, yielding Iset1; then <cond2> is applied to Iset1, yielding Iset2. Now <cond3> is applied to Iset0 (the initial Iset) yielding Iset3; then Iset2 and Iset3 are combined, yielding Iset4. There are no more conditions so Iset4 becomes the Result.

Logically, `ALSO WHERE` expands the scope of the search by applying an additional <cond> to the initial *dataset*. When mixing `AND`s and `OR`s in multiple condition commands, this difference should be kept in mind.

Since `AND` narrows the scope of the search, putting this restriction first in a Find command reduces the cost of searching a large dataset. `ALSO WHERE` can be used to "return" the search to the original set when necessary.

FULLDOC Micro Command

Purpose: To print both the general and the technical documentation for a dataset, fields in a dataset, or categories of a field.

Synonyms: FDoc

Effect on the Result: None.

Usage

FUlldoc ALL DATASETS [<option>] ...

FUlldoc [ALL BUT] <datasets> [, <datasets>] ... [<option>] ...

FUlldoc IN *dataset* [<option>] ...

FUlldoc IN *dataset* [ALL BUT] <fields> [, <fields>] ... [<option>] ...

FUlldoc IN *dataset* CATEGORYS OF *field* [ON *file*]

FUlldoc IN *dataset* *field* [ALL BUT] <categories> [, <categories>] ... [ON *file*]

where

<datasets> can be any of:

dataset
pattern

<fields> can be any of:

field
pattern
field THRU *field*

<categories> can be any of:

category
pattern
category THRU *category*

<option> can be any of:

NO CATEGORIES NOCATS FIELDS ONLY *
ON *file*

Examples

```
FullDoc ? on -docall
```

This command documents all datasets on the file -DOCALL.

```
FullDoc Students on -doc
```

This command documents dataset Students on file -DOC.

```
FDoc in Fin-Aid no cats
```

This command documents all fields in dataset Fin-Aid, omitting the dataset header and category information.

```
FD in A f1 thru f5, f10
```

This command documents fields 1-5 and 10 in dataset A, omitting the dataset header.

```
FD cats of Race in Students
```

This command documents categories of field Race in dataset Students.

Settings

None.

Modifiers

None.

Description

The Fulldoc command produces general and technical documentation for one or more datasets, or fields within datasets. When a complete dataset is documented, a header like the following is printed:

```
Fin-Aid -- Example data set -- Financial aid information   09-11-81/ILIR
        Not Scrambled     Not Destroyable     Not Replaceable
  Dictionary file = ILIR:FIN-AID#      Micro Data file = ILIR:FIN-AID
```

The header consists of the dataset name, description, creation date and owner; the scrambled, destroyable and replaceable status; and the names of associated files. When 'IN' appears on the command, the dataset header is omitted, and only fields and/or categories are documented. In any case, they are documented as shown below.

```
------------------------------------------------------------------------------------------------
F#  Field name      Abbr      Value   Description                          Type Length Scale Factor Disp
------------------------------------------------------------------------------------------------

F1  Student-ID-Num  SIN       DataReq Student Identification Number        U    4                  0

F2  Year-Awarded    Year      DataReq Year Awarded (YY format)             U    1                  4

F3  Term-Awarded    Term      DataReq Term Awarded                         UC   1                  5
        3 categories (required)
      Fall          F               1 Fall Term
      Winter        W               2 Winter Term
      Summer        S               3 Summer Term

F4  Class-Level     CL        DataReq Class Level                          UC   1                  6
        4 categories (required)
      Freshman      Fr              1 Freshman
      Sophomore     So              2 Sophomore
      Junior        Jr              3 Junior
      Senior        Sr              4 Senior

F5  Type-of-Aid     Type      DataReq Type of Aid                          UC   1                  7
        6 categories (required)
      Tuition       Tu              1 Tuition
      Room-Board    RB              2 Room and Board
      Cash          Ca              3 Cash
      Loan          Lo              4 Loan
      Work-Study    WS              5 Work Study Program
      Other         Oth             6 Other

F6  Source-of-Funds SOF       DataReq Source of Funds                      UC   1                  8
        6 categories (required)
      University    Univ            1 University
      State-Gov     SGov            2 State Government
      Federal-Gov   FGov            3 Federal Government
      Industry      Ind             4 Industry
      Foundation    Foun            5 Foundation
      Other         Oth             6 Other

F7  Amount          Amt       DataReq Amount of Award                      S    4   /     100      9

F8  Remark*         Rem*              Additional Comments                  E    4                  13
        External file = ILIR:REMARKS*
```

For fields, the `F#` through `Abbr` columns in the first line of this table are self-explanatory. The `Value` is the data required status, the `Description` the field description. The `Type` is the Micro field type, the `Length` the length in bytes. The `Scale` is the scaling function, for which the Amount field has divison (/); the `Factor` is the scale factor applied when data are entered, 100 for the Amount field. The `Disp` is the displacement of the field in bytes from the start of the record.

If a field has categories, the number and the categories required switch are shown. If categories are documented, `F#` is blank; `Name` and `Abbr` are the category name and abbreviation. `Value` is the category value, `Description` the category description. If `NOCATS` is specified, the individual category information is omitted.

GET Micro Command

Purpose: To make the datasets from the directory file on another userid available for use in Micro.

Synonyms: None.

Effect on the Result: None.

Usage

`Get` *userid* [; *userid*] ...

Examples

```
Get ILIR
```

This command gets the datasets from the directory file ILIR:Directory.

```
Get ABC1; ABC2; ABC3
```

This command gets the datasets from the directory files on the userids ABC1, ABC2 and ABC3.

Settings

None.

Modifiers

None.

Description

The Get command makes datasets from directory files on other userids available for use in Micro. The directory file on the user's own id is automatically read when Micro is run. Other directories are obtained by specifying the userid (not a file name) on the Get command. If a file named

DIRECTORY exists on the specified id and if the current id has read access, the datasets in that directory are available for use. The List command, e.g., '`List from` *userid*', lists the datasets obtained from the directory file of the userid specified.

HELP Micro Command

Purpose: To display information about Micro commands and other Micro topics.

Synonyms: ?

Effect on the Result: None.

Usage

```
Help
Help [ON file] <section> [, <section>] ...
Help [ON file] [FOR] <commands> [<section> [, <section>] ...]
        [; [FOR] <commands> [<section> [, <section>] ...] ] ...
```

where

<commands> can be any of:

command
pattern

<section> can be any of:

ALL	FULL
DESCRIPTION	ACTION
EXAMPLES	EGS
MODIFIERS	MODS
PURPOSE	
RESULT	EFFECT ON THE RESULT
USAGE	SYNTAX
SETTINGS	
SYNONYMS	

Examples

```
Help
```

This starts with the basic help menu. You can follow the menus to any topic in the Help system.

`Help usage, mods, settings`

This displays some general information about the Usage, Modifiers, and Settings sections of Micro command descriptions.

`Help Sel; Sort; Set`

This displays help menus for the Select, Sort, and Settings commands. (If '`Settings`' is specified, general information about the Settings section of the Micro command reference is printed. Enclosing it in quotes (`"Settings"`) or using the abbreviation, as here, denotes the command itself.

`Help Print mods`

This displays menus for the Print command modifiers. They are subdivided by function, corresponding to menu choice, since there are so many of them.

`Help ? on -miccmds`

This writes the command references for all of the Micro commands (they all match the pattern '`?`') on the file -MICCMDS. Page ejects are placed before each command reference so the file prints conveniently.

Settings

None.

Modifiers

None.

Description

The Help command enters the Micro Help System, which presents menus of choices on which help is available. The following menu is presented in response to '`help`' entered at the command prompt:

```
Hopefully, at least one of these topics will be of service ...
-*-
 1. Micro command list         4. Learning the ropes
 2. Micro command information  5. Having problems
 3. Information, please        6. Using the Micro Help system
-*-
Choose a Topic (or just Return to exit menu):
```

A number can be entered to select one of the topics listed. Some choices lead to further menus.

The Help command can also produce help on one or more topics directly. The topics listed in the Usage section above, '`ALL`', '`DESCRIPTION`', etc., exist for each Micro command in the Help System. '`help print all`' produces all topics for the Print command. Individual topics correspond to the sections of the Command Reference: Usage, Examples, Description, etc. All commands have at least the topics listed above, though a few have more. The text for the Help System and the Command Reference is generated from the same source; topic and content are identical.

```
   help print modifiers ; xtab description
```

This produces the Modifiers section of the Print command reference, and the Description section of the Crosstabulate command.

```
   help usage; modifiers; settings
```

This produces an explanation of the Usage, Modifiers and Settings sections for the Command Reference.

The Help System can usually be entered in response to a Micro prompt concerning a command error.

```
xtab studnets by sex
"studnets" is not a dataset.
Enter another dataset name to replace "studnets"
     (or enter HELP, LIST [pattern], CANCEL):
```

Entering '`help`' elicits:

```
Micro can't use the given dataset.  In order to continue the current
operation, Micro must ask for the name of a replacement.  You have the
following options whenever Micro asks for a replacement dataset:

   *  Enter a replacement to continue the operation.
   *  Enter a null reply (if possible) to continue the operation.
   *  Enter "CANCEL" to cancel the operation.
   *  Enter "LIST" or "LIST pattern" to see a menu of possible choices.

More information:
-*-
 1. Datasets        3. Entering a replacement  5. Entering "CANCEL"
 2. Dataset Names   4. Entering a null reply   6. Entering "LIST"
-*-
Choose a Topic (or just Return to exit menu):
```

These topics explain the requirements and effects of the various possible answers. Such a menu is usually available when Micro catches an error in a command.

JOIN Micro Command

Purpose: To create a Result dataset containing all of the fields from two datasets, usually linked together by common fields.

Synonyms: None.

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

```
Join [{ALL|MATCHING}] dataset1 [BY <fields> [, <fields>] ...] WITH
     [{ALL|MATCHING}] dataset2 [BY <fields> [, <fields>] ...]
```

where

<fields> can be any of:

field
pattern
field THRU *field*

Examples

```
Join Students by SIN with Fin-Aid
```

The Result set contains the link field, SIN, followed by the remaining fields from Students, followed by the remaining fields from Fin-Aid. Records in Students and Fin-Aid with matching SIN values are joined together in the Result. Records in Students with no matching SIN in Fin-Aid are joined with filler values for the Fin-Aid fields. Records in Fin-Aid with no matching SIN in Students are joined with filler values for the Students fields.

```
Join Students by SIN with matching Fin-Aid
```

The Result set contains the link field, SIN, followed by the remaining fields from Students, followed by the remaining fields from Fin-Aid. Records in Students and Fin-Aid with matching SIN values are joined together in the Result. Records in Students with no matching SIN in Fin-Aid are joined with filler values for the Fin-Aid fields. Records in Fin-Aid with no matching SIN in Students are not included in the Result.

`Join matching A by Ayr, Amon with matching B by Byr, Bmon`

The Result set contains the link fields from A, Ayr and Amon, followed by the remaining fields from A, followed by the remaining fields from B. Records in A with Ayr and Amon values that match the Byr and Bmon values in B are joined together in the Result. Records in A and B with no matching link fields are not included in the Result.

`Join A with B`

The Result set contains all of the fields from A followed by all of the fields from B. Every record from A is paired with every record from B. The Result is the Cartesian product of A and B.

Settings

The following setting affects the Join and RestrictAndJoin commands.

Join Settings

Group	Item	Allowable values	Initially
`JOIN:`	`GROWTH FACTOR` `GF`	$1 <= i <= 9$	3

`GROWTH FACTOR` *i*
`GF`

The `GROWTH FACTOR` setting controls the size of the Result set created by the Join command, which can be as large as the product of the number of records in each of the two sets. It may be necessary to increase this value if a "partial completion" message appears on the screen. (default: 3)

Modifiers

`@GF=`*i* This modifier is used to override the `GF` setting.

Description

Before reading this section, it may be useful to look at the Join command discussion and examples in Tutorial 2.

The Join command generates a Result dataset containing the fields from one dataset appended to the fields of another dataset. Records from the first dataset are joined with records from the second.

Link Fields

How the records are joined together is determined by several factors related to the *link fields* specified in the command. For example, SIN (the student identification number) is the link field in the following command.

```
Join matching Students by SIN with matching Fin-Aid by SIN
```

The Result contains the link field, SIN, followed by the remaining fields from Students, followed by the remaining fields from Fin-Aid. Note that it is not necessary for link fields to have the same name in both datasets. However, as in Students and Fin-Aid, it is common (and recommended). If the names are the same, the link fields need only be specified once. For example, the command

```
Join matching Students by SIN with matching Fin-Aid
```

is equivalent to the previous Join command. Although the field names need not be the same, the link fields must be compatible in type and length.

Inner and Outer Joins

The '`matching`' keyword applied to both datasets tells Micro to perform an *inner join*. The Result contains only the records where the link field values match. If there is more than one link field, the values for all of them must match. If a Students record has a SIN value matching a SIN value in a Fin-Aid record, those records are joined together and included in the Result. Records with links that don't match do not contribute to the Result. If there are no matches at all, the Result is null. Therefore, the Result of the above commands includes only the records for students who have financial aid joined with their financial aid records.

To generate a Result containing the records for all students, not just those with financial aid, change the '`matching`' keywords to '`all`' as in the command

```
Join all Students by SIN with all Fin-Aid
```

The '`all`' keyword applied to both datasets tells Micro to perform an *outer join*. The Result contains all of the records from both datasets. Where the link field values match, the records are joined as they are for the inner join. Where the link field values don't match, filler data is inserted for the "missing" non-link fields to complete the Result record. The filler data is the value for the first default category, if one exists, otherwise it is zero for numeric fields, blanks for character fields. If a Students record has a SIN value matching a SIN value in a Fin-Aid record, those records are joined together and included in the Result. If a Students record has a SIN value that doesn't match any SIN value in Fin-Aid, the Students record is joined with filler data for the Fin-Aid fields. If a Fin-Aid record has a SIN value that doesn't match any SIN value in Students, the Fin-Aid record is joined with filler data for the Students fields. Therefore, the Result of the above command includes the records for all students and for all financial aid awards, joined together where the SIN values match and joined with filler data where they don't.

To generate a Result containing the records for all students but only the matching financial aid records, use the command

```
Join all Students by SIN with matching Fin-Aid
```

The '`all`' keyword forces an outer join for Students; the '`matching`' keyword forces an inner join for Fin-Aid. Therefore, the Result of the above command includes the records for all students, joined with their financial aid records where the SIN values match and joined with filler financial aid data where they don't.

Join and RestrictAndJoin

If `ALL` or `MATCHING` are specified for both datasets then the Join and RestrictAndJoin (RAJ) commands are equivalant. If they are omitted in the Join command, `ALL` is assumed. The following commands

```
Join Students by SIN with Fin-Aid
Join all Students by SIN with all Fin-Aid
RAJ all Students by SIN with all Fin-Aid
```

are equivalent; each performs an outer join. If these keywords are omitted in the RAJ command, `MATCHING` is assumed. The following commands

```
RAJ Students by SIN with Fin-Aid
RAJ matching Students by SIN with matching Fin-Aid
Join matching Students by SIN with matching Fin-Aid
```

are equivalent; each performs an inner join.

Data Relationships

Students is an example of a dataset where the link field is also a *key field*, a field that uniquely identifies each record. There is one record in Students for each student, each with its own identifying number, SIN. Suppose there is another dataset, StudentAddr, containing the SIN and the addresses of the students in Students, one record per student. Since SIN is also a key field in StudentAddr, there exists a *one-to-one relationship* between Students and StudentAddr. A join of Students by SIN with StudentAddr yields a Result containing one record per student. For each student, one record from Students is joined with the record from StudentAddr having the same SIN.

There can be any number of financial aid records in Fin-Aid for one student, all with the same value for SIN. Since SIN is not a key field in Fin-Aid, there exists a *one-to-many relationship* between Students and Fin-Aid. A join of Students by SIN with Fin-Aid yields a Result containing one record per financial aid award. For each student, one record from Students is joined with all of the records from Fin-Aid having the same SIN.

Suppose there is another dataset, StudentDupl, containing two records for each student, both with the same SIN. Since SIN is not a key field in StudentDupl, there exists a *many-to-many relationship* between StudentDupl and Fin-Aid. A join of StudentDupl by SIN with Fin-Aid yields a Result containing two records per financial aid award. For each student, two records from StudentDupl are joined with all of the records from Fin-Aid having the same SIN. The many-to-many relationship, where the link fields are not key fields in either dataset, usually indicates a problem with one or both of the datasets.

The Cartesian Product

If no link fields are specified, each record from the first dataset is joined with each record from the second, yielding the *Cartesian product* of the two. The cardinality (number of records) of the Result is the product of the cardinalities of the two sets, which is the upper bound on the size of a Join Result.

The Cartesian product is a useful way to append new fields that have no data yet. Suppose there is a dataset, NewFields, containing some new fields to be added to Students, and that it has one record holding the initial values for these fields. Then the command

```
   Join Students with NewFields
```

generates a Result containing all of Students with the initial values for the new fields appended to each record. The cardinality is the same as for Students since NewFields contains only one record.

If the cardinality of a Cartesian product will exceed a threshold value (10,000) and the cardinality of both datasets is more than one, Micro issues a warning message before continuing the operation. This is often due to accidentally omitting the link fields in the command.

Miscellaneous

The size of the Result is determined by the size of the specified datasets, the use of the `ALL` and `MATCHING` keywords (an outer join is at least as large as an inner join of the same datasets), and the nature of the data relationships between the datasets. It is possible to create a large Result dataset with the Join command. Sometimes a "partial completion" message appears on the screen and the operation stops. If this occurs, increase the *growth factor* using the `GROWTH FACTOR` setting or the

@GF command modifier. The values for growth factor can be from 1 to 9.

Since the Join command takes data from two datasets, it is possible to have more than one field with the same name in the Result. Micro does not check for this. Fields with the second and later occurrences of this name cannot be referenced in subsequent commands until the first occurrence is renamed.

The Join command does not preserve crosstabulation. If either of the specified datasets is crosstabulated, the Result contains the Count field but its values do not reflect crosstabulation in the Result.

KEY Micro Command

Purpose: To create a Result set which is a copy of another dataset with a new or existing field holding a key value that identifies each record uniquely.

Synonyms: NUmber

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

```
Key dataset [{FROM number | STARTING WITH number | INIT = number}]
            [{BY number | INCREMENT = number | INCR = number}]
            [{AT BEGINNING | AT END}]
            [INTO field]
```

Examples

```
Key Fin-Aid
```

The Result set is a copy of dataset Fin-Aid, with a Key field, the first in the dataset, containing the number of each record in the current sorted order.

```
Key A  init = 1000  incr = 100
```

The Result set is a copy of dataset A, with a Key field of values starting at 1000 in increments of 100.

```
Key B into LineNumber from 1.000 by .001 at end
```

The Result set is a copy of dataset B, with a key field named LineNumber, containing values starting at 1, in increments of .001. The field is scaled by division; the scale factor is 1000. The key field is placed at the end of the Result set.

Settings

None.

Modifiers

None.

Description

Records in Micro datasets are unique, since Micro weeds duplicate records. The Key command is used to identify records uniquely in a positive manner, usually before operations which might cause records to lose uniqueness, such as selecting certain fields for analysis.

If not specified, the key field is called Key, with abbreviation K, and is at the start of the record. Using the '`AT END`' phrase on the command places a new key field at the end. If the dataset already contains a field named Key, permission to overwrite it is asked for. If the key field is created by the Key command, it is a four-byte integer, type U or S, depending on whether a negative initial value and/or increment have been specified. The field is scaled if the initial value or increment require it. If a key value does not fit into a four-byte field, a message is printed and the command is canceled.

If the key field is already present in the dataset, its length is unchanged, but the type and scaling may change, according to the initial value and increment. Any categories are discarded. If a key value does not fit into an existing field, a message is printed and a replacement field prompted for.

LIST Micro Command

Purpose: To print a list of Micro commands, datasets, fields, or categories.

Synonyms: None.

Effect on the Result: None.

Usage

```
List  COMMANDS [[MATCHING] pattern]
List  [#] [ACTIVE] [{TEMP|PERM}] [DATASETS] [FROM userid] [[MATCHING] pattern]
List  IN dataset [#] FIELDS [[MATCHING] pattern]
List  IN dataset [#] CATS  OF field
```

Examples

`List commands`

This command lists the Micro commands.

`list`

This command lists the available datasets.

`List datasets from ILIR matching S?`

This command lists datasets from the directory file on the ILIR userid beginning with 'S'.

`list active`

This command lists the active datasets.

`List # fields in Students`

This command lists the number of fields in dataset Students.

`List cats of CL in Fin-Aid`

This command lists the categories of field CL in dataset Fin-Aid.

Settings

None.

Modifiers

None.

Description

If Micro commands are listed, the list includes each command and its minimum abbreviation.

The List command can also determine what datasets are available, or active. A permanent dataset is available if the directory file containing it has been obtained with the Micro Get command. A permanent dataset is active if it has been used in a Micro command. A temporary dataset is always active.

Fields and categories, or simply counts of the same, can also be listed for a particular dataset.

Note that patterns can be used in listing commands, datasets, fields and categories. This is how you find a dataset name, or field or category name, during a busy Micro session when you don't recall the exact dataset or field or category for a particular operation.

LOCK Micro Command

Purpose: To lock a dataset's data and/or dictionary files in order to restrict access to other users and to prevent file deadlocks.

Synonyms: None.

Effect on the Result: None.

Usage

```
LOck   <datasets> [<what>] [, <datasets> [<what>] ] ... [<how>]
    [; <datasets> [<what>] [, <datasets> [<what>] ] ... [<how>]] ...
```

where

<datasets> can be any of:

dataset
pattern

<what> can be any of

```
DICT
DATA
ALL
```

<how> can be any of

```
TO READ
TO MODIFY
TO DESTROY
```

Examples

`Lock Fin-Aid84, Fin-Aid85, Fin-Aid86`

This command locks to modify the data for the Fin-Aid84, Fin-Aid85 and Fin-Aid86 datasets.

`Lock Budget?`

This command locks to modify the data for all datasets whose names begin with '`Budget`'.

```
Lock to read a, b?, c ; xx dict, yy data, zz all to modify
```

This command locks to read the data for the datasets A, C, and those whose names begin with 'b' and it locks to modify XX's dictionary, YY's data, and ZZ's dictionary and data.

Settings

None.

Modifiers

None.

Description

Micro commands lock a permanent dataset's dictionary and/or data files according to the operations they perform. The Replace, Enterdata Perm, and Update Perm commands lock the data file to modify, and the dictionary file to read. The Replace command locks the dictionary to modify, if the dataset structure has changed. The commands that produce a Result set lock dictionary and data files to read. The documentation commands (Document, Describe, List, etc.) lock only the dictionary file to read, and that only if fields or categories are referred to. The Destroy command locks both dictionary and data files to destroy. Locking is not a consideration with temporary datasets, since they are accessible only to one Micro session.

The need for explicit file locking is shown by the command sequence:

```
Change in Students ...
Change in It ...
Change in It ...
    ...
Replace Students with It
```

It is possible for another user to read the dataset during the Change commands, since Change locks only to read. The Replace command could not proceed since the other user would have the files locked. Using '`Lock Students`' before the first Change prevents this by locking the data file to modify. Other users who attempt to lock the data file, implicitly or explicitly, get a message saying the command is canceled because the data file is already locked.

Dataset Locking by Micro Commands

Command	Data	Dict	Comment
Calculate	R	R	
Change	R	R	
Combine	R	R	
Comment			
Xtab	R	R	
Datestamp	R	R	
Describe		R	
Destroy	D	D	destroys dictionary file only if not shared
Display			
Document		R	only if fields or categories are documented
Enterdata	R/M	R	locks data file to modify if 'perm' specified; unlocks both when done
Export	R	R	
Find	R	R	
Fulldoc		R	only if fields or categories are documented
Get			
Help			
Join	R	R	
Key	R	R	
List		R	only if fields or categories are listed
Lock	M		default behavior; either file can be locked to read, modify or destroy
Message			
MTS			
Name		R	only if fields or categories are renamed
Print	R	R	
Purge	U	U	all available datasets
Read	R	R	locks specified datafile to read
Release	U	U	those datasets specified
Remove	R	R	
Replace	M	R/M	locks dictionary file to modify if dataset structure changed, else to read; unlocks both when done
Reset			
Restrict	R	R	
RAJ	R	R	
Save	M	M	unlocks both when done
Select	R	R	
Settings			
Sort	R	R	
Stop	U	U	unlocks all locked files
Techdoc		R	only if fields or categories are documented
Update	R/M	R	locks data file to modify if 'perm' specified; unlocks both when done
Write	M	R	locks specified data file to modify; unlocks it when done

The above table summarizes the locking effects of Micro commands: D means destroy, M lock to modify, R lock to read, U unlock, and R/M Read or Modify, as explained.

There is no Unlock command. The Release command releases the files associated with a dataset, thus unlocking them. A Micro command that makes permanent changes to a dataset releases its files when done. Purge and Stop release the files associated with all active datasets.

MESSAGE Micro Command

Purpose: To print a message on a file or device.

Synonyms: None.

Effect on the Result: None.

Usage

`MEssage "`*string*`" [ON `*file*`] [; "`*string*`" [ON `*file*`] ] ...`

If 'ON *file*' is omitted, *SINK* is used. This is usually the terminal.

Examples

```
Message  " Starting BudCheck MPF function ..."
```

This command prints the message on *SINK*.

```
Message  -
     "-***  Error while processing budget data  ***" on *MSINK* ;  -
     " Error in BudCheck function:  null result" on MicroLog(*L+1)
```

This command prints the first message on *MSINK*, and the second to the end of the MicroLog file.

```
Message  -
     "0During data entry:" ;  -
     "    - use category names and abbrs rather than values" ;  -
     "    - remember to enclose data for the Name field in primes" ;  -
     "    - long text fields can be continued on the next line" ;  -
     "          by typing a hyphen '-' then pressing Return"
```

This command prints the messages on *SINK*, one per line.

Settings

Messages are subject to the `Cmd: OutLen` setting. Messages longer than this value are broken into separate lines.

Modifiers

None.

Description

The text is written to the file or device with the first character used as a logical carriage control. If the text is longer than the `Cmd: OutLen` setting, it is broken into separate lines. Lines after the first are supplied with a blank carriage control.

Note: If the text is going to the terminal, the device support parameters will affect how it appears.

The Message command is mostly used in runfiles, command files and MPF commands to send messages to the user such as instructions, progress, indications, bulletins, and error messages.

MTS Micro Command

Purpose: To leave Micro temporarily and return to MTS command mode.

Synonyms: SYstem

Effect on the Result: None.

Usage

```
MTs
```

Examples

```
MTS
System
```

Both of these examples return the user to MTS. Micro may be reentered by issuing '`$Restart`'.

Settings

None.

Modifiers

None.

Description

The MTS command allows you to go into MTS command mode and then return to Micro using the $Restart command. All datasets and settings remain unchanged, unless Micro is unloaded, explicitly or implicitly, with a $Run, $Rerun, $Debug, $Load, or $Unload command.

NAME Micro Command

Purpose: To change temporarily dataset names, field names and abbreviations, and category names and abbreviations.

Synonyms: REname

Effect on the Result: The most common use of the Name command is to change the name of a Result dataset to something other than "Result" or "It". This keeps it from being replaced by a new Result.

A temporary dataset can be made the Result dataset by changing its name to "Result" or "It" (providing that a Result dataset does not already exist). A permanent dataset may not be named "Result" or "It".

The Name command can temporarily change the name or abbreviation of any field or category in any dataset, including those in the Result.

Usage

`Name` <name_operation> [; <name_operation>] ...

where

<name_operation> can be any of:

dataset [`AS`] *new_dataset_name*

`IN` *dataset field* [`ABBR`] [`AS`] *new_field_name*
[, *field* [`ABBR`] [`AS`] *new_field_name*] ...

`IN` *dataset field category* [`ABBR`] [`AS`] *new_cat_name*
[, *category* [`ABBR`] [`AS`] *new_cat_name*] ...

Examples

```
Name it A
```

This command changes the Result dataset name to A, which keeps it from being replaced by a new Result.

```
Name FY85 FY85old; Result FY85
```

This changes the FY85 dataset name to FY85old, and the Result dataset name to FY85, thus keeping the old FY85 dataset and establishing a new one.

```
Name in FY85 LName as Last.Name, Salary abbr Slry
```

This changes the LName field name to Last.Name and the Salary field abbreviation to Slry in the FY85 dataset.

```
Name in FY85 Month Jan as January, Feb abbr Febr
```

This changes the Jan category name to January and the Feb category abbreviation to Febr for the Month field in the FY85 dataset.

```
Name FY85 FY85old; Result FY85; -
     in FY85 LName as Last.Name, Salary abbr Slry; -
     in FY85 Month Jan as January, Feb abbr Febr
```

This command does what the previous three examples do. The Result is already named FY85 when the field and category names and abbreviations are renamed. The changes are made in the new FY85, not in FY85old.

Settings

None.

Modifiers

None.

Description

The Name command temporarily changes dataset names, field names and abbreviations, and category names and abbreviations. The changes are temporary in that they last only for the duration of the Micro session. The Save (or Replace) command must be used to make such changes permanent.

The following restrictions apply:

- A new dataset name must be distinct from the names of all other datasets.
- A new field name or abbreviation must be distinct from the names and abbreviations of all other fields and all categories in the same dataset.
- A new category name or abbreviation must be distinct from the names and abbreviations of all fields in the same dataset, and must be distinct from the names and abbreviations of all

other categories in the same field.

- Any new name must follow the rules for Micro names (with respect to maximum length and legal characters, as explained in the section on Dataset Structure).

If there is any problem with a new name, a message is printed, and Micro asks for a replacement. Canceling the command here cancels the current rename operation and all those following it in the command.

PRINT Micro Command

Purpose: To print the records from a dataset.

Synonyms: None.

Effect on the Result: None.

Usage

`Print` [`FOR` <format>] [`ON` *file*] *dataset*

`Print` [`FOR` <format>] [`ON` *file*] `IN` *dataset* [`ALL BUT`] <fields> [, <fields>] ...

`Print IN` *dataset* `COUNT`

where

<format> can be one of:

```
DCA
RTF
TEXH      TEX
TEXS
TEXTEDIT
```

<fields> can be any of:

field
pattern
field `THRU` *field*

The first prototype prints data for all fields in the dataset, the second prints the specified fields, and the third prints only the number of records in the dataset.

Examples

`Print Students`

This example prints the dataset Students as a table, using records as rows and fields as columns, with default settings.

`Print in Fin-Aid count`

This example prints only the number ('`count`') of records in dataset Fin-Aid.

```
Print@strat=1 in it Sex, Race@cat=desc@width=12, tot.Amt, ave.Amt
```

This example prints, from the Result set ('`it`'), the fields Sex, Race, tot.Amt and ave.Amt. The first of these is printed on a line by itself ('`strat=1`', the first field in vertical layout). The field Race, which is categorical, is printed as the category description ('`@cat=desc`'), up to a maximum of 12 characters wide ('`@width=12`'), followed by the remaining fields.

```
Print@nocc@noheading@#lines=1@#spaces=0 on -fixfmt in A   -
    F1@width=5, F2@width=10, F3 thru F10@value@width=2,   -
    F11@cat=abbr@width=4
```

This example generates fixed format data, on the file -FIXFMT, from dataset A. Command modifiers control the general format of the output. Carriage controls ('`@nocc`') and column headings ('`@noheadings`') are suppressed, each cell has 1 line ('`@#lines=1`'), and there are no spaces to the left of cells ('`@#spaces=0`').

Field modifiers control the format of the each cell. Field 1 has a width of 5 ('`F1@width=5`'), Field 2 has 10 ('`F2@width=10`'). For Fields 3 through 10, the value is printed, in a of width 2 ('`@value@width=2`'). For Field 11, the category abbreviation is printed, in a width of 4 ('`F11@cat=abbr@width=4`').

```
Print@title="Students 1991"@align=center for rtf students on -stu
```

This example prints dataset Students, with title '`Students 1991`', centered ('`@align=center`') in RTF format, on the file -STU, for downloading to Microsoft Word on a microcomputer.

Settings

The settings in the following table affect the Print command. Not all Print settings apply to text processing formats, as noted in the last column.

Print Settings

Group	Item	Allowable values	Initially	Text Format
PRINT:	CATEGORY CAT	NAme, ABbr, DEsc, or Value	NAme	×
PRINT:	DIRECTION	Forward or Reverse	Forward	×
PRINT:	FIELD FLD	NAme, ABbr, DEsc, or OFF	NAme	×
PRINT:	FORWARD	ON or OFF	ON	×
PRINT:	HEADER CC HCC	carriage control	'0'	
PRINT:	HORIZONTAL HORIZ	ON or OFF	ON	×
PRINT:	HIRCC	carriage control	'␣'	
PRINT:	LINES/PAGE LPP	positive integer	60	×
PRINT:	OUTLEN	#chars	79 (chars)	
PRINT:	PAGE CONTROL PC	ON or OFF	OFF	
PRINT:	PAGE CC PCC	carriage control	'1'	
PRINT:	PDATE	*string* or OFF	OFF	
PRINT:	REVERSE	ON or OFF	OFF	×
PRINT:	TITLE	*string* or OFF	OFF	×
PRINT:	TITLE CC TCC	carriage control	'0'	
PRINT:	UNDERLINE	character or OFF	'_'	
PRINT:	VERTICAL VERT	ON or OFF	OFF	×
PRINT:	VIFCC	carriage control	'␣'	
PRINT:	VIRCC	carriage control	'0'	

CATEGORY {NAme|ABbr|DEsc|Value}
CAT

The **CATEGORY** setting determines how categorical data are printed, as category names, abbreviations, descriptions or values. (default: **NAme**)

DIRECTION {**Forward**|**Reverse**}	The **DIRECTION** setting determines the order in which records are printed, from first to last, or last to first. Records are stored in a sorted order at all times; the key used in the sort is the entire record. If records begin with a numeric field, the records with the smallest values appear first. If records begin with a character field, records appear in alphabetical order. The Micro Sort command can be used to rearrange the fields and, therefore, reorder the records. This sorted order is the sequence in which records are printed, first to last or last to first. (default: **Forward**)
FIELD {**NAme**|**ABbr**|**DEsc**|**OFF**} **FLD**	The **FIELD** setting determines how headings for columns (fields) are printed. (default: **NAme**)
FORWARD {**ON**|**OFF**}	The **FORWARD ON** setting is the same as **DIRECTION FORWARD**, as discussed above. (default: **ON**)
HEADER CC *c* **HCC**	The **HCC** setting affects the carriage control on the line containing column headings. (default: '0')
HORIZONTAL {**ON**|**OFF**} **HORIZ**	The **HORIZONTAL** setting causes records to be printed as rows in a table, with fields corresponding to columns. (default: **ON**)
HIRCC *c*	The **HIRCC** setting sets the carriage control on table rows in horizontal format. This controls spacing between lines containing records on output. (default: '␣')
LINES/PAGE *i* **LPP**	The **LPP** setting determines the number of lines on a page. In horizontal format (one row per record) one record contributes one line (unless the fields wrap in their columns). For RTF and DCA text processing formats, in horizontal format, all lines contributed by records (including fields that wrap) are included, as are lines resulting from a title and column headings. TEXH allows TEX to do the wrapping, and each record is treated as one line. When printing in horizontal format for TEXTEDIT, **LPP** is ignored; the program generates page breaks automatically. TEXTEDIT also wraps character fields. Grouping records on one page is also possible with the **RPP** (records per page) command modifier. (default: 60)
OUTLEN *i*	The **OUTLEN** setting controls the length of output lines, in number of characters. If this length is more than the sum of column widths, a message is printed, and lines are wrapped. (default: 79)

PAGE CONTROL {**ON**|**OFF**} **PC**	The **PC** setting enables page control. If this setting is on, every **LPP** lines a line begins with the page control character (**PCC**). If **LPP** has not been set, the default value (60) is used. (default: **OFF**)
PAGECC *c* **PCC**	The **PCC** setting supplies a page control character to be output when the **PC** setting is '**ON**', as discussed above. (default: '1')
PDATE {*string*|**OFF**}	The **PDATE** setting provides a string printed at the top left of each page, if the **@PD** modifier is used on the Print command. This string defaults to the current date, where the month is a capitalized three-letter abbreviation. (default: **MMM DD, YYYY**)
REVERSE {**ON**|**OFF**}	The **REVERSE ON** setting is the same as **DIRECTION REVERSE**, as discussed above. (default: **OFF**)
TITLE {*string*|**OFF**}	The **TITLE** setting provides a title for the output, which is printed at the top left of the page if page control is enabled. For text processing formats, inserting '`\n`' causes subsequent text to appear on a new line, e.g., '`@title="Financial Aid\nSenior Undergraduates\nUniversity of Michigan"`' prints a three-line title. When printing in horizontal format for TEXTEDIT, the first line of the title is generated as a continuation title for replication at the top of subsequent pages. (default: **OFF**)
TITLE CC *c* **TCC**	The **TCC** is the title carriage control character, which appears at the start of the line containing the title (provided by the **TITLE** setting). (default: '**0**')
UNDERLINE {*c*|**OFF**}	The **UNDERLINE** setting provides a character which is used to draw a line between the column headings (the field names, or abbreviations, etc.) and the first record. If this setting is turned off, printing of the line is suppressed. If the **UNDERLINE** character is the underscore and normal Print command output (not text processing output) is directed to a file, a '+' carriage control is used to suppress the line feed on the MTS printers, producing underlined column headers. (default: '–')
VERTICAL {**ON**|**OFF**} **VERT**	The **VERT** setting prints the records in vertical format. Each line contains one field name or abbreviation followed by the field value for the record. (default: **OFF**)

VIFCC *c*	The **VIFCC** setting controls the spacing between fields (lines) when records are printed in vertical format. (default: '␣')
VIRCC *c*	The **VIRCC** setting controls spacing between records (groups of lines containing one field per line) when output is in vertical format. (default: '0')

Modifiers

Print Modifier Introduction

Modifiers for the Print command are of two types, command and field modifiers. Command modifiers apply to the overall behavior of the command; they affect the general format of the output, or the format of all fields. Some command modifiers are used to override the corresponding Print settings. Field modifiers apply to individual fields. They override any Print settings and command modifiers for the fields to which they are applied.

Also, some modifiers have an effect only on the text processing formats; others are ignored when printing for these formats.

The following table summarizes the Print modifiers by functional group, indicates their corresponding settings, whether they are command or field modifiers, and whether they apply to text processing formats.

Print Command Modifiers

Function	Setting Parameter	Modifier	Command Modifier	Field Modifier	Text Processing
		Page Control			
title	`TITLE` *string*	`@TITLE=`*string*	×		×
lines per page	`LPP` *i*	`@LPP=`*i*	×		×
no carriage ctrl		`@[NO]CC`	×		
page# and date		`@PAD[=`*p*`]`	×		
begin on new page		`@Page`	×		
enable page control	`PC ON`	`@PC`	×		
		Record Control			
# records printed		`@Count=`*n*	×		×
# of first record		`@FIrst=`*f*	×		×
order of printing	`FORWARD ON`	`@Forward`	×		×
ith record printed		`@Increment=`*i*	×		×
reverse print order	`REVERSE ON`	`@Reverse`	×		×
		General Layout			
fields on one line	`HORIZONTAL ON`	`@Horizontal`	×		×
left margin		`@LEFTMargin=`*i*`[`*unit*`]`	×		
no column headings		`@[NO]HEADings`	×		×
length of output lines	`OUTLEN` *n*	`@OUTLEN=`*n*`[`*unit*`]`	×		
some fields one line, some on separate lines		`@STratification=`*i*	×		×
fields on separate lines	`VERTICAL ON`	`@Vertical`	×		×
characters/inch on screen		`@PITCH=`*i*	×		
char for rules	`UNDERLINE` *c*	`@UNDERLINE=`*c*	×		
		Cell Parameters			
position of cell		`@COLumn=`*c*`[`*unit*`]`		×	
# lines in cell		`@#LINes=`*y*	×	×	×
# spaces to left of cell		`@#SPaces=`*s*`[`*unit*`]`	×	×	×
width of cell		`@WIDth=`*x*`[`*unit*`]`	×	×	×
fill character		`@FILL=`*c*	×	×	
		Item Format			
column headings	`FIELD` *form*	`@FIELD[=`*form*`]`	×	×	×
category format	`CATegory` *form*	`@CATegory[=`*form*`]`	×	×	×
print part of item		`@LENgth=`*n*	×	×	×
print hexadecimal form		`@HExadecmial`	×	×	×
print external value		`@VALue`	×	×	×
		Data Formats			
dollar sign prefix		`@$`		×	×
integers as `'MM-DD-YY'` dates		`@Date`		×	×
integers as Social Security numbers, `'xxx-xx-xxxx'`		`@SSn`		×	×
integers as US zipcodes (with leading zeroes)		`@ZIPcode`		×	×
integers as US Zip+4 codes, `'iiiii-jjjj'`		`@ZIP+4`		×	×

Print Modifiers (continued)

Function	Setting Parameter	Modifier	Command Modifier	Field Modifier	Text Processing
		Cell Alignment			
center within cell		**@CENTered**	×	×	×
left-justify in cell		**@LEFTJustify**	×	×	×
right-justify in cell		**@RIGHTJustify**	×	×	×
		Text Processing Formats			
table alignment, l-r-c		**@ALIGNment={Left|Center|Right}**	×		×
left border of table cell (RTF)		**@LEFTBORDer={THIN|THICK|DOUBLE|DOT|SHADOW}**	×	×	×
right border of table cell (RTF)		**@RIGHTBORDer={THIN|THICK|DOUBLE|DOT|SHADOW}**	×	×	×
top border of table cell (RTF)		**@TOPBORDer={THIN|THICK|DOUBLE|DOT|SHADOW}**	×	×	×
bottom border of table cell (RTF)		**@BOTTOMBORDer={THIN|THICK|DOUBLE|DOT|SHADOW}**	×	×	×
left margin (RTF and DCA)		**@LEFTMARgin=***i*[*unit*]	×		×
right margin (RTF and DCA)		**@RIGHTMARgin=***i*[*unit*]	×		×
top margin (RTF and DCA)		**@TOPMARgin=***i*[*unit*]	×		×
bottom margin (RTF and DCA)		**@BOTTOMMARgin=***i*[*unit*]	×		×
# records per page		**@RPP=***i*	×		×
portrait (default) text orientation (parallel to width of paper)		**@PORTrait**	×		×
landscape text orientation (parallel to length of paper)		**@LANDscape**	×		×
lines around cells		**@[NO]RULEs**	×		×
set point size for nominal char size for TEXS and RTF		**@POINTsize=***i*	×		×
TeX glue for `\halign`		**@TABSKIPglue=** *TeX value*	×		×
TeX hsize for `\halign`		**@HSIZE=** *TeX value*	×		×

Page Control

The Page Control modifiers allow the number of lines on a page and related parameters to be set. *Page control* is "enabled" or "in effect" when the **PC** setting is on, or if any of the **@PC**, **@TITLE**, **@LPP**, or **@PAD** modifiers is given. When page control is enabled, the Page Control Character (**PCC**) is printed to start a new page whenever **LPP** lines have been printed. A title can be printed as well as the page number and date. If page control in not in effect, the @PAGE modifier can be used to start the output on a new page by printing the PCC only on the first line. Page control modifiers are command modifiers only.

@TITLE {*string*|OFF}	The *string* is printed at the top left of the page and page control is enabled. @TITLE=OFF omits a title for this command. For text processing formats, inserting '\n' causes subsequent text to appear on a new line, e.g., '@title="Financial Aid\nSenior Undergraduates\nUniversity of Michigan"' prints a three-line title. When printing in horizontal format for TEXTEDIT, the first line of the title is generated as a continuation title for replication at the top of subsequent pages. (default: OFF)
@LPP=*i*	*i* lines per page are printed and page control is enabled. In horizontal format (one row per record) one record contributes one line unless fields wrap in their column widths. For RTF and DCA text processing formats, in horizontal format, all lines contributed by records (including fields that wrap) are included, as are lines resulting from the title and column headings. TEXH allows $\TeX$ to do the wrapping, and each record is treated as one line. When printing in horizontal format for TEXTEDIT, LPP is ignored; the program generates page breaks automatically. TEXTEDIT also wraps character fields. Grouping records on one page is also possible with the @RPP (records per page) command modifier. (default: 60)
@NOCC @¬CC @-CC	The carriage control column (column 1) is not printed. This is useful for generating fixed format data on a file in conjunction with the @NOHEADings command modifier and the @COLumn field modifier. Page control is disabled. (default: @CC)
@PAD[=*p*] @PD[=*p*]	The page number and date are printed at the top of each page. If *p* is specified, page numbering begins with *p*. Page control is enabled. (default: 1)
@Page	The first character of the first line printed is the Page Carriage Control character (PCC), if page control is disabled.
@PC	Page control is enabled; every LPP lines, the Page Control Character (PCC) is printed to start a new page.

Record Control

Record control modifiers affect the number of records printed and the direction (forward or reverse) in the dataset. Record control modifiers are command modifiers only.

@Count=*n*	A maximum of *n* records is printed. The *cardinality* of the dataset, the number of records, is the default value for `Count`.
@FIrst=*f*	The first record printed is record *f*. If not specified, *f* is 1 when printing in the forward direction, and *f* is the cardinality when printing in the reverse direction.
@Forward	The records are printed in the forward (default) direction.
@Increment=*i*	Every *i*th record is printed. If $i < 0$, records are printed in the reverse direction. (default: 1)
@Reverse	The records are printed in the reverse direction.

General Layout

General Layout modifiers affect the overall layout of the output. Records can be printed in horizontal (or tabular) layout with field values arranged horizontally in rows, or vertically with one field value per row. The vertical layout is useful when printing fields containing long character strings, or when there are so many fields to print that the horizontal layout would force records to wrap around. These layouts can be mixed, with some fields printed vertically, others horizontally. A left margin can be specified. Column headings can be suppressed for generating data in fixed format. The maximum length for an output line can be set. General Layout modifiers are command modifiers only.

In the discussion below *n*[*unit*] refers to a number followed by a unit or dimension. The unit can be characters, points, inches, centimeters, millimeters, or twips (twentieths of a point). For example, `12char`, `72pt`, `1in`, `2.54cm`, `25.4mm`, `1440twip`. If the unit is omitted, it is assumed to be characters.

@Horizontal	All fields are printed in the horizontal, or tabular, layout (the default), one record per row with the fields printed left to right in the order specified in the command.
@LEFTMARgin=*i*[*unit*] **@LMARgin=*i*[*unit*]**	A left margin of *i units* is used. (default: 0)
@NOHEADings **@~HEADings** **@-HEADings**	The column headings and underlines (in the horizontal layout) are not printed. This is useful for generating fixed format output on a file in conjunction with the **@NOCC** command modifier and the **@COLumn** field modifier. (default: **@HEADings**)
@OUTLEN=*n*[*unit*]	An output length of *n units* is used.

`@STratification=`*i*	The first *i* fields specified are printed in the vertical layout; the remaining fields are printed in the horizontal layout. (default: 0)
`@Vertical`	All fields are printed in the vertical layout, one field per row with the field name (or abbreviation, etc., according to the `@FIELD` modifier or `FIELD` setting) followed by the field item.
`@PITCH=`*i*	This value is the pitch (number of characters per inch) of an output device. It allows measurements specified in units other than characters (points, inches, etc.) to be approximated on a fixed pitch device. For example, if the output is directed to a character mode screen with 12 characters per inch, `@PITCH=12` should be specified. This translates a column width of, say, 2 inches (`@WIDth=2in`) into 24 characters on the screen. Thus Print command output normally destined for a text processor can be previewed on the screen (by omitting "`FOR` <format> `ON` *file*" in the command), at least for things like line breaks, before printing. (default: 10)
`@UNDERLINE=`{*c*|`OFF`}	This modifier determines the character used in the horizontal layout to draw a line between the column headings (the field names, or abbreviations, etc.) and the first record. If `@UNDERLINE=OFF`, printing of this line is suppressed. If `UNDERLINE` is the underscore and normal Print command output (not text processing output) is directed to a file, a '+' carriage control is used to suppress the line feed on the MTS printers, producing underlined column headers. (default: '–')

Cell Parameters

A *cell* refers to a row-column location in the printout that will contain an *item*, the printed representation of a value from a dataset record. The horizontal position and width of a cell can be specified as well as its vertical height (number of lines). Fixed format output can be produced if spacing, carriage controls, and headings are suppressed. Cells can be preceded by spaces, and can be filled by default with a fill character. With one exception, Cell Parameters are both command and field modifiers.

`@COLumn=`*i*[*unit*]	This is a field modifier only. The cell begins in column *i*, or at a distance *i units* from the left margin. This is useful for generating fixed format output on a file in conjunction with the `@NOCC` and `@NOHEADING` command modifiers. Take care when using this modifier: it is possible to overlap another cell, and the interaction with any wrap-around may not print what you had in mind.

@HEIght=*y* @#LINes=*y*	Each cell is forced to have *y* lines. If an item does not fit into *y* lines, it is truncated; if an item does not fill *y* lines, the cell is padded with blank lines. If not specified, the HEIght is as many lines as are necessary to hold the item, which varies with the LENgth and WIDth. Note: You can achieve some efficiency by using '@height=1' as a command modifier where appropriate. This avoids the line-break mechanism used to wrap items around within cell widths.
@SPaces=*s*[*unit*] @#SPaces=*s*[*unit*]	There are *s units* to the left of each cell. If not specified, *s* is 2 characters for items that are left-justified in the cell and 1 otherwise.
@WIDth=*x*[*unit*] @#COLumns=*x*[*unit*]	Each cell is forced to have a width of *x units*. If an item does not fit into *x units*, it is broken into lines; if an item does not fill *x units*, the cell is filled with the FILL character. If not specified, WIDth is 20 characters for text items (category descriptions, character fields, and external text), the LENgth for non-text (other) items.
@FILL=*c*	The cell is filled with this character before the item is inserted. (default: blank)

Item Format

An *item* is a representation of a field value from a record (or the field heading, which heads the column in horizontal layout, the row in vertical layout). Item Format modifiers determine the format of cell contents. Field headings can be field names, abbreviations, descriptions, or empty. Categorical data can have category names, abbreviations, values, or descriptions printed for field values. A maximum length for items can be set. An item can be printed in hexadecimal. The Item Format modifiers are command and field modifiers.

@FIELD[=*form*] @FLD[=*form*]	This affects the form of the column headings. The *form* can be NAmes, ABbreviations, DEscriptions, or OFF. (default: NAme)
@CATegory[=*form*]	This affects the form of a category item. The *form* can be NAmes, ABbreviations, DEscriptions, or VAlues. (default: NAme)
@LENgth=*n* @#CHaracters=*n*	Only the first *n* characters of each item are printed. This is a way to limit the length of any item. (default: length of the item, based on the field type and length, and on @CAT form.)
@HExadecimal @X	The hexadecimal representations of the values are printed. For character fields, this is the EBCDIC code. For numeric fields, this is the internal integer form of the value.

`@VALue`	This is the same as `@CAT=VALUE` but is included for external fields, which have no categories. The "value" for an external field is the four-byte item (usually an MTS line number), rather than the character string.

Data Formats

Data Format modifiers control the representation of data values. Values can be represented as dollar amounts, printed in date form, as social security numbers, or zipcodes. Data Format modifiers are field modifiers only.

`@$`	Numeric values appear with a dollar sign.
`@Date`	Numeric values stored in the record in YYMMDD form appear in '`MM-DD-YY`' form.
`@SSn`	Numeric values appear in standard Social Security Number form, '`xxx-xx-xxxx`'.
`@ZIPcode`	Numeric values appear in standard U.S. ZipCode form. For example, the value 1234 appears as '`01234`'. This will not handle Canadian zipcodes.
`@ZIP+4`	Numeric values appear in standard U.S. Zip+4 form. For example, the value 12345678 appears as '`01234-5678`'. This will not handle Canadian zipcodes.

Note that these data formats only affect numeric values; they do not apply to category names or abbreviations. This allows you to have Missing or NotApplicable categories print as such, not as data-formatted values. If needed, the `@CAT=VALUE` modifier can be used to override this.

Cell Alignment

Cell Alignment modifiers affect the position of an item within a cell. Items can be centered, or left- or right-justified. Cell Alignment modifiers are command and field modifiers.

`@CENTered`	Each item is centered within its cell. Use `@WIDth` with this when printing in the vertical layout.
`@LEFTJustify` `@LJustify`	Each item is left-justified within its cell (text items are left-justified by default).
`@RIGHTJustify` `@RJustify`	Each item is right-justified within its cell (non-text items are right-justified by default). Use `@WIDth` with this when printing in the vertical layout.

Text Processing Formats

These modifiers apply to some or all of the five text processing formats in which a dataset can be generated as a table. The formats are IBM's DCA (Document Content Architecture) for IBM wordprocessing products, and those which import that format; Microsoft's RTF (Rich Text Format) for Microsoft wordprocessing products, and any which import that format; two formats for TEX, the typesetting program used in academic circles; and STAT:TEXTEDIT, the text processing program that runs on MTS.

@ALIGNment={Left|Center|Right}	The alignment applies to the table title, if given, as well as the table itself. The table is printed flush left on the page, centered, or flush right. (default: **Left**)
@LEFTBORDer={THIN|THICK|DOUBLE|DOT|SHADOW} **@LBORDer**	This modifier sets the left cell border for RTF format. (default: **THIN**)
@RIGHTBORDer={THIN|THICK|DOUBLE|DOT|SHADOW} **@RBORDer**	This modifier sets the right cell border for RTF format. (default: **THIN**)
@TOPBORDer={THIN|THICK|DOUBLE|DOT|SHADOW} **@TBORDer**	This modifier sets the top cell border for RTF format. (default: **THIN**)
@BOTTOMBORDer={THIN|THICK|DOUBLE|DOT|SHADOW} **@BBORDer**	This modifier sets the bottom cell border for RTF format. (default: **THIN**)
@LEFTMARgin=*i*[*unit*] **@LMARgin**	This modifier sets the left margin for RTF and DCA formats. DCA always uses 10 characters per inch pitch, and all sizes are translated into characters on that basis. In RTF, which has proportional fonts, if the *units* are omitted, a nominal character size in twips is used. Note: This modifier also affects normal (non text-processing) output. (default: 1in)
@RIGHTMARgin=*i*[*unit*] **@RMARgin**	This modifier sets the left margin for RTF and DCA formats. DCA always uses 10 characters per inch pitch, and all sizes are translated into characters on that basis. In RTF, which has proportional fonts, if the *units* are omitted, a nominal character size in twips is used. (default: 1in)
@TOPMARgin=*i*[*unit*] **@TMARgin**	This modifier sets the top margin for RTF and DCA formats. If *unit* is not given, a number of characters is assumed, which is multiplied by a nominal line size for RTF. (default: 1in)
@BOTTOMMARgin=*i*[*unit*] **@BMARgin**	This modifier sets the bottom margin for RTF and DCA formats. If *unit* is not given, a number of characters is assumed, which is multiplied by a nominal character size for RTF. (default: 1in)

@RECORDSPERPAGE=*i* **@RPP=*i***	If this value is given, a page-eject control sequence for the appropriate text processing format is inserted every *i* records. This, rather than **@LPP** (lines per page), is the preferred way of ensuring that records are not split across pages when printing for a text processing format. If given, the **TITLE** value is printed at the top of each page.
@PORTrait	This modifier causes text to appear in portrait mode, parallel to the width of the paper. This is the default orientation.
@LANDscape	This modifier causes text to appear in landscape mode, parallel to the length of the paper. Portrait is the default orientation.
@TABSKIPglue= *TeX value*	This modifier sets the tabskip glue when TEXH format (**\halign**) is selected. (default: **1em plus .5em**)
@HSIZE= *TeX value*	This modifier sets the width of the table when TEXH format (**\halign**) is selected. (default: none)
@POINTsize=*i* **@PTsize**	This modifier sets the nominal character size (to ten times the point size in twentieths of a point) when using RTF or TEXS format (or TEXTEDIT, when mixing horizontal and vertical layouts) so the table is sized properly. For RTF, control information is inserted to set the point size for text as well. For TEXS and TEXTEDIT this is *not* done; it is assumed the proper point size is in effect. (default: 10)
@[NO]RULEs	This modifier causes horizontal and vertical lines to appear around column headings, between entries, and at the end of a table in **TEXH**, **TEXS** or TEXTEDIT formats. Specifying **@NORULEs** disables this feature. (default: **@RULEs**)

Description

The Print command prints the contents of a dataset either in the *normal format* (text only) to the screen or on a file, or in one of several *text processing formats* on a file for use later in a text processing program. Five text processing formats are supported: IBM's DCA (Document Content Architecture), which is imported by IBM wordprocessing products and many non-IBM products; Microsoft's RTF (Rich Text Format), which is imported by Microsoft wordprocessing products and others; two formats for TeX, the typesetting program used in academic circles; and STAT:TEXTEDIT, the text processing program that runs on MTS.

Specifying Fields

There are several ways to specify which fields to print. If no fields are specified, all of the fields are printed. The command

```
   Print Students
```

prints all of the fields in Students, in the order they appear in the dataset. To print a subset of the fields or to change the order, the fields must be specified in the command. For example

```
   Print in Students Name, SIN
```

prints only the Name and SIN fields from Students. Note that the Print command does not sort the records. Students is sorted by SIN; the output of this Print command is not sorted by Name. It is also possible to specify the fields that are not to be printed. The command

```
   Print in Students all but Name, SIN
```

prints the fields from Students other than Name and SIN, in the order they appear in the dataset. A special case Print command

```
   Print in Students Count
```

prints only the number of records in Students, none of the fields (even if there is a field named Count in the dataset).

Print Layouts

There are two basic print layouts. The *horizontal (or tabular) layout* prints the records as horizontal rows, one record per row, with the fields as columns. The *column headings* can be field names, abbreviations, descriptions, or blank. This is the default layout and is used for most printouts.

The *vertical layout* prints the records as vertical columns with one field per row. Each row has a *row heading*, which can be the field name, abbreviation, description, or blank, followed by the field value. The vertical layout is useful when printing fields containing long character strings or when there are so many fields to print that the horizontal layout would force records to wrap around. The `@Vertical` command modifier causes all fields to be printed in the vertical layout.

The *stratified layout*, a combination of the vertical and horizontal layouts, is useful for printing "stratified" data. For example, if the first two fields in a dataset are State and County, and the remaining fields are various observations, there might be many records for each State and County pair. If State and County are the first fields to be printed and the `@strat=2` command modifier is given, then within each group of records having the same values for State and County, those two fields are printed only once in the vertical layout, followed by the remaining fields printed in the horizontal layout, one row for each record in the group. A new group of records begins when a value changes for any of the stratified fields. Since datasets are always sorted in Micro, the stratified fields should be the first fields in the dataset.

Print Settings and Modifiers

Layouts, formats, and other attributes of the output are controlled by *Print settings*, which are specified in the Set command; *command modifiers*, which are attached to the word `Print` in the Print command; and *field modifiers*, which are attached to the fields specified in the Print command.

A Print setting controls an attribute for all subsequent uses of the Print command during a Micro session. The command

```
Set Print: Outlen 132, Cat Abbr
```

sets the output length (maximum length of an output line) to 132 and causes category abbreviations to be printed by default instead of category names.

A command modifier controls an attribute only for the command in which it appears. Some command modifiers can override Print settings. For example, assuming the above Set command was done, the command

```
Print@Count=20@Outlen=80@Cat=Name Students
```

prints the first 20 records for all of the fields in the Students dataset. The output length is 80 and category names are printed, overriding the above settings.

A field modifier controls an attribute only for the field to which it is attached. Some field modifiers can override Print settings and command modifiers. The command

```
Print@Incr=5@Cat=Name in Students SIN@SSN, Name, Sex, Race@Cat=Desc
```

prints every fifth record for the specified fields in Students. Category names are printed, overriding the setting, except for the Race field where category descriptions are printed, overriding the command modifier. The SIN field is printed in standard Social Security number format.

The table Print Settings, in the Settings section, shows which print attributes can be set in advance. The table Print Modifiers, in the Modifiers section, summarizes the modifiers by function and gives the corresponding settings.

Generating Fixed Format Data

The Print command can be used to generate fixed format output into a file for use as input to another program. This requires stripping out the non-data information normally generated by the Print command, and specifying the width, and possibly the position, of each field. For example

```
Print@-Head@-CC@Spaces=0@#Lines=1 on -fixed in Students  -
       SIN@Col=1@Width=9@RJ@Fill='0', Name@Width=24@LJ,  -
       Sex@Col=35@Width=1@Cat=Value, Race@Col=37@Width=1@Cat=Value
```

prints all of the records from Students for the specified fields on the temporary file -FIXED. The column headings and carriage controls are not printed. There are no spaces between fields and each row is one line (field items do not wrap around within their column widths). The SIN field starts in column 1 and is right justified within a 9 character width that is filled with '0's. No column is specified for the Name field so it starts in the next available column, 10, and is left justified within a 24 character width (filled with blanks by default). The Sex and Race fields start in columns 35 and 37, respectively, and the category values are placed in 1 character widths. The "gaps" between fields are filled with blanks.

Text Processing Formats

For any text processing format, specifying '\n' in the `@TITLE` string puts the following text on a new line, e.g., '`@TITLE="Financial Aid\nSenior Undergraduates\nUniversity of Michigan"`' prints a three-line title. For TEXTEDIT format, a continuation title containing the first line of the title, followed by '` (continued)`', is generated when records are in horizontal layout.

The DCA format simply puts tabs between the data items. The tabs are spaced according to the column width and separator size determined from the command, and the entire display is centered or left- or right-aligned according to the `@ALIGNment` modifier. The font used is Courier, 10 characters per inch fixed pitch. The DCA file contains non-printing control information so it must be downloaded from MTS to the microcomputer as a binary file. All text is in EBCDIC, as expected by the various conversion utilities (since DCA is an IBM product) and this encoding is preserved in the binary download. DCA is imported by DisplayWrite, Microsoft Word, and Sprint.[4]

The RTF format uses an RTF table mechanism. The font used is Times Roman. If column and separator sizes are specified in characters, a nominal character value based on the point size is used to determine sizes. If not given, the point size defaults to 10. The nominal character (in twentieths of a point) is ten times the point size. The RTF file is a text file (generated in EBCDIC on MTS) that should be downloaded as a text file (which converts it to ASCII on the microcomputer). RTF is imported by Microsoft Word.

The TEXH and TEXS formats correspond to the `\halign` and `\settabs` mechanisms in TeX, respectively. TEX is a synonym for TEXH. Column widths in `\halign` are determined implicitly by that mechanism, perhaps in conjunction with the value given with the `@HSIZE` modifier, which

[4] DisplayWrite is a registered trademark of IBM Corporation. Microsoft Word is a registered trademark of Microsoft Corporation. Sprint is a registered trademark of Borland International, Inc.

is passed straight through to TEX. Column separators are determined implicitly from the `\tabskip` glue, by default '`1em plus .5em`'. This can also be set, using the `@TABSKIPglue` modifier. Both `@TABSKIPglue` and `@HSIZE` values must be enclosed in quotes if they contain blanks.

Column widths for TEXS, if specified in characters rather than in units, are calculated from a "nominal character size" based on the interword spacing. For Computer Modern 10 point, the sum of the interword space and the interword stretch (font parameters two and three) is 5 points, for a nominal character size of ten times the point size, in twentieths of a point ($5 \times 20 = 10 \times 10$). This ratio is used for nominal character values for all point sizes. If not given, the point size defaults to 10 (14.454 characters per inch).

The nominal character size is also used to wrap lines when a column entry is too long for a single row. (In TEXH, the wrapping is done by `\halign`). Note that `@POINTsize` does *not* insert TEX control sequences affecting point size, it just sizes a TEXS table properly.

Column separators for TEXS are the TEX `\quad` units, divided between columns (one em apiece); left and rightmost columns have one en at the left and right respectively. If rules are present (`@RULE`, the default) a `\strut` and `\vrule` are used, as they are in TEXH.

TEXS uses a `\newdimen`, '`\tabwidth`', when only fields in horizontal layout are printed and a title is also printed. When vertical layout fields are printed (whether or not a title is present) '`\stratwidth`' is also used. TEXH uses `\stratwidth` when vertical layout fields are printed, regardless of title.

TEXTEDIT format uses the STAT:TEXTEDIT table facility for records in horizontal layout, and the frame facility for records in vertical layout, when field names or abbreviations are also printed. The text control record must contain '`macros=on`' (or specify an individual macro library) so the frame macros are recognized. When '`@LANDscape`' is specified, the TEXTEDIT `PLACEMENT` keyword is '`BROADSIDE`'. The `ENTRIES` keyword is always '`TOP`'.

If fields are printed only in horizontal layout, character fields (types C, CC and E) are specified as text columns, with widths. Column separator values are divided between the left and right side of the column. Tables longer than one page are automatically continued to subsequent pages by TEXTEDIT, with automatic replication of column headings and a continuation title; column widths are constant across pages.

When fields are printed in vertical layout, and row headings (field names, etc.) are present, a text frame is used to align the field values. When horizontal fields are printed with vertical fields, all columns are specified as text columns (with sizes), and column separators are set to 1, to ensure uniform width of all tables of horizontal fields.

The column size is based on an estimate of the maximum length of the field value, for numeric fields; for character fields (which may wrap) it is 20. Column widths and separators given as field modifiers on the Print command override these values (`@WIDth`, `@SPaces`). If given in characters, column sizes are based on the nominal character size, ten times the point size in twentieths of a point. The point size defaults to 10 if not given.

Examining the files generated by the TEX and TEXTEDIT formats will answer further questions about the use of Print command output with those programs.

PURGE Micro Command

Purpose: To purge the list of available datasets and then obtain again the datasets contained in userid's directory.

Synonyms: None.

Effect on the Result: All temporary datasets, including the Result, are released.

Usage

```
PUrge
```

Examples

```
Purge
```

Settings

None.

Modifiers

None.

Description

The Purge command empties the list of available datasets, freeing storage and releasing files where necessary, and rereads your directory. This has the same effect on the available datasets as if you were to stop Micro and run it again.

The effect of the Purge command might not be the same as that of the command '`Release all datasets`'. Permanent datasets obtained by way of the Get command are still available after using the Release command but not after a Purge.

READ Micro Command

Purpose: To create a Result set by reading data from a Micro data file; to create a Result set by reading an array of records from a file.

Synonyms: None.

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

```
REAd FROM file [[NOT] UNSCRAMBLED] USING dataset
REAd ARRAY FROM file USING dataset
```

Examples

```
Read from -st82 unscrambled using Students
```

This command reads from the Micro data file -ST82, prompting first for a scramble key, into the Result set using the structure from dataset Students.

```
Read Array from -xyz using A
```

This command reads data in array format from file -XYZ into the Result set using the structure from dataset A.

Settings

None.

Modifiers

None.

Description

The Read command reads data from a file, stored in one of two formats, into the Result set using the structure (dictionary information) from the specified dataset.

If ARRAY is omitted, Micro expects the file to be a Micro data file as prepared by the Micro Write command, the Micro Save command, or any of the data entry facilities.

When reading from a Micro data file, Micro needs to know if the data in the specified file must be unscrambled. If UNSCRAMBLED is given or if the specified dataset is scrambled, Micro asks for a scramble key. A null reply to the scramble key prompt tells Micro that the data is not to be unscrambled. The following equivalent commands force a prompt for a scramble key:

```
read from -sfile after unscrambling using Students
read from -sfile unscrambled using Students
```

If the data is unscrambled during the read operation, the Result set has its scrambled status enabled.

If the specified dataset is scrambled but the Micro data file to be read is not, the prompt can be suppressed by specifying NOT UNSCRAMBLED in the command. This command causes no scramble key to be prompted for and none applied:

```
read from -sfile not unscrambled using Students
```

If ARRAY is given, Micro expects the file to contain unscrambled, binary data stored in "array" format, each line containing an integral number of records, as produced by the Micro Write Array command. An MTS sequential file is preferred for the array format.

RELEASE Micro Command

Purpose: To release any or all of the available datasets.

Synonyms: None.

Effect on the Result: Any dataset, including the Result, can be released.

Usage

`RELease {ALL DATASETS | *}`

`RELease [ALL BUT] <datasets> [, <datasets>] ...`

where

<datasets> can be any of:

dataset
pattern

Examples

`Release *`

This example releases all datasets; '`*`' and '`ALL DATASETS`' are synonyms.

`Release A, B, it`

This example releases datasets A, B, and the Result set.

`Rel ?junk?`

This example releases all datasets containing the string '`junk`' in their names.

`Rel all but Students, Fin-Aid`

This example releases all datasets except Students and Fin-Aid.

Settings

None.

Modifiers

None.

Description

If a released dataset is a permanent set, it is made inactive. Any internal storage (memory) allocated for the dataset are released. The files containing the dataset still exist, so the dataset can be reactivated. This is also true for a dataset made available using the Get command. (This is one way Release differs from Purge.)

If a released set is temporary, it is destroyed and cannot be reactivated. Saving a temporary set makes it permanent and inactive.

REMOVE Micro Command

Purpose: To create a Result dataset containing the records from one dataset which are not found in another dataset.

To create a Result dataset containing the records from a dataset which do not satisfy the specified search criteria.

Synonyms: None.

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

`REMove` *dataset1* `FROM` *dataset2*

`REMove FROM` *dataset* `WHERE` <condition> [<log_op> <condition>] ...

where

<log_op> can be any of:

`&`	`AND`
`|`	`OR`
`;`	`ALSO WHERE`

<condition> can be any of:

field [`IS`] <rel_op> <right_side>
field [`IS NOT`] <rel_op> <right_side>
field `=` <value> [`OR` <value>] ...
field `IS BETWEEN` <value> `AND` <value>
field `IS FROM` <value> `TO` <value>

<rel_op> can be any of:

`<`	`LESS THAN`
`<=`	`LESS THAN OR EQUAL TO`
`=`	`EQUAL TO`
`<>`	`IS NOT EQUAL TO`
`MAT`	`MATCHES`
`>=`	`GREATER THAN OR EQUAL TO`
`>`	`GREATER THAN`

<right_side> can be any of:

field
<value>

<value> can be any of:

category
number
string

Examples

```
Remove it from A
```

The Result set is dataset A with the previous Result removed.

```
Remove from Students where SIN = 987654321
```

The Result set is the Students dataset without the record where field SIN has value 987654321.

```
Rem from Fin-Aid where CL is Sr and Amount <= 400
```

The Result set is the Fin-Aid dataset without the records where field CL has the values '`Sr`', and where field Amount is less than or equal to 400.

```
Rem from it w name matches S
```

The Result set contains the current Result excluding all records where field Name begins with '`S`'.

Settings

None.

Modifiers

None.

Description

The Remove command creates a Result set containing records from one dataset that are not identical to those in another dataset. The two sets can be given explicitly, *dataset1* to be removed from *dataset2*. In that case, the two sets must have the same structure, the same type and number of fields. The Result set is *dataset2* less the intersection of *dataset1* and *dataset2*. The set to be removed can also be implied by specifying conditions for records to be removed from a single *dataset*. The

Result set consists of the *dataset* less the records specified. The specification syntax is the same as for the Find command. You can in effect "find" a subset of the dataset and produce the complement as the Result.

REPLACE Micro Command

Purpose: To replace one dataset with the contents of another dataset.

Synonyms: None.

Effect on the Result: If the Result dataset is replaced, it still exists but its contents are replaced. If the Result dataset is used to replace another dataset, there is no longer a dataset named Result but its contents are moved into the dataset being replaced.

Usage

`REPlace` *dataset1* `WITH` *dataset2* [{`OK` | `!`}]

Examples

```
Replace Students with it
```

Dataset Students is replaced with the Result set, which then does not exist. Since Students is permanent (but replaceable) Micro asks for confirmation.

```
Repl A w/ B ok
```

Dataset A is replaced with dataset B; '`ok`' suppresses the confirmation prompt.

Settings

None.

Modifiers

None.

Description

The Replace command copies the contents of *dataset2* into *dataset1*. If *dataset2* is permanent, it remains as a permanent, inactive dataset; if temporary, it is released.

If *dataset1* is permanent, it can be replaced only if its replace status (set in MicDef or with the Save command) is Yes. Otherwise the command is cancelled.

If `OK` is not given in the command, Micro asks for confirmation before doing the permanent replace operation. Micro issues a warning if *dataset2* contains fewer records than *dataset1*. It also warns if the structures of the datasets are different (number and order of fields, field types, field lengths, etc.). If *dataset1* is permanent and either of these conditions occurs, Micro asks again for confirmation, unless `OK` is given on the command.

If *dataset1* is permanent, its records are backed up by moving the Micro data file to a temporary file. Then the records from *dataset2* are written to *dataset1*'s Micro data file. Should any problems occur, the backup is restored. If the structures are different, Micro also replaces *dataset1*'s dictionary file with the structure from *dataset2* using the same method. Since Micro backs up to temporary files, if the system goes down during the replace operation the files will have to be restored using *Restore or *FS.

If *dataset1* is permanent and scrambled, Micro asks for the scramble key to be used to scramble the data. If this key is different from the current scramble key (the one used to open the dataset) Micro asks if the key is being changed. If not, Micro asks for the scramble key again. If the key is being changed, Micro asks for the current and new scramble keys again to confirm the change. The key is stored in memory while a dataset is active, to prevent unwanted scramble key changes.

RESETTINGS Micro Command

Purpose: To reset any of the global Micro settings to its initial value.

Synonyms: None.

Effect on the Result: None.

Usage

`RESEttings` <group>: <items> [; <group>: <items>] ...

where

<group> can be any of:

```
COMMAND        CMD
USER
CALC
JOIN
PRINT
XTAB
```

<items> can be any of:

item [, *item*] ...
`*` (resets all items in the group)

Notes: Currently there can be no space between <group> and the colon, and there must be at least one space following the colon.

Examples

```
Resetting  Cmd: Ready, Cost, Time, Outlen;  -
         Print: Field, Category, Underline;  -
          Xtab: Percent, Print
```

This example resets the items from the given groups to their default values: from the `Cmd` group, `Ready`, `Cost`, `Time` and `Outlen`; from the `Print` group, `Field`, `Category` and `Underline`; from the `Xtab` group, `Percent` and `Print`.

```
Reset  Command: Input, Output, Verify
```

This example resets the `Input`, `Output` and `Verify` items from the `Command` group to their default values.

```
Reset  Print: *
```

This example resets all items in the `Print` group to their default values.

Settings

None.

Modifiers

None.

Description

The global settings fall into several groups. Tables for the different groups appear here. Explanations for the User and Command groups are in the Settings command; explanations for the other groups are in the Settings section for the corresponding Micro command.

The following settings affect the user's Micro environment. The values for these settings are stored in the directory file on the user's MTS userid. Changes to these settings remain in effect for subsequent Micro sessions.

User Settings

Group	Item	Allowable values	Initially
USER:	MASTER ID MSID	userID or OFF	OFF
USER:	RUNFILE	*file* or OFF	OFF

The following settings affect Micro commands in general. Synonyms appear on the second line of each entry.

Command Settings

Group	Item	Allowable values	Initially
CMD:	CLOCK	ON or OFF	OFF
CMD:	CONTCHAR	*c*	'_'
CMD:	COST $	ON or OFF	OFF
CMD:	ECHO	ON or OFF	ON
CMD:	INPUT I	*fdname*	*SOURCE*
CMD:	OUTPUT O	*fdname*	*SINK*
CMD:	OUTLEN ORL (changes PRINT: OUTLEN)	positive integer	79
CMD:	READY	ON or OFF	ON
CMD:	TIME	positive integer or OFF	10
CMD:	TRACE (once on, it stays on)	ON	OFF
CMD:	VERIFY V	ON or OFF	ON

The following settings affect the Calculate command.

Calculate Settings

Group	Item	Allowable values	Initially
Calc:	Include	Goodrecords or Allrecords	Goodrecords

The following settings affect the Crosstabulate (Xtab) command. In addition, the Print settings are used if the Result dataset is printed, as it is by default.

Crosstabulate Settings

Group	Item	Allowable values	Initially
XTAB:	AVE FACTOR	power of 10 or OFF	100
XTAB:	STD FACTOR	power of 10 or OFF	100
XTAB:	% FACTOR	power of 10 or OFF	100
XTAB:	PERCENT %	ON or OFF	ON
XTAB:	PRINT	ON or OFF	ON

The following setting affects the Join and RestrictAndJoin commands.

Join Settings

Group	Item	Allowable values	Initially
JOIN:	GROWTH FACTOR GF	$1 <= i <= 9$	3

The settings in the following table affect the Print command. Not all Print settings apply to text processing formats, as noted in the last column.

Print Settings

Group	Item	Allowable values	Initially	Text Format
PRINT:	CATEGORY CAT	NAme, ABbr, DEsc, or Value	NAme	×
PRINT:	DIRECTION	Forward or Reverse	Forward	×
PRINT:	FIELD FLD	NAme, ABbr, DEsc, or OFF	NAme	×
PRINT:	FORWARD	ON or OFF	ON	×
PRINT:	HEADER CC HCC	carriage control	'0'	
PRINT:	HORIZONTAL HORIZ	ON or OFF	ON	×
PRINT:	HIRCC	carriage control	'␣'	
PRINT:	LINES/PAGE LPP	positive integer	60	×
PRINT:	OUTLEN	#chars	79 (chars)	
PRINT:	PAGE CONTROL PC	ON or OFF	OFF	
PRINT:	PAGE CC PCC	carriage control	'1'	
PRINT:	PDATE	*string* or OFF	OFF	
PRINT:	REVERSE	ON or OFF	OFF	×
PRINT:	TITLE	*string* or OFF	OFF	×
PRINT:	TITLE CC TCC	carriage control	'0'	
PRINT:	UNDERLINE	character or OFF	'_'	
PRINT:	VERTICAL VERT	ON or OFF	OFF	×
PRINT:	VIFCC	carriage control	'␣'	
PRINT:	VIRCC	carriage control	'0'	

RESTRICT Micro Command

Purpose: To create a Result dataset containing the records from one dataset which have fields satisfying a logical relationship to fields in another dataset.

Synonyms: None.

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

```
Restrict IN dataset1 WHERE <fields> [, <fields>] ...
                     <verb> <fields> [, <fields>] ... [IN dataset2]

Restrict IN dataset1 WHERE <fields> [, <fields>] ...
                     <verb> [<noise words>] IN dataset2
```

where

<fields> can be any of:

field
pattern
field THRU *field*

<verb> can be any of

```
=       IS        ARE
<>      IS NOT    ARE NOT
```

<noise words> can be any combination of

```
THE SAME
THE SAME AS
THAT
THOSE
```

Examples

Restrict in Students where SIN is SIN in Fin-Aid

The Result set contains the records from Students having a SIN value that matches the SIN value in at least one record in Fin-Aid. In other words, the Result contains the students who have financial aid.

```
Restrict in Faculty where SSN, Dept are not Soc#, Unit in AdminData
```

The Result set contains the records from Faculty whose values for SSN and Dept do not match the values for Soc# and Unit, respectively, in any record in AdminData. In other words, the Result contains the Faculty records that are not listed in AdminData.

```
Rest in A w F1, Sal? = F1, Sal? in B
```

```
Rest in A w F1, Sal? = those in B
```

These two commands are equivalent. The Result set contains the records from A whose values for Field 1 and all fields beginning with '`Sal`' match the values for Field 1 and all fields beginning with '`Sal`', respectively, in at least one record in B. In the second command, the fields specified for A are assumed for B.

```
Rest in A w Year thru Date <> those in B
```

The Result set contains the records from A whose values for fields Year through Date do not match the values for fields Year through Date, respectively, in any record in B. The fields specified for A are assumed for B.

```
Rest in MedRecords w PatientSSN = DoctorSSN
```

```
Rest in MedRecords w PatientSSN = DoctorSSN in MedRecords
```

These two commands are equivalent. The Result set contains records from MedRecords whose values for PatientSSN match the values for DoctorSSN in any record in MedRecords. In other words, the Result contains the patients who are also doctors in this dataset.

Settings

None.

Modifiers

None.

Description

Before reading this section, it may be useful to look at the Restrict command discussion and examples in Tutorial 2.

The Restrict command creates a Result set containing a subset of the records from a dataset based on the relationship between the *link fields* specified in the command. For example, SIN (the student identification number) is the link field in the following command.

```
Restrict in Students where SIN is SIN in Fin-Aid
```

The Result contains a subset of the records in Students, the records where the link field values match. If a Students record has a SIN value matching a SIN value in any Fin-Aid record, it is included in the Result. If there is more than one link field, the values for all of them must match.

Note that it is not necessary for link fields to have the same name in both datasets. However, as in Students and Fin-Aid, it is common (and recommended). If the names are the same, the link fields need only be specified once. For example, the command

```
Restrict in Students where SIN is in Fin-Aid
```

is equivalent to the previous Restrict command. In both cases, the Result contains the students who have financial aid. Although the field names need not be the same, the link fields must be compatible in type and length.

If the `NOT` keyword is specified in the command, a *not-restriction* is performed. For example, the command

```
Restrict in Students where SIN is not in Fin-Aid
```

generates a Result containing a subset of the records in Students where the link field values do not match. If a Students record has a SIN value matching a SIN value in any Fin-Aid record, it is not included in the Result. The following sequence of commands generates the same Result.

```
Restrict in Students w SIN is in Fin-Aid
Name it WithAid
Remove WithAid from Students
```

In both cases, the Result contains the students who do not have financial aid.

The link fields need not be from different datasets. For example, in the command

```
Restrict in MedRecords w PatientSSN = DoctorSSN in MedRecords
```

the PatientSSN and DoctorSSN fields are both in the MedRecords dataset. If a MedRecords record has a PatientSSN value matching a DoctorSSN value in any MedRecords record, it is included in the Result. If the dataset names are the same, the second can be omitted. For example, the command

```
Restrict in MedRecords w PatientSSN = DoctorSSN
```

is equivalent to the previous Restrict command. In both cases, the Result contains the patients who are also doctors in MedRecords. Note that the Result of this Restrict command is different from that of a Find command with this same syntax which finds only the records where PatientSSN and DoctorSSN match in the same record (i.e., where the patient is his own doctor).

RESTRICTANDJOIN Micro Command

To create a result dataset containing all of the fields from two datasets, usually linked together by common fields.

Synonyms: `RAj`, `RAm` (Most people use RAJ.)

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

```
RestrictAndJoin [{ALL|MATCHING}] dataset1 [BY <fields> [, <fields>] ...] WITH
                [{ALL|MATCHING}] dataset2 [BY <fields> [, <fields>] ...]
```

where

<fields> can be any of:

field
pattern
field `THRU` *field*

Examples

`RAJ Students by SIN with Fin-Aid`

The Result set contains the link field, SIN, followed by the remaining fields from Students, followed by the remaining fields from Fin-Aid. Records in Students and Fin-Aid with matching SIN values are joined together in the Result. Records in Students and Fin-Aid with no matching values for SIN are not included in the Result.

`RAJ all Students by SIN with Fin-Aid`

The Result set contains the link field, SIN, followed by the remaining fields from Students, followed by the remaining fields from Fin-Aid. Records in Students and Fin-Aid with matching SIN values are joined together in the Result. Records in Students with no matching SIN in Fin-Aid are joined with filler values for the Fin-Aid fields. Records in Fin-Aid with no matching SIN in Students are not included in the Result.

`RAJ A by Ayr, Amon with B by Byr, Bmon`

The Result set contains the link fields from A, Ayr and Amon, followed by the remaining fields from A, followed by the remaining fields from B. Records in A with Ayr and Amon values that match the Byr and Bmon values in B are joined together in the Result. Records in A and B with no matching link fields are not included in the Result.

`RAJ A with B`

The Result set contains all of the fields from A followed by all of the fields from B. Every record from A is paired with every record from B. The Result is the Cartesian product of A and B.

Settings

The following setting affects the Join and RestrictAndJoin commands.

Join Settings

Group	Item	Allowable values	Initially
`JOIN:`	`GROWTH FACTOR` `GF`	$1 <= i <= 9$	3

`GROWTH FACTOR` *i*
`GF`

The `GROWTH FACTOR` setting controls the size of the Result set created by the Join command, which can be as large as the product of the number of records in each of the two sets. It may be necessary to increase this value if a "partial completion" message appears on the screen. (default: 3)

Modifiers

`@GF=`*i* This modifier is used to override the `GF` setting.

Description

Before reading this section, it may be useful to look at the RestrictAndJoin command discussion and examples in Tutorial 2.

The RestrictAndJoin (RAJ) command generates a Result dataset containing the fields from one dataset appended to the fields of another dataset. Records from the first dataset are joined with records from the second.

Link Fields

How the records are joined together is determined by several factors related to the *link fields* specified in the command. For example, SIN (the student identification number) is the link field in the following command.

```
RAJ matching Students by SIN with matching Fin-Aid by SIN
```

The Result contains the link field, SIN, followed by the remaining fields from Students, followed by the remaining fields from Fin-Aid. Note that it is not necessary for link fields to have the same name in both datasets. However, as in Students and Fin-Aid, it is common (and recommended). If the names are the same, the link fields need only be specified once. For example, the command

```
RAJ matching Students by SIN with matching Fin-Aid
```

is equivalent to the previous RAJ command. Although the field names need not be the same, the link fields must be compatible in type and length.

Inner and Outer Joins

The '`matching`' keyword applied to both datasets tells Micro to perform an *inner join*. The Result contains only the records where the link field values match. If there is more than one link field, the values for all of them must match. If a Students record has a SIN value matching a SIN value in a Fin-Aid record, those records are joined together and included in the Result. Records with links that don't match do not contribute to the Result. If there are no matches at all, the Result is null. Therefore, the Result of the above commands includes only the records for students who have financial aid joined with their financial aid records.

To generate a Result containing the records for all students, not just those with financial aid, change the '`matching`' keywords to '`all`' as in the command

```
RAJ all Students by SIN with all Fin-Aid
```

The '`all`' keyword applied to both datasets tells Micro to perform an *outer join*. The Result contains all of the records from both datasets. Where the link field values match, the records are joined as they are for the inner join. Where the link field values don't match, filler data is inserted for the "missing" non-link fields to complete the Result record. The filler data is the value for the first default category, if one exists, otherwise it is zero for numeric fields, blanks for character fields. If a Students record has a SIN value matching a SIN value in a Fin-Aid record, those records are joined together and included in the Result. If a Students record has a SIN value that doesn't match any SIN value in Fin-Aid, the Students record is joined with filler data for the Fin-Aid fields. If a Fin-Aid record has a SIN value that doesn't match any SIN value in Students, the Fin-Aid record is joined with filler data for the Students fields. Therefore, the Result of the above command includes the records for all students and for all financial aid awards, joined together where the SIN values match and joined with filler data where they don't.

To generate a Result containing the records for all students but only the matching financial aid records, use the command

```
RAJ all Students by SIN with matching Fin-Aid
```

The '`all`' keyword forces an outer join for Students; the '`matching`' keyword forces an inner join for Fin-Aid. Therefore, the Result of the above command includes the records for all students, joined with their financial aid records where the SIN values match and joined with filler financial aid data where they don't.

Join and RestrictAndJoin

If `ALL` or `MATCHING` are specified for both datasets then the Join and RAJ commands are equivalant. If they are omitted in the Join command, `ALL` is assumed. The following commands

```
Join Students by SIN with Fin-Aid
Join all Students by SIN with all Fin-Aid
RAJ all Students by SIN with all Fin-Aid
```

are equivalent; each performs an outer join. If these keywords are omitted in the RAJ command, `MATCHING` is assumed. The following commands

```
RAJ Students by SIN with Fin-Aid
RAJ matching Students by SIN with matching Fin-Aid
Join matching Students by SIN with matching Fin-Aid
```

are equivalent; each performs an inner join.

Data Relationships

Students is an example of a dataset where the link field is also a *key field*, a field that uniquely identifies each record. There is one record in Students for each student, each with its own identifying number, SIN. Suppose there is another dataset, StudentAddr, containing the SIN and the addresses of the students in Students, one record per student. Since SIN is also a key field in StudentAddr, there exists a *one-to-one relationship* between Students and StudentAddr. A join of Students by SIN with StudentAddr yields a Result containing one record per student. For each student, one record from Students is joined with the record from StudentAddr having the same SIN.

There can be any number of financial aid records in Fin-Aid for one student, all with the same value for SIN. Since SIN is not a key field in Fin-Aid, there exists a *one-to-many relationship* between Students and Fin-Aid. A join of Students by SIN with Fin-Aid yields a Result containing one record per financial aid award. For each student, one record from Students is joined with all of the records from Fin-Aid having the same SIN.

Suppose there is another dataset, StudentDupl, containing two records for each student, both with the same SIN. Since SIN is not a key field in StudentDupl, there exists a *many-to-many relationship* between StudentDupl and Fin-Aid. A join of StudentDupl by SIN with Fin-Aid yields a Result containing two records per financial aid award. For each student, two records from StudentDupl are joined with all of the records from Fin-Aid having the same SIN. The many-to-many relationship, where the link fields are not key fields in either dataset, usually indicates a problem with one or both of the datasets.

The Cartesian Product

If no link fields are specified, each record from the first dataset is joined with each record from the second, yielding the *Cartesian product* of the two. The cardinality (number of records) of the Result is the product of the cardinalities of the two sets, which is the upper bound on the size of a RAJ Result.

The Cartesian product is a useful way to append new fields that have no data yet. Suppose there is a dataset, NewFields, containing some new fields to be added to Students, and that it has one record holding the initial values for these fields. Then the command

```
RAJ Students with NewFields
```

generates a Result containing all of Students with the initial values for the new fields appended to each record. The cardinality is the same as for Students since NewFields contains only one record.

If the cardinality of a Cartesian product will exceed a threshold value (10,000) and the cardinality of both datasets is more than one, Micro issues a warning message before continuing the operation. This is often due to accidentally omitting the link fields in the command.

Miscellaneous

The size of the Result is determined by the size of the specified datasets, the use of the `ALL` and `MATCHING` keywords (an outer join is at least as large as an inner join of the same datasets), and the nature of the data relationships between the datasets. It is possible to create a large Result dataset with the RAJ command. Sometimes a "partial completion" message appears on the screen and the operation stops. If this occurs, increase the *growth factor* using the `GROWTH FACTOR` setting or the `@GF` command modifier. The values for growth factor can be from 1 to 9.

Since the RAJ command takes data from two datasets, it is possible to have more than one field with the same name in the Result. Micro does not check for this. Fields with the second and later occurrences of this name cannot be referenced in subsequent commands until the first occurrence is renamed.

The RAJ command does not preserve crosstabulation. If either of the specified datasets is crosstabulated, the Result contains the Count field but its values do not reflect crosstabulation in the Result.

SAVE Micro Command

Purpose: To make a temporary dataset permanent by saving its contents in permanent file storage.

Synonyms: None.

Effect on the Result: Saving a dataset other than the Result has no effect on the Result. Since a permanent dataset may not be named "Result" or "It", you must specify a new dataset name when saving the Result set. In this case, there is no longer a Result set.

Usage

```
SAve dataset [AS new_dataset_name] [[NOT] SCRAMBLED]
                                   [[NOT] DESTROYABLE]
                                   [[NOT] REPLACEABLE]
                                   [DESCRIPTION = text]
                                   [ON file]
                                   [WITHOUT [MODIFICATION]]
                                   [SHARING dataset DICTIONARY]
```

Examples

```
Save A
```

This makes the temporary dataset A permanent. Micro asks for the destroy status, the replace status, and the description. If any of A's parent datasets are scrambled, Micro also asks for a scramble key. If given, Micro uses the key to scramble the data, else the data is not scrambled.

```
Save it scrambled as FY86  replaceable  not destroyable  -
    description = "Fiscal Year 1986"
```

This saves the Result dataset, giving it the name FY86. Micro asks only for the scramble key; the destroy status, replace status, and description are given in the command.

```
Save it as FY87 sharing FY86 dictionary
```

This saves the Result dataset, giving it the name FY87. Assuming that the dataset FY86 exists, Micro checks the field structure of the Result dataset against that for FY86 and, if they are compatible, FY87 will share FY86's dictionary file. Micro asks for the destroy status, the replace status, and the description. If any of the Result's parent datasets are scrambled, Micro asks for a scramble key.

Settings

None.

Modifiers

None.

Description

The Save command saves temporary datasets to permanent files. The destroyable and replaceable attributes and the dataset description can be given on the command. If omitted, they are prompted for. The name of the dataset to be saved cannot be the name of an existing permanent dataset (or "It" or "Result"); a new name can be given if the current name is not suitable. The date on which the set is saved, and the userid from which it is saved, are stored in the directory file with the description.

If a file name is not given, the first 11 characters of the dataset name, followed by '#', are used for the dictionary file name, and the same root, followed by '$', is used for the data file name. The data file can be specified as a temporary file, but the dictionary file is always permanent. Files on other userids can be used if read/write access is available. Micro confirms that the set has been saved, or "saved", if to a temporary file:

```
...
Dictionary file = CCID:HUGE#   Micro data file = -HUGE$
RESULT has been "saved" as Hugeset.
```

SELECT Micro Command

Purpose: To create a Result dataset containing selected fields from another dataset.

Synonyms: None.

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

`Select IN` *dataset* [`ALL BUT`] <fields> [, <fields>] ...

where

<fields> can be any of:

field
pattern
field `THRU` *field*

Examples

```
Select in Students SIN, Name, Sex, Race
```

The Result set contains fields SIN, Name, Sex and Race from dataset Students. Since SIN is a key in Students, weeding does not occur.

```
Sel in Fin-Aid all but Rem*
```

The Result set contains all fields from Fin-Aid except Rem*. Any duplicate records in the Result are weeded.

```
Select in Students S?
```

The Result set contains all fields from Students beginning with 's'. Since SIN is included and SIN is a key in Students, weeding does not occur.

```
Sel in Fin-Aid f1 thru f5, f10
```

The Result set contains fields 1 through 5, and field 10.

Settings

None.

Modifiers

None.

Description

The Select command creates a Result set containing fields selected from another dataset. If the fields selected do not contain enough information to maintain unique records, the duplicate records are weeded out of the Result and a message is printed. If duplicate records are to be retained, a unique field can be added to the original set with the Key command, and the Select must include this key in the Result.

If fields are selected explicitly, by name or number, they appear in the Result set in the order named. If they are selected with '`ALL BUT`', or with a pattern such as '`S?`', they appear in the Result set in the order they appear in the set from which they are selected.

SETTINGS Micro Command

Purpose: To change any of the global Micro settings.

Synonyms: TUrn

Effect on the Result: None.

Usage

```
SETtings <group>: <setting> [, <setting>] ...
      [; <group>: <setting> [, <setting>] ...] ...
```

where

<group> can be any of:

```
COMMAND        CMD
USER
CALC
JOIN
PRINT
XTAB
```

<setting> can be any of:

item [=] *value*
item (for `ON`/`OFF` items, assumes `ON`)

Currently there can be no space between <group> and the colon, and there must be at least one space following the colon.

Examples

```
Settings  Cmd: Ready off, Cost on, Time off, Outlen=132;  -
        Print: Field abbr, Category abbr, Underline '=';  -
         Xtab: Percent off
```

This example affects settings in the `Cmd`, `Print`, and `Xtab` settings groups.

```
Set  Command: Input = Budget.bat, Output = -out, Verify off
```

This affects settings in the `Cmd` group.

Settings

None.

Modifiers

None.

Description

The global settings fall into several groups. The User and Command group settings are summarized in tables and explained here. Settings for the other groups have only tables here; explanations for these groups are in the Settings sections for the corresponding commands.

The following settings affect the user's Micro environment. The values for these settings are stored in the directory file on the user's MTS userid. Changes to these settings remain in effect for subsequent Micro sessions.

User Settings

Group	Item	Allowable values	Initially
`USER:`	`MASTER ID` `MSID`	userID or `OFF`	`OFF`
`USER:`	`RUNFILE`	*file* or `OFF`	`OFF`

`MASTER ID` {*userid*|`OFF`} `MSID`	The master userid is the MTS userid on which the master command trace file and the default MPF library are stored. This defaults to the user's userid, but may be set to another userid if desired. An MTS project that administers a number of accounts for users of Micro may wish to keep the command trace for those accounts, for instance. Setting this to `OFF` restores the master userid to the user's userid. (default: user's userid)
`RUNFILE` {*file*|`OFF`}	The `RUNFILE` setting is the name of an MTS file containing Micro commands to be executed when Micro is $Run in MTS, beginning with the session following the current one.

The following settings affect Micro commands in general. Synonyms appear on the second line of each entry.

Command Settings

Group	Item	Allowable values	Initially
CMD:	CLOCK	ON or OFF	OFF
CMD:	CONTCHAR	*c*	'–'
CMD:	COST $	ON or OFF	OFF
CMD:	ECHO	ON or OFF	ON
CMD:	INPUT I	*fdname*	*SOURCE*
CMD:	OUTPUT O	*fdname*	*SINK*
CMD:	OUTLEN ORL (changes PRINT: OUTLEN)	positive integer	79
CMD:	READY	ON or OFF	ON
CMD:	TIME	positive integer or OFF	10
CMD:	TRACE (once on, it stays on)	ON	OFF
CMD:	VERIFY V	ON or OFF	ON

CLOCK {ON|OFF}

If ON, the CLOCK setting prints the CPU time and elapsed time for the execution of each command. (default: OFF)

CONTCHAR *c*

The CONTCHAR is the continuation character for Micro command lines. A command line ended with this character can be continued to the next line. (default: '–')

COST {ON|OFF}
$

The COST setting enables or disables printing of cost amounts after each command. If ON, four amounts are printed: the cost of the command, the Micro cost, which is the MTS cost while running Micro, the surcharge amount in parentheses, and the total cost of the MTS session thus far. (default: OFF)

ECHO {ON|OFF}

The ECHO setting controls printing of command lines read from a command file. When ECHO is ON, command lines are printed at the terminal. (default: ON)

INPUT *fdname*
I

INPUT sets the source of command input to an MTS *fdname*, file/device name. Input can be read from the terminal, a file, or other MTS pseudodevice. (default: *SOURCE*)

`OUTPUT` *fdname* `I`	`OUTPUT` sets the source of command input to an MTS *fdname*, file/device name. Output can be printed at the terminal, on a file, or other MTS pseudodevice. (default: *SOURCE*)
`OUTLEN` *i* `ORL`	The `OUTLEN` setting is the length of output lines. Changing this via `CMD: OUTLEN` also changes `PRINT: OUTLEN`. (default: 79)
`READY {ON|OFF}`	This setting controls the printing of the Micro command prompt. If `OFF`, the '`Ready:`' heading is not printed on the line above the '–' which marks the command prompt.
`TIME` *i*	The `TIME` setting is a limit, in CPU seconds, on the execution of a single Micro command. (default: 10)
`TRACE ON`	The `TRACE` facility traces command execution. The facility uses two command trace files, the *local command trace file* and the *master command trace file*. The local file keeps all commands for a session, the master file for all sessions. When this facility is activated, the local command trace file, if not empty, is copied to the end of the master command trace file, and emptied. Each time a Micro command is executed, the time, date, name of the command and command text are written to the local command trace file. If the Micro session ends normally, the local file is copied to the end of the master file, and then emptied. Once the command trace facility is activated during a session, it cannot be deactivated. The local file is located on the userid in use during the Micro session, the master file on a master userid, which may be different. The files are named MASTERCMT and LOCALCMT by default, which may be changed using ILIR:MicDef. (default: `OFF`)

`VERIFY {ON|OFF}`
`V`
	The `VERIFY` enables or disables confirmation of the results of Micro commands, such as the number of records or other information about a Result set, etc. (default: `ON`)

The following settings affect the Calculate command.

Calculate Settings

Group	Item	Allowable values	Initially
`Calc:`	`Include`	`Goodrecords` or `Allrecords`	`Goodrecords`

The following settings affect the Crosstabulate (Xtab) command. In addition, the Print settings are used if the Result dataset is printed, as it is by default.

Crosstabulate Settings

Group	Item	Allowable values	Initially
`XTAB:`	`AVE FACTOR`	power of 10 or `OFF`	100
`XTAB:`	`STD FACTOR`	power of 10 or `OFF`	100
`XTAB:`	`% FACTOR`	power of 10 or `OFF`	100
`XTAB:`	`PERCENT` `%`	`ON` or `OFF`	`ON`
`XTAB:`	`PRINT`	`ON` or `OFF`	`ON`

The following setting affects the Join and RestrictAndJoin commands.

Join Settings

Group	Item	Allowable values	Initially
`JOIN:`	`GROWTH FACTOR` `GF`	$1 <= i <= 9$	3

The settings in the following table affect the Print command. Not all Print settings apply to text processing formats, as noted in the last column.

Print Settings

Group	Item	Allowable values	Initially	Text Format
PRINT:	CATEGORY CAT	NAme, ABbr, DEsc, or Value	NAme	×
PRINT:	DIRECTION	Forward or Reverse	Forward	×
PRINT:	FIELD FLD	NAme, ABbr, DEsc, or OFF	NAme	×
PRINT:	FORWARD	ON or OFF	ON	×
PRINT:	HEADER CC HCC	carriage control	'0'	
PRINT:	HORIZONTAL HORIZ	ON or OFF	ON	×
PRINT:	HIRCC	carriage control	'␣'	
PRINT:	LINES/PAGE LPP	positive integer	60	×
PRINT:	OUTLEN	#chars	79 (chars)	
PRINT:	PAGE CONTROL PC	ON or OFF	OFF	
PRINT:	PAGE CC PCC	carriage control	'1'	
PRINT:	PDATE	*string* or OFF	OFF	
PRINT:	REVERSE	ON or OFF	OFF	×
PRINT:	TITLE	*string* or OFF	OFF	×
PRINT:	TITLE CC TCC	carriage control	'0'	
PRINT:	UNDERLINE	character or OFF	'_'	
PRINT:	VERTICAL VERT	ON or OFF	OFF	×
PRINT:	VIFCC	carriage control	'␣'	
PRINT:	VIRCC	carriage control	'0'	

SORT Micro Command

Purpose: To sort a dataset on one or more fields.

Synonyms: Order

Effect on the Result: A new Result dataset is created, replacing the existing Result.

Usage

`SOrt IN` *dataset* [<fields> [, <fields>] ...]
[`ALL BUT` <fields> [, <fields>] ...]

`SOrt` *dataset* `BY` [<fields> [, <fields>] ...]
[`ALL BUT` <fields> [, <fields>] ...

where

<fields> can be any of:

field
pattern
field `THRU` *field*

Examples

```
Sort Students by Name, SIN
```

In the Result set the fields Name and SIN are first and second, and the remaining fields of Students appear in the original order. The Result is in alphabetical order, the records sorted by student name, and where ties occur, by id number.

```
Sort Fin-Aid by SIN and all but Year thru Term
```

The Result set is sorted by the Student Id Number. In the Result, the field SIN is first, and all fields except those from Year to Term, inclusive, follow. The fields from Year to Term appear at the end of the Result set.

```
Sort it by zip, state, city, addr
```

The Result set has fields zip, state, city and addr, with the remaining fields in the order they appear in the Result set. A mailing list is thus sorted by zipcode.

```
Sort it by all but f3, f2
```

The Result set has fields F1 through FN, where N is the last field, in that order, except that fields F3 and F2, in that order, are at the end.

```
sort it by f5 thru f1, f3
```

The Result set has fields F5 through F1, *including* F3, followed by another F3, followed by the remaining fields. The word '`thru`' includes all fields from the beginning through the end of the range, in the order implied. Naming F3 again adds another field of the same type and size (and name, description, etc.) to the dataset.

Settings

None.

Modifiers

None.

Description

Records in Micro datasets are kept in sorted order at all times. The sort is done using the entire record as the key. In effect, records are sorted on the first field in the record, in the case of ties on the second field, and so on to the last field if necessary, since duplicates are weeded and records are therefore unique. If an alphabetical field is first in the record, records are in alphabetical order. If a numeric field is first, records are in numeric order.

The Sort command allows a sort to be performed on certain fields. Fields that are named explicitly on the command are placed first in the Result set. Fields that appear after the '`ALL BUT`' phrase are placed at the end of the Result set, in the order given on the command. If a field is specified more than once in the command, it will appear in the Result that many times. This is a way of creating an extra field in a dataset. Fields cannot be deleted from the record by the Sort command.

STOP Micro Command

Purpose: To end the Micro run permanently and return to MTS command mode.

Synonyms: END

Effect on the Result: All datasets are released, including the Result.

Usage

```
STop
```

Examples

```
Stop
```

This ends the Micro session.

Settings

None.

Modifiers

None.

Description

The Stop command terminates the Micro session and returns you to MTS command mode. You can't return to Micro using the MTS $Restart command.

All datasets are released. If you have any temporary datasets that you want to keep for use in future Micro sessions, you must use the Save command to make them permanent before stopping.

Any material in the command after Stop (e.g., '`stop Micro`') will cause a syntax error and the command will not be processed.

TECHDOC Micro Command

Purpose: To print the technical documentation for a dataset, fields in a dataset, or categories of a field.

Synonyms: TDoc

Effect on the Result: None.

Usage

`TEchdoc ALL DATASETS` [<options>] ...

`TEchdoc [ALL BUT]` <datasets> [, <datasets>] ... [<options>] ...

`TEchdoc IN` *dataset* [<options>] ...

`TEchdoc IN` *dataset* `[ALL BUT]` <fields> [, <fields>] ... [<options>] ...

`TEchdoc IN` *dataset* `CATEGORYS OF` *field* `[ON` *file*`]`

`TEchdoc IN` *dataset* *field* `[ALL BUT]` <categories> [, <categories>] ... `[ON` *file*`]`

where

<datasets> can be any of:

dataset
pattern

<fields> can be any of:

field
pattern
field `THRU` *field*

<categories> can be any of:

category
pattern
category `THRU` *category*

<options> can be any of:

`NO CATEGORIES` `NOCATS` `FIELDS ONLY` `*`
`ON` *file*

Examples

```
TechDoc Students on -doc
```

This example writes technical documentation for dataset Students on the file –DOC.

```
TDoc in Fin-Aid no cats
```

This example writes technical documentation for dataset Fin-Aid at the terminal, omitting the documentation of categories.

```
TD in A f1 thru f5, f10
```

This example writes technical documentation of dataset A, fields F1 through F5, and field F10, at the terminal.

```
TD cats of Race in Students
```

This example writes technical documentation of the categories for field Race in dataset Students at the terminal.

Settings

None.

Modifiers

None.

Description

The Techdoc command produces technical documentation for one or more datasets, or fields or categories within a dataset. When a complete dataset is documented, a header like the following is printed.

```
Fin-Aid -- Example data set -- Financial aid information   09-11-81/ILIR

        Not Scrambled     Not Destroyable     Not Replaceable

  Dictionary file = ILIR:FIN-AID#      Micro Data file = ILIR:FIN-AID
```

The dataset header gives the name of the dataset, its description, creation date and owner, its scrambled, destroyable and replaceable status, and its files. When '`IN`' appears on the command, the dataset header is omitted and only fields are documented. In any case, fields and categories are documented as shown below.

```
-------------------------------------------------------------------------
 F#  Field name        Abbr          Value  Type Length Scale Factor Disp
-------------------------------------------------------------------------
 F4  Class-Level       CL          DataReq  UC        1                 6
              4 categories (required)
        Freshman       Fr                1
        Sophomore      So                2
        Junior         Jr                3
        Senior         Sr                4
 F7  Amount            Amt         DataReq  S         4  /       100    9
```

For fields, F# through Abbr are self-explanatory. The Value is the data required required status. The Type is a one or two-letter abbreviation of the Micro field type. The Length is the length in bytes of the field. The Scale is the function by which field values are scaled, / for division in the case of Field 7. The Factor is the term by which the function is applied, i.e., divided by 100, etc. If the field is not scaled, these two columns are blank. The Disp is the displacement in the record to the field, in bytes.

When categories are documented, each category occupies a row also. The F# column is blank. The Field name and Abbr contain the category name and abbreviation respectively. The Value is the value of the category. The Type is Default for default categories, otherwise blank. If categories are not documented, only the line telling how many categories the field has and if they are required appears.

UPDATE Micro Command

Purpose: To update selected fields in selected records in a dataset.

Synonyms: None.

Effect on the Result: If PERMANENTLY is specified then the Result dataset is not affected; the updates are made directly in the specified dataset. Otherwise a new Result dataset is created, replacing the existing Result. This new Result consists of the dataset named on the command, with the updates applied.

Usage

```
Update dataset [IDS = <fields> [, <fields> ] ...]
               [UPS = <fields> [, <fields> ] ...]
               [PERMANENTLY]
               [FROM input_file [WITH ERRORS ON error_file] ]
               [{OK | ! }]

Update IN dataset <fields> [, <fields>] ...
                  [IDS = <fields> [, <fields> ] ...]
                  [PERMANENTLY]
                  [FROM input_file [WITH ERRORS ON error_file] ]
                  [{OK | ! }]
```

where

<fields> can be any of:

field
pattern
field THRU *field*

Examples

```
Update Students
```

This command creates a Result dataset. When Micro asks, you supply the names of the fields to be used to identify records and the names of the fields to be updated. You then enter the data for these fields interactively as in the Enterdata command. Micro uses this data to perform the updates, yielding a Result dataset containing all of the fields and records from the Students dataset, but with the updates applied.

```
Update in Students  Name  id=SIN
```

This creates a Result dataset containing all of the fields and records from the Students dataset, with updates applied to the Name field. The SIN field is used to identify which records are to be changed. The data you enter interactively for the SIN and Name fields is used to perform the updates.

```
Update Fin-Aid perm from FA/Up with errors on -err
```

This updates the Fin-Aid dataset permanently. Micro asks for the names of the fields to identify records and then for the fields to be updated. The update data for these fields, which must be in the free format of the Enterdata command, is read from the input file FA/Up. Any records from FA/Up that are rejected are placed in the error file -ERR.

Settings

None.

Modifiers

None.

Description

The Update command requires *identifying* fields, to determine which records are to be changed, and *update* fields, to determine which fields to change. An identifying field may also be updated. The first form shown in the Usage section, '`Update` *dataset* `ids=...  ups=...`', indicates the identifying and update fields. In the second form, '`Update in` *dataset* `...  ids=...`', the update fields follow the dataset name, and the identifying fields are given in keyword fashion. If either group of fields is omitted, it is prompted for.

If '`PERMANENTLY`' is given on the command, updates are applied to the permanent data file of the dataset. If permanent updating causes duplicate records to be weeded from a dataset, confirmation is prompted for. This may be suppressed by appending '`OK`' or '`!`' to the command. If a permanent update fails for some reason (such as inadequate file space) the updated set is preserved as the temporary set Updated_Dataset. A temporary dataset may not be updated permanently. If permanent updating is not specified, the updated set is the Result set.

The list of identifying and update fields constitutes an intermediate temporary set. Input data, whether entered interactively or read from a file, must correspond in type and size to the id and update fields, in the order given.

If '`from` *input_file*' is given, data are read from a file. If an input file is given, errors are written to a file, specified with '`with errors on` *error_file*'. If file names are omitted, data are entered, and errors corrected, interactively.

The Update command is also available in a stand-alone version, ILIR:Micro.Up. Like Update, Micro.Up can also read data from and print errors on a file. Behavior is the same, except for some initial prompting by Micro.Up. This is described in the section on ILIR:Micro.Up (and Micro.Add), under Data Entry and Updating, in Part I. The requirements for file and interactive data entry, and the process of error correction, common to Micro.Up and Update, are described in the sections on Interactive and Free Format, under Data Entry and Updating, in Part I.

WRITE Micro Command

Purpose: To write data as a Micro data file; to write data to a file as an array of records; to write data for analysis by the MIDAS statistical program on MTS.

Synonyms: None.

Effect on the Result: None.

Usage

`Write` *dataset* `[[NOT] SCRAMBLED] ON` *file*

`Write` *dataset* `ARRAY ON` *file*

`Write` *dataset* `FOR MIDAS ON` *file*

Examples

```
Write it on Fin-Aid85$
```

This command writes the data from the Result set on the Micro data file Fin-Aid85$.

```
Write X array on -Xarray
```

This command writes the data from dataset X in array format on the file –XARRAY, which is created as a temporary sequential file.

```
Write it for midas on -stat1
```

This command writes the data from the Result set on –STAT1MD and generates a MIDAS command file that refers to it on –STAT1.

Settings

None.

Modifiers

None.

Description

The Write command writes a dataset to a file in one of three formats: as a Micro data file, in array format, or for reading by MIDAS. It is best to let the Write command create the files it needs. Sequential files are required for the Micro data file and recommended for data in array format.

If neither `ARRAY` nor `FOR MIDAS` is given, Micro writes a Micro data file. This file can be read later using the Micro Read command.

When writing a Micro data file, Micro needs to know if the data in the specified file is to be scrambled. If `SCRAMBLED` is given or if the dataset is scrambled, Micro asks for a scramble key. A null reply to the scramble key prompt tells Micro that the data is not to be scrambled in the file. The following command forces a prompt for a scramble key:

```
write Students on -sfile scrambled
```

The file can be read later with the command

```
read from -sfile unscrambled using Students
```

If the dataset is scrambled but the Micro data file to be written should not be scrambled, the prompt can be suppressed by specifying `NOT SCRAMBLED` in the command. This command causes no scramble key to be prompted for and none applied:

```
write Students on -sfile not scrambled
```

The file can be read later with the command

```
read from -sfile not unscrambled using Students
```

If `ARRAY` is given, Micro writes binary data (not scrambled) to the file in "array" format, each line containing an integral number of records. The file written by the command

```
write array students on -sarray
```

can be read later with the command

```
read array from -sarray using students
```

If `FOR MIDAS` is given, Micro builds a MIDAS Read command in the specified file (e.g., –STAT1) and writes the data to a second file, the MIDAS data file. The name of the data file is the specified file name with 'MD' appended (e.g., –STAT1MD). Labels, tags, and scaling information are included in the MIDAS Read command, along with a reference to the data file. The files generated by the command

```
write for midas on -stat1
```

can be used later in MIDAS (after stopping Micro) by issuing the MTS command

```
$Run STAT:MIDAS input=-stat1
```

A type U or UC field not scaled by '*' or '/' is passed as a categorical variable. If categories are required, the level is the largest category value, otherwise the level is the maximum value as determined by the field length. Therefore, the setting of a field's CategoriesRequired switch is important if you wish to stratify on that field in MIDAS. If the field is scaled by '+' or '-', this is used to set up a translation factor in MIDAS.

A type S, SC, or E field (the line number) is passed as an analytical variable. Any numeric field scaled by '*' or '/' is also passed as an analytical variable along with its scaling information.

If you have zeros in type U or UC fields that should not be treated as missing data, you can use the Calculate command to generate type S or SC fields that will be passed as analytical variables. Zero values in categorical variables are treated as missing data in MIDAS.

A type C or CC field is passed as an analytical variable. Thus, these fields are of no use in MIDAS.

Micro field abbreviations become MIDAS labels; category abbreviations become tags. If an abbreviation is an illegal label or tag in MIDAS, Micro asks for a replacement. This only affects the MIDAS Read command file; the Micro dataset is not changed.

ILIR:MicDef

ALTER MicDef Command

Purpose: The Alter command makes changes to existing dataset information, field definitions, or category definitions in the active dataset. It also modifies run-time settings used by Micro to perform command tracing and to locate a MPF library or runfile.

Synonyms: Change

Usage

Alter *field* <fitem> **TO** *newvalue*

Alter IN *field category* <citem> **TO** *newvalue*

Alter <ditem> **TO** *newvalue*

Alter <useritem> **TO** *newvalue*

Alter CCID FROM *userid* **TO** *userid*

The <fitem> (field item) may be one of **NAME**, **ABBReviation**, **LENgth**, **MAXvalue**, **CATSrequired**, **DATArequired**, **FACtor**, **FUNction**, **EXTernal**, or **DESCription**.

The <citem> (category item) may be one of **NAME**, **ABBReviation**, **DEFault**, **VALue**, or **DESCription**.

The <ditem> (dataset item) may be one of **DSN**, **DICTionary**, **DATAfile**, **FORMatfile**, **DESTroystatus**, **REPLacestatus**, **SCRAMblestatus**, or **DESCription**.

The <useritem> may be one of **SYSID**, **RUNFILE**, **LCMT**, or **MCMT**.

The *newvalue* is the value to be assigned to the item.

The *userid* is a four-character MTS userid.

The first form alters an item in a field definition. The field type is the only item that cannot be changed with the Alter command (you must use the Redo command to change this). *Note: if data already exists for the active dataset, changing the field length (maximum value) makes the dictionary and data files incompatible.*

The second form alters an item in a category definition. Any item may be altered. This command does not effect the compatibility with an existing data file.

The third form alters an item in the dataset information. Changes made to the dataset information are immediately written to the directory file, as opposed to changes made to the field and category information, which are written to the dictionary file only by the Write, Stop, and Edit commands. Caution: changing the dictionary file name may activate a new dictionary.

The fourth form alters one of the useritems. This form of the Alter command does not operate on the active dataset. These changes are immediately written to the directory file.

The fifth form alters a userid for the files associated with the active dataset (dictionary file, data file, and conversion file). It does not create or rename any files. These changes are immediately written to the directory file.

Examples

```
alt race maxvalue to 277
```

This changes the maxvalue, which is used to change the size of the field.

```
ch in sex missing name to noreply
```

This changes the Missing category of field Sex to Noreply.

```
alt dict to ILIR:STUDENTS#
```

This changes the dictionary file to ILIR:STUDENTS# for the active dataset.

```
change runfile to ilir:runfile
```

This changes the runfile, which contains commands to be executed when Micro is started, to ILIR:RUNFILE.

```
change ccid from ilir to stuv
```

This changes the userid of the active dataset files from ILIR to STUV.

APPEND MicDef Command

Purpose: The Append command adds fields (columns) or categories (tags associated with data values) to the dataset definition.

Synonyms: None.

Usage

```
APpend [*]
APpend [FROM file] field [*]
```

The first form appends fields to the active dataset. For each new field, MicDef asks for the field information appropriate to its field type and, if a categorical field, its category information. If the asterisk (*) is specified, MicDef does not ask for the data required status, categories required status, scaling information, or categories; it assumes data not required, categories not required, no scaling, and no categories. *Note: if data already exists for the active dataset, appending new fields makes the dictionary and data files incompatible.*

The second form appends categories to the specified field. For each new category, MicDef asks for the category name, category abbreviation, default status, category value, and description. If the asterisk is specified, MicDef does not ask for the default status; it assumes that none of the categories appended are default categories. If '`FROM` *file*' is specified, the categories are read from a file containing one category per line with the category items separated by at least one blank.

Examples

```
APPEND
app *
```

These examples add fields to the dataset.

```
APPEND ethnicity
```

This example appends categories to the field Ethnicity.

```
APP from catfile ethnicity
```

This example appends categories to the field Ethnicity from the file CATFILE.

COPY MicDef Command

Purpose: The Copy command copies fields from a dataset to the active dataset.

Synonyms: None.

Usage

```
COpy FROM dataset {field|pattern|field THRU field} ... [TO field]
```

The specified fields are copied from the specified dataset to the active set, where they are positioned after the 'TO' field. If 'TO *field*' is omitted, the fields are copied to the beginning of the active dataset definition. *Note: if data already exists for the active dataset, copying new fields makes the dictionary and data files incompatible.*

If a copied field is categorical, the categories are also copied. If the copied fields shared categories in the specified dataset, the categories are not shared in the active dataset.

Examples

```
COpy from students SIN F3 GENDER TO FL
```

This examples copies SIN, Field 3, and Gender from dataset Students to a position following FL (the last field) in the active dataset.

```
COpy FROM fin-aid ?e?
```

This example copies from dataset Fin-Aid all fields containing an 'e' and positions them at the beginning of the active dataset.

DELETE MicDef Command

Purpose: The Delete command deletes fields or categories from the active dataset.

Synonyms: None.

Usage

`DELete` {*field*|*pattern*|*field* `THRU` *field*} ...

`DELete IN` *field* {*category*|*pattern*|*category* `THRU` *category*} ...

The first form deletes the specified fields. *Note: if data already exists for the active dataset, deleting fields makes the dictionary and data files incompatible.*

The second form deletes the specified categories.

Examples

```
DELETE f7 f9 thru comments
```

This example deletes Field 7, and Fields 9 through Comments.

```
DEL ssn
```

This example deletes field Ssn.

```
DEL in f7 c1 thru cL
```

This example deletes all categories in Field 7.

```
DEL in ethnicity Ot?
```

This example deletes all categories in field Ethnicity beginning with '`Ot`'.

DESCRIBE MicDef Command

Purpose: The Describe command displays dataset information for any dataset in your directory. It also displays the name of the active dataset.

Synonyms: None.

Usage

```
Describe [*]
Describe {dataset|pattern} [*]
Describe SETS
```

For each dataset, MicDef displays the dataset name, description, dictionary file name, data file name, conversion file name, destroy status, replace status, and scramble status. If the asterisk (*) is specified, MicDef lists only the dataset names, not the other dataset information. Then it displays the name of the active dataset. Note: describing a dataset does not make it the active dataset

Examples

`Desc`

This describes the active dataset.

`Describe *`

This displays the name of the active dataset.

`D stu? *`

This displays the names of all datasets beginning with '`stu`'.

`D SETS`

This describes all datasets.

DESTROY MicDef Command

Purpose: The Destroy command destroys the active dataset.

Synonyms: None.

Usage

```
DESTroy
```

The Destroy command destroys the active dataset. If the dictionary file is not used by another dataset, it is also destroyed. The data file is not changed. There is no active dataset following this command (use the Edit command to activate another dataset).

Examples

```
dest
```

DOC MicDef Command

Purpose: The Doc command prints the field and category information on the terminal or on a file.

Synonyms: None.

Usage

`DOc` [`ON` *fdname*] {*field*|*pattern*|*field* `THRU` *field*} ... [`*`]

`DOc` [`ON` *fdname*] `IN` *field* {*category*|*pattern*|*category* `THRU` *category*} ...

The first form shows descriptive information for the specified fields and their categories. If the asterisk (*) is specified, the category information is omitted. If no fields are specified then all fields are shown. Up to 511 fields may be specified.

The second form shows the descriptive information for the the specified categories. Up to 511 categories may be specified.

Examples

```
Doc ON *PRINT*
```

This example documents all fields and categories on *PRINT*, the pseudodevice for an MTS printer.

```
Doc f1 ssn f5 date f7 thru f9
```

This example documents Field 1, Ssn, Field 5, Date, and Fields 7 through 9.

```
Doc *
```

This example documents all fields, omitting the category information.

```
Doc in f7 c1 female thru cL
```

This example documents category C1, and categories Female through the final category, in Field 7.

```
Doc in f7 m?
```

This example documents all of the categories in the Field 7 that begin with the letter '`m`'.

EDIT MicDef Command

Purpose: The Edit command makes permanent any changes you have made to the field and category information, and then activates another dataset.

Synonyms: DEFine

Usage

`EDit [FULL]` *dataset*

If any changes have been made to the field and category information in the active dataset, they are written to the dictionary file before the specified dataset is made the new active dataset.

If the specified dataset already exists, it becomes the active dataset. If FULL is specified in this case, it is ignored.

If the dataset does not exist, MicDef creates it. If FULL is omitted, MicDef only asks for the description for the new dataset. It assumes default values for the dictionary file name, data file name, conversion file name, destroy status, replace status, and scramble status. If FULL is specified, MicDef asks for all of the dataset information, allowing you to override the defaults. The new dataset becomes the active dataset and MicDef proceeds to ask for its field and category information.

Examples

```
edit students
```

This example makes the existing dataset Students the active dataset.

```
ed newdataset
```

This example creates the new dataset Newdataset, asks only for a description, and makes it the active set.

```
ed full newdataset
```

This example creates the new dataset Newdataset, asks for all dataset information, and makes it the active set. If, in response to the dictionary file prompt, an existing dictionary is given, the new dataset uses that file, and no field information is prompted for. This is the method in MicDef for creating a new dataset that shares the structure of an existing one.

FDOC MicDef Command

Purpose: The Fdoc command prints the full documentation (both technical and descriptive) on the terminal or on a file.

Synonyms: Show

Usage

FDoc [ON *fdname*] [{*field*|*pattern*|*field* THRU *field*} ...] [*]

FDoc [ON *fdname*] IN *field* [{*category*|*pattern*|*category* THRU *category*} ...]

The first form prints both technical and descriptive information for the specified fields. If the asterisk (*) is specified, the category information is omitted. If no fields are specified then all fields are shown. Up to 511 fields may be specified.

The second form prints the technical and descriptive information for the specified categories. Up to 511 categories may be specified.

Examples

`fdoc on *print*`

This example documents all fields and categories on *PRINT*, the pseudodevice for an MTS printer.

`fd f1 ssn f5 date f7 thru f9`

This example documents Field 1, Ssn, Field 5, Date, and Fields 7 through 9.

`fd *`

This example documents all fields but without their category information.

`fd in f7 c1 female thru cL`

This example documents categories C1, and Female through the last (cL) in Field 7.

`fd in f7 m?`

This example documents categories in Field 7 which begin with the letter 'm'.

GET MicDef Command

Purpose: The Get command transfers datasets to the current userid from another userid.

Synonyms: None.

Usage

`Get {COPY|ORIGINAL}` *{dataset|pattern}* **`FROM`** *userid*

The specified datasets are transferred from the specified userid to the current userid. If `COPY` is specified, MicDef copies the datasets to the current userid, leaving the originals intact. If `ORIGINAL` is specified, MicDef copies the datasets to the current userid and then deletes the originals.

Transferring datasets from one userid to another is a two step process. First, you must run MicDef on the userid containing the datasets and use the Permit command to grant the necessary file access. Then run MicDef on the userid to which the datasets are to be transferred and use the Get command to transfer the files.

Examples

```
get copy students from ilir
```

This example gets a copy of the Students dataset from ILIR.

```
get original budget? from stuv
```

This example moves all datasets beginning with '`budget`' from userid STUV to the current userid (assuming the correct file permissions have been obtained using the Permit command).

HELP MicDef Command

Purpose: The Help command provides on-line help for MicDef.

Synonyms: None.

Usage

```
HELP
```

The Help command displays a menu of the MicDef commands. Choosing one produces help information for the command.

```
-*-
1. Alter command       7. DOcument command  13. Menu command
2. APPend command      8. EDit command      14. Permit command
3. COpy command        9. Get command       15. Redo command
4. DELete command     10. HELP command      16. Show command
5. Describe command   11. Insert command    17. Write command
6. DESTroy command    12. Move command
-*-
```

In menu mode, each MicDef menu includes a Help item. Selecting it displays a Help menu of topics corresponding to the choices in the MicDef menu. For example, choice 7 in the Dataset Menu is the Help item. Choosing it displays the following menu:

```
-*-
1. List existing datasets          4. Other/misc commands            7. Stop
2. Activate a set and edit fields  5. Find datasets matching pattern
3. Change/describe dataset info    6. Exit menu
-*-
```

Examples

```
help
```

INSERT MicDef Command

Purpose: The Insert command puts new fields (columns) into the active dataset or new categories (tags associated with values) into a field.

Synonyms: None.

Usage

`Insert` [*field*]

`Insert IN` *field* [*category*]

The first form inserts new fields after the specified field. For each new field, MicDef asks for the field information appropriate to its field type and, if a categorical field, its category information. If the *field* is omitted, the new fields are inserted at the beginning of the list. *Note: if data already exists for the active dataset, inserting new fields makes the dictionary and data files incompatible.*

The second form inserts new categories after the specified category. For each new category, MicDef asks for the category name, category abbreviation, default status, category value, and description. If the *category* is omitted, the new categories are inserted at the beginning of the list.

Examples

`Insert`

This example inserts new fields at the start of the dataset.

`insert ssn`

This example inserts new fields after field SSN.

`insert in ethnicity other`

This example inserts new categories following category Other in field Ethnicity.

`insert in F7 CL`

This example inserts new categories after the last category in Field 7.

MENU MicDef Command

Purpose: The Menu command puts MicDef into menu mode.

Synonyms: None.

Usage

```
Menu
```

MicDef goes into *menu mode*. You can then tell MicDef what to do by selecting from menus rather than by entering commands.

Examples

```
Menu
```

MOVE MicDef Command

Purpose: The Move command moves fields from one location in the field list to another, or moves categories from one field to another.

Synonyms: None.

Usage

`MOve` {*field*|*pattern*|*field* `THRU` *field*} ... [`TO` *field*]

`MOve IN` *field* {*category*|*pattern*|*category* `THRU` *category*} ... `TO` *field* [*category*]

The first form moves the specified fields to a location following the '`TO`' field. If '`TO` *field*' is omitted, the location is the beginning of the dataset. This command is equivalent to deleting the fields and then inserting them after the '`TO`' field. *Note: if data already exists for the active dataset, moving fields makes the dictionary and data files incompatible.*

The second form moves the specified categories to the '`TO`' field. This command is equivalent to deleting the categories from the '`IN`' field and then inserting them in the '`TO`' field after the category specified. If the '`TO`' category is omitted, the categories become the first in the field.

Categories in type UC or SC fields can to be moved to another numeric field with a different type and length. Categories in type CC fields can be to be moved to another field of the same type but different length. Moving categories between character and numeric fields is not allowed.

Examples

```
move ssn race thru f6 to comments
```

This example moves fields Ssn and Race through Field 6 to just after field Comments.

```
move in race c1 thru cl to ethnicity
```

This example moves categories 1 through L (last) from field Race to the beginning of the category list in field Ethnicity.

```
move in race o? to ethn c1
```

This example moves categories beginning with 'o' in field Race to just after category 1 in field Ethn. If `c1` is omitted, the categories are moved to the beginning of the category list.

PERMIT MicDef Command

Purpose: The Permit command changes the access that other userids have to the specified datasets.

Synonyms: None.

Usage

`PErmit {COPY|ORIGINAL|UPDATE|NONE} {`*dataset*`|`*pattern*`} TO {ALL|`*userid*`|P=`*userid*`}`

If `COPY` is specified, the specified userids are given read access to the specified datasets. This access is necessary to allow the `GET COPY` command to copy the datasets. It also allows Micro to read the datasets.

If `ORIGINAL` is specified, the specified userids are given unlimited access to the specified datasets. This access is necessary to allow the `GET ORIGINAL` command to move the datasets.

If `UPDATE` is specified, the specified userids are given read/write access to the specified datasets. This access is necessary to allow Micro's Enterdata or Update commands to modify the datasets from other userids.

If `NONE` is specified, the specified userids are denied access to the specified datasets.

The specified userid can be a single *userid*, `ALL` userids on MTS, or only the userids associated with a single MTS project.

Examples

```
permit original students to stuv
```

This example gives userid STUV unlimited access to dataset Students.

```
permit copy fin? to all
```

This example allows all userids to read the datasets beginning with '`fin`'.

```
permit update students to p=spqr
```

This example allows all userids in project SPQR to make changes to the Students dataset.

```
permit none students to all
```

This example denies all userids access to dataset Students.

REDO MicDef Command

Purpose: The Redo command replaces all of the information for a dataset, field, or category.

Synonyms: None.

Usage

`Redo`

`Redo` *field*

`Redo IN` *field category*

The first form changes all of the dataset information for the active dataset. MicDef asks for a new dataset name, dictionary file name, data file name, conversion format file name, destroy status, replace status, scramble status, and description. The new dataset information is immediately written to the directory file, as opposed to changes made to the field and category information, which are written to the dictionary file only by the Write, Stop, and Edit commands.

The second form changes all of the field information for the specified field. MicDef asks for a new field name, abbreviation, type, length, scaling, data required status, and description. Categories for the field may be lost if the type or length change. *Note: if data already exists for the active dataset, changing the field type or length (maximum value) makes the dictionary and data files incompatible.*

The third form changes all of the category information for the specified category. MicDef asks for a new category name, abbreviation, default status, value, and description.

Examples

```
redo
```

This example redoes the dataset information for the active dataset.

```
redo ethnicity
```

This example redoes the field information for field Ethnicity.

```
redo in ethnicity cl
```

This example redoes the category information for the last (CL) category in field Ethnicity.

STOP MicDef Command

Purpose: The Stop command makes permanent any changes you have made to the field and category information, and then stops Micdef.

Synonyms: None.

Usage

```
STop
```

If any changes have been made to the field and category information in the active dataset, they are written to the dictionary file. Then MicDef stops, returning control to MTS.

Examples

```
Stop
```

TDOC MicDef Command

Purpose: The Tdoc command prints the field and category information on the terminal or on a file.

Synonyms: None.

Usage

`TDoc` [`ON` *fdname*] {*field*|*pattern*|*field* `THRU` *field*} ... [`*`]

`TDoc` [`ON` *fdname*] `IN` *field* {*category*|*pattern*|*category* `THRU` *category*} ...

The first form shows technical information for the specified fields and their categories. If the asterisk (*) is specified, the category information is omitted. If no fields are specified then all fields are shown. Up to 511 fields may be specified.

The second form shows the technical information for the the specified categories. Up to 511 categories may be specified.

Examples

`Tdoc ON *PRINT*`

This example documents all fields and categories on *PRINT*, the pseudodevice for an MTS printer.

`Tdoc f1 ssn f5 date f7 thru f9`

This example documents Field 1, Ssn, Field 5, Date, and Fields 7 through 9.

`Tdoc *`

This example documents all fields, omitting the category information.

`Tdoc in f7 c1 female thru cL`

This example documents category C1, and categories Female through the final category, in Field 7.

`Tdoc in f7 m?`

This example documents all of the categories in the Field 7 that begin with the letter '`m`'.

WRITE MicDef Command

Purpose: The Write command makes permanent any changes you have made to the field and category information.

Synonyms: CLose

Usage

```
WRITE
```

If any changes have been made to the field and category information in the active dataset, they are written to the dictionary file. This is done automatically by the Edit and Stop commands.

Examples

```
Write
```

ILIR:Micro.Form

CLOSE Micro.Form Command

Purpose: To store an up-to-date permanent copy of the conversion table in the conversion format file, thus saving the format specification work completed during the current session. After closing the conversion format file, `CORRECTIONS` mode is reentered, in which the format may be further edited.

Synonyms: None.

Usage

```
CLose
```

Examples

```
close
```

This example saves the current format in the conversion format file.

DELETE Micro.Form Command

Purpose: To delete one or more recode specifications for a field, without deleting the field format information.

Synonyms: None.

Usage

`DElete` *field* [*number*]

If a recode *number* is specified, only it is deleted; else all recodes for the field are deleted. (To change information for a field, use the Redo command). To see the recodes for a fiel, use the Document command.

Examples

```
del sic-code
```

This example deletes all recodes for field Sic-code.

```
del sic-code 2
```

This example deletes the second recode for field Sic-code.

DOCUMENT Micro.Form Command

Purpose: To print conversion format information for the entire file; or conversion format information for a single field; or recode information for a field.

Synonyms: Show

Usage

`Document` [*field* [*number*]]

The *field* is the name of a field, and *number* the number of a recode defined for the field.

Examples

```
show
```

This example documents format information for the entire file.

```
doc idnum
```

This example documents format information for field Idnum.

```
doc sic-code 2
```

This example documents the second recode for field Sic-code.

RECODE Micro.Form Command

Purpose: To provide recode specifications for a field.

Synonyms: None.

Usage

`RECode` *field*

The program prompts for `RECODE TYPE` and other recode information, as described in the section on Data Entry. Entering a null reply (Return) at the first prompt finishes recode entry and returns to command mode.

Examples

```
rec sic-code
```

This example prompts for `RECODE TYPE` and other information as necessary.

REDO Micro.Form Command

Purpose: To redo the format specification for a single field or to redo the initial information for the whole format file.

Synonyms: None.

Usage

`Redo` [*field*]

If the *field* is given, the program prompts for new format information for the field (starting column, input type, etc.) If *field* is omitted initial information for the format file is prompted for (maximum total number of columns, maximum number of cards, etc.)

Examples

```
redo sic-code
```

This example redoes format information for field Sic-code.

```
redo
```

This example redoes initial information for the entire format file.

STOP Micro.Form Command

Purpose: To store the conversion table created during the current session in the conversion format file and to terminate the program.

Synonyms: None.

Usage

```
STop
```

Examples

```
stop
```

This example stores the conversion format in the conversion format file and returns to MTS.

Part IV

Micro Programming Facility

Introduction

The Micro Programming Facility provides an augmented version of the C language, with a syntax for declaring C functions to be invoked (and receive parameters) from Micro command level, and for building and executing Micro commands. C functions are provided to access Micro's internal run-time data structures, which hold dataset, field and category attributes, and individual data records. The Micro command processor itself can be accessed, allowing an MPF program to be a Front End Processor for Micro, handling all interaction with the user. Exceptional conditions in MPF code can be distinguished from conditions in Micro, and handled conveniently.

MPF source code is interpreted by a preprocessor, which generates a file containing C target code and macros for the special syntax, as well as any lines of regular C code. C header files <string.h>, <stdlib.h>, <stdio.h>, <signal.h>, <ctype.h>, <setjmp.h>, and <mts.h> are included in the target code, as are header files containing typedefs, function declarations and other information used by the target code. C `#line` preprocessor directives ensure that error messages resulting from compilation of the target code correspond to the line numbers of the source code, assuming there are no fractionally-numbered lines in the source code.

The file ILIR:Demo.MPF contains examples of MPF routines, in addition to those shown here. The C language itself is best documented by Kernighan and Ritchie in *The C Programming Language*, second edition, which is based on the American National Standards Institute proposed C standard. The MTS C compilers implement the ANSI standard.

MPF Micro Commands

This declares an *MPF Micro command*, a C function which can be called from Micro command level:

```
micro report rgen {
  printf("Printing report\n") ;
}
```

The word '`report`' is the name of the MPF command, used at the Micro command prompt. Typing '`report`' causes the message to be printed (by the `printf` function from the standard C library). '`rgen`' is the C name of the function, by which it is declared in the target code generated by the MPF preprocessor. (The opening brace is optional, supplied by the preprocessor if omitted).

Here is the same MPF command, declared with parameters:

```
micro report rgen(int ncopies=1, char *title, -
   char *sponsor="XYZ Institute", float version=)
printf("Printing report: copies=%d  title=%s  sponsor=%s  version=%f\n",
  ncopies, title, sponsor, (double)version) ;
}
```

The MPF and C names are as before. Following the C name, enclosed in parentheses, is the parameter list. A parameter followed by an equal sign (such as Ncopies in this example) is optional and may be omitted from the invocation of the MPF command. An optional parameter which has a value following the equal sign (like Ncopies) will have that value assigned, if one is not supplied on the invocation. An initial value containing a blank must be enclosed in quotes. A dash (-) at the end of a line continues a `micro` declaration to the next line.

Parameters are assigned by combining the information in the declaration with that in the invocation at the Micro command prompt:

```
Ready:
-report title="Current Research Projects"  5
```

The word '`title`' identifies the first parameter assignment. The named parameter is assigned the value. The second value, '`5`', is assigned to the first parameter unassigned so far, Ncopies. The parameter Sponsor, which is omitted from the invocation, has its declared initial value, while Version, also omitted, has its default, the conversion of a C null string to `double`, which is 0.0 on MTS.

In general, if '*par=value*' is in the invocation, the named parameter is assigned the value (as with Title). Otherwise, if an unnamed *value* is in the invocation ('`5`' in the example), it is assigned to the next unassigned position in the parameter list. If a parameter is declared as '*par=value*' (as with Sponsor), it is assigned the *value*. If a parameter is declared as '*par=*' (as with Version), it is optional, and if omitted from the invocation is assigned a default value, `'\0'`, the C null string, or whatever a conversion routine does with that value. If a parameter is declared without an equal sign it is required, and if omitted from the invocation it is prompted for.

Arrays are not permitted in MPF command calling sequences, with one exception. The calling sequence '`char *`*ident*`[]`', where *ident* is a C identifier, is recognized. Character strings on the invocation, rather than being assigned or converted to local variables, are inserted as `char *` values into the array, starting at position 1 (the MPF command name is at position 0), for direct access. A NULL pointer denotes the end of the parameter list. An MPF command can examine the command line if desired, with the convenience of having the line broken up in advance, as follows.

The material after the MPF command name on the invocation is divided into tokens. A character string separated by blanks from adjacent material is a single token. A character string containing embedded blanks must be enclosed in single or double quotes (`'...'` or `"..."`). Several punctuation characters

```
 ,  ;  <  =  >  |  &  !
```

also delimit strings, and are themselves distinct tokens, occupying their own location in the `char *` argument vector.

For example, if the calling sequence for `report` had been declared as '`char *argv[]`', the above invocation would cause `argv[0]` to hold '`report`' (the MPF command name), `argv[1]` '`title`', `argv[2]` '`=`', `argv[3]` '`Current Research Projects`', `argv[4]` '`5`', and `argv[5]` the C value '`NULL`'.

The MPF preprocessor saves information on MPF commands in a library file, distinct from the object code file. When Micro is $Run, the *master userid*, which defaults to the user's id, is checked for such a file, named MPFLIB. If it exists its contents are activated and the MPF commands in it are available automatically.

If the library file is not available this way, it can be specified with the Micro Set command:

```
set mpf: lib reportmpflib
```

More than one such library of MPF commands may be active at once. Libraries are searched for commands in order of activation. If a library is already present when activated, it is deactivated and activated anew, reflecting the result of any changes since the last activation. All libraries may be deactivated by replacing the file name in the example with '`off`'.

The preprocessor turns the MPF declaration into the beginning of a `void` C function with a `void` argument list, or a list consisting of a single argument, an array of char pointers. Each parameter is declared as a local variable, initialized from a position in the `char *[]` array corresponding to its position in the declaration, directly or using an appropriate conversion routine for non-`char *`. In addition to the types shown here, `long`, `short`, `double`, and `signed` or `unsigned` are recognized. The target code for this example, and all other examples, can be found at the end. Note the absence of an opening brace ({) in those examples, which is provided by the target code.

Reserved Words

The following words should not be used as C identifiers, because they cause special action by the MPF preprocessor, or are C macros expanded by the C preprocessor:

```
build          end        lock          release          update
calculate      enterdate  longjump      remove           write
calc           execute    micall        replace          xtab
call           explain    micro         reset
cancel         export     mts           restrict
change         extcall    name          restrictandjoin
combine        fep        pgnt_info     save
comment        find       pgnt_message  select
crosstabulate  fulldoc    plot          set
datestamp      get        print         setjump
describe       handler    prompt        sort
destroy        join       purge         stop
display        key        raj           techdoc
document       list       read          unload
```

Most of the words are Micro commands or abbreviations; at the start of a line, they begin a Micro command. The Micro commands and certain others (**build**, **fep**, **handler**, and **micro**) invoke the MPF preprocessor. Recognition of these words is case-insensitive. The others (**cancel**, **execute**, **longjump**, **pgnt_info**, **pgnt_message**, **prompt**, **setjump**, **stop** and **unload**) are C macros defined in lower case, and must appear that way. A colon (**:**) at the start of a line also has special significance, as explained in the next section.

Many other terms and names are pre-defined, but most of them are functions which begin with an underscore. The remainder are C typedefs, enumerations and define terms which are entirely capitalized. The function prototypes are in ILIR:MPFLIB.H, most define terms and typedefs are in ILIR:MPF.DEF, and the remainder are in ILIR:MPF.H, which is included in the target code, and includes the other two. The file ILIR:PGNT.H contains macros and data structures for handling program interrupts, and is of use in large, complex applications, in conjunction with an MPF **handler**, as described in the section on Exceptional Conditions, below. It is not included automatically in the target code.

Generating Micro Commands

Source code may be used to generate Micro commands, and pieces of commands. Source code is recognized as beginning a Micro command if a source line begins with the full name (or a couple of common abbreviations) of a Micro command, or with the word '**build**', the command text starting after that word.

Source code which begins with a colon (**:**) is added to the existing Micro command (without the colon). All types of Micro command text—lines beginning with command names, with '**build**', or with a colon—may be continued to the next source line by ending the line with a dash (-). Micro commands may be followed by modifiers, as in '`print@count=...`'.

All types of Micro command text may contain expressions in braces, which are added to the existing command by the target code, if they evaluate to type `char *`. (If not, a serious run-time error can result.)

Once complete, a Micro command must be explicitly executed. Or it may be explicitly cancelled. The following examples show the possibilities.

```
sort {ds_name} by -
   {field_name}
execute
```

The target code formats the text and the contents of the braces into a global command buffer, `_microcmd`, which `execute` passes to Micro for execution. In the above example, the command buffer would contain '`sort % by %`', where the two percent signs stand for the contents of `ds_name` and `field_name` respectively.

After each use of `execute` a global `int` variable `_micrortn` is set to reflect the result of the command execution. C definitions for the possible values are in the file ILIR:MPF.DEF. The value zero means successful execution. Non-zero values are discussed below, in the section on Exceptional Conditions.

The following example illustrates conditional command construction, using the colon notation.

```
find in Fin-aid where
if (! strcmp(field_name, "source-of-funds"))
  : source-of-funds = industry or UNIV or 3
else
  : Amt is between {lower} and {upper}
execute
```

The target code causes the command buffer to be filled with '`find in Fin-aid where`', and one of the two colon-prefixed strings, depending on whether the condition is true or false.

The following illustrates use of `build` and `cancel`.

```
build report title="Current Research Projects" 5
...
cancel
```

Note that, while '`report`' is not a Micro command, it can be placed in the command buffer by using the '`build`' notation. This is how one MPF command is invoked from another.

In addition to these, a `prompt` macro causes prompting by the Micro command interpreter, and a `stop` macro terminates execution of an MPF command and returns control to the Micro command prompt. These are illustrated in the FEP example in the section on Exceptional Conditions, below.

C comments may not be used on Micro command lines, for they will be emitted as part of the command. The preprocessor notices comments, *if* the begin comment characters '/*' are the first, and the end comment characters '*/' the last, non-blanks on the lines of a commented block. Lines within such a block, like 'micro' and 'calculate' in this example, do not cause special processing.

```
/* ...
micro ...
calculate ... */
```

An MPF Micro command is loaded dynamically and unloaded after it returns, subject to the setting of the global `int _unloadsw`. This switch is initially set to 1, meaning that code is unloaded. Unloading may be disabled by setting the switch to 0.

Micro Interface Functions

Introduction

The programming facility provides functions to access Micro datasets, fields and records. C prototypes and typedefs for these functions are in ILIR:MPF.H, ILIR:MPF.DEF and ILIR:MPFLIB.H; the first includes the latter two, and is included in the target code generated by the preprocessor. The C typedefs, which define types used as identifiers for datasets, fields, categories and records, are:

```
typedef int INTEGER ;
typedef unsigned int DATASET ;
typedef unsigned int FIELD ;
typedef unsigned int CATEGORY ;
typedef unsigned int RECORD ;
```

Several of the Micro interface functions are defined as type `INTEGER`. The Micro identifiers are currently implemented as fullwords, and use of typedefs will minimize the impact of any change to the underlying representation. Two other typedefs, for Micro field types and scaling functions, are discussed below.

Size definitions for arguments to some functions are also included. They allow for a terminating end-of-string, a definition for which is also included in the header files (`_EOS`). The use of these terms is noted below.

```
#define FILENAME_SIZE 21
#define DSNAME_SIZE 17
#define FLDNAME_SIZE 17
#define FLDABBR_SIZE 5
#define CATNAME_SIZE 12
#define CATABBR_SIZE 5
#define DSEVEC_SIZE 20
```

Dataset Functions

The **_ds_lookup** function returns an identifier for the dataset, or zero for failure:

```
DATASET _ds_lookup(char *ds_name) ;
```

Once a dataset has been looked up, attributes of the dataset can be obtained with the following functions.

```
INTEGER _can_destroy(DATASET ds_id) ;
INTEGER _can_replace(DATASET ds_id) ;
INTEGER _is_scrambled(DATASET ds_id) ;
char *_dict_file(DATASET ds_id, char *dict_name) ;
char *_data_file(DATASET ds_id, char *data_name) ;
char *_ds_name(DATASET ds_id, char *dsname) ;
char *_ds_desc(DATASET ds_id, char *desc_text) ;
INTEGER _fld_cnt(DATASET ds_id) ;
INTEGER _recd_cnt(DATASET ds_id) ;
```

The functions **_can_destroy**, **_can_replace** and **_is_scrambled** return 1 if the dataset has that property, else 0. Function **_dict_file** fills the character argument with the dictionary file name for the dataset, or **_EOS**, and returns it. Function **_data_file** returns the data file name, or **_EOS**. Function **_ds_name** returns the name of the dataset or **_EOS**. Function **_ds_desc** returns the dataset description or **_EOS**. Functions **_fld_cnt** and **_recd_cnt** return the number of fields in a record of a dataset, and number of records in a dataset, respectively. Function **_fld_cnt** locks the dataset's dictionary file to read (and leaves it locked). Function **_recd_cnt** locks the dataset's data and dictionary files to read.

The buffers supplied for **_dict_file** and **_data_file** should be of size **FILENAME_SIZE**; this term is defined in ILIR:MPF.DEF. The buffer supplied for **_ds_name** should be of size **DSNAME_SIZE**, also defined in ILIR:MPF.DEF.

Information can also be obtained to refer to fields:

```
FIELD _fld_lookup(DATASET ds_id, char *fld_name) ;
```

The **ds_id** in this prototype is one obtained by a call to **_ds_lookup**. Function **_fld_lookup** takes a valid dataset id and a field name, and returns a field id, or zero for failure. Function **_fld_lookup** locks the dictionary file to read.

Field Functions

Once the necessary lookups have been performed, field information can be obtained with:

```
INTEGER _data_reqd(FIELD fld_id) ;
INTEGER _cat_reqd(FIELD fld_id) ;
```

```
char *_fld_name(FIELD fld_id, char *fld_name) ;
char *_fld_abbr(FIELD fld_id, char *abbr) ;
INTEGER _fld_len(FIELD fld_id) ;
INTEGER _cat_cnt(FIELD fld_id) ;
INTEGER _max_val(FIELD fld_id) ;
INTEGER _min_val(FIELD fld_id) ;
FIELD _frst_fld(DATASET ds_id) ;
FIELD _next_fld(FIELD fld_id) ;
char *_ext_filename(FIELD fld_id, char *filename) ;
INTEGER _max_flen(char *filename) ;
char *_fld_desc(FIELD fld_id, int mts_fdub, char *desc_text) ;
FIELD_TYPE _fld_type(FIELD fld_id) ;
FIELD_SCFUNC _fld_func(FIELD fld_id) ;
INTEGER _fld_disp(FIELD fld_id) ;
INTEGER _fld_fact(FIELD fld_id) ;
```

The function **_data_reqd** returns 1 if data is required for the field, else 0.

The function **_cat_reqd** returns 1 if the data for the field must be one of a list of categories for the field, else 0.

Functions **_fld_name** and **_fld_abbr** return the field name and abbreviation respectively. The function value is the second argument, which points to or **_EOS** if the information cannot be obtained.

The function **_fld_len** takes a field id and returns the length in bytes of the field.

The function **_cat_cnt** returns the number of categories for the field.

Functions **_min_val** and **_max_val** return the minimum and maximum values for an integer field.

The functions **_frst_fld** and **_next_fld** are used to traverse the fields for a dataset. Function **_frst_fld** takes a dataset identifier and returns the identifier to the first field or zero, for no fields; it also locks the dictionary file to read. Function **_next_fld** takes a field identifier and returns the next field identifier, or zero, for no more fields.

Function **_ext_filename** takes a field identifier and a character buffer, and returns the name of the external file for the field.

Function **_max_flen** takes a file name and returns −1 for non-existence or any other error, 0 for empty, or a positive value, the maximum length of a line in the file. This allows the maximum size of an external field to be determined, from the external file name, analogous to **_fld_len** (using a field identifier) for a fixed-length field.

The function **_fld_desc** takes a field identifier, the MTS fdub for the dictionary file for the dataset, and a character buffer, and returns the description of the field.

The character arguments to **_fld_name** and **_fld_abbr** should be of size **FLDNAME_SIZE** and **FLDABBR_SIZE**, defined in ILIR:MPF.DEF.

The function **_fld_type** returns a value of type **FIELD_TYPE**, corresponding to the type of field. The C typedef and enumeration of possible values are included in the target code:

Enumerations for type FIELD_TYPE

Enumeration	Micro Type
`U = 1`	unsigned integer
`C = 2`	character
`UC = 3`	unsigned integer with categories
`CC = 4`	character with categories
`S = 5`	signed integer
`E = 6`	external
`SC = 7`	signed with categories

Function `_fld_func` returns a value of C type `FIELD_SCFUNC`, corresponding to the arithmetic operation applied to the field. The typedef and enumerations are included in the target code:

Enumerations for type FIELD_SCFUNC

Enumeration	Input	Output
`NOSCALE = 0`	value	value
`ADD = 1`	value $-=$ factor	value $+=$ factor
`SUBTR = 2`	value $+=$ factor	value $-=$ factor
`MULT = 3`	value $/=$ factor	value $*=$ factor
`DIV = 4`	value $*=$ factor	value $/=$ factor

A C enumeration enforces type checking, to restrict eligible values when a variable of the enumerated type changes. It is used here to prevent field types and scaling functions from being set to an illegal value, by assignment, or by one of the functions discussed below.

Function `_fld_fact` returns the field scale factor.

Function `_fld_disp` returns the displacement in bytes to the field in the record.

Record Functions

In addition to attributes of a dataset, individual records may be looked up and examined, once the dataset has been looked up, and opened:

```
void _ds_open(DATASET dsid, INTEGER dsevec[DSEVEC_SIZE]) ;
void _ds_clos(DATASET dsid, INTEGER dsevec[DSEVEC_SIZE]) ;
RECORD _recd_lookup(INTEGER recnum, INTEGER dsevec[DSEVEC_SIZE]) ;
char *_fld_strval(FIELD fld, RECORD rec, char *format, INTEGER fdub, char *string) ;
INTEGER _fld_ival(FIELD fld_id, RECORD recid) ;
double _fld_dval(FIELD fld, RECORD rec) ;
char *_fld_cval(FIELD fld_id, RECORD recid, char *cvalue) ;
```

Function `_ds_open` takes a dsid, obtained by `_ds_lookup`, and the `dsevec` vector, whose size definition, `DSEVEC_SIZE`, is included in the target code. The same vector is necessary to close the dataset and free storage through `_ds_clos`. Function `_recd_lookup` takes a record number, and a `dsevec` as filled by `_ds_open`, and returns a record id or zero for failure.

Once a field id and record id have been obtained, several functions are available to retrieve field values from individual records. `_fld_strval` returns any type of field formatted as a string. The `*format` parameter is a string, one of `"N"`, `"A"`, `"V"` or `"D"`, meaning the field name, abbreviation, value or description is returned in the last argument. If the format argument is `"D"`, then the MTS fdub for the dataset's dictionary file must be supplied; otherwise the parameter is not used.

Function `_fld_ival` returns the integer value (unscaled) for an integer field. If the field type is external, the value returned is the (internal) MTS line number in the data file.

Function `_fld_dval` returns a double-precision value, i.e., an integer value with the scaling function and scale factor applied, cast to double.

Function `_fld_cval` returns the value of a character field. The last argument is filled and returned as the function value.

Category Functions

A lookup can also be performed to obtain category information:

```
  CATEGORY _cat_lookup(FIELD fld_id, char *cat_name) ;
```

Function `_cat_lookup` takes a field id and a category name, and returns a category id or zero for failure. Once this lookup has been performed, several functions are available:

```
  char *_cat_name(CATEGORY cat_id, char *catname) ;
  char *_cat_abbr(CATEGORY cat_id, char *abbr) ;
  char *_cat_desc(CATEGORY cat_id, int mts_fdub, char *desc_text) ;
  INTEGER _dfalt_cat(CATEGORY cat_id) ;
  INTEGER _cat_ival(CATEGORY cat_id, FIELD fld_id) ;
  char *_cat_cval(CATEGORY cat_id, char *cat_cval, FIELD fld_id) ;
  CATEGORY _frst_cat(FIELD fld_id) ;
  CATEGORY _next_cat(CATEGORY cat_id) ;
  CATEGORY _cat_cidval(FIELD fld, char *value) ;
  CATEGORY _cat_iidval(FIELD fld, INTEGER value) ;
```

Functions `_cat_name` and `_cat_abbr` return the name and abbreviation for the category indicated in the identifier. The function value is the second argument, which is set to `_EOS` for failure. The character arguments to `_cat_name` and `_cat_abbr` should be of size `CATNAME_SIZE` and `CATABBR_SIZE`.

The function `_cat_desc` returns the description for the category. The MTS fdub for the dataset's dictionary file must be provided in the second argument. The function value is the third argument, which is `_EOS` for failure. The function `_dfalt_cat` returns 1 if the category is the default category for a field, else 0.

The functions `_cat_ival` and `_cat_cval` return the integer and character contents of a category, respectively. Function `_cat_ival` should be used only with type SC or UC (signed and unsigned integer with categories) fields; `_cat_cval` should be used only with type CC fields. The latter returns its character argument as the function value.

The functions `_frst_cat` and `_next_cat` are used to traverse the categories for a field. Function `_frst_cat` takes a field identifier and returns the identifier to the first category or zero, for no categories; `_next_cat` takes a category identifier and returns the next category identifier, or zero, for no more categories.

Function `_cat_iidval` takes a category value of the type returned by `_cat_ival` (and a field id) and returns the id of the first category with that value.

Function `_cat_cidval` takes a category value of the type returned by `_cat_cval` (and a field id) and returns the id of the first category with that value.

Front End Processor

A Front End Processor is a special MPF program that stands between Micro and the user, handling all interaction with the user, and calling Micro to execute commands. Through the MPF, programmers can do all they need to write an FEP. An FEP is declared as such in the MPF source file:

```
fep driver cfep(void)
...
}
```

That is, '`fep`' simply replaces '`micro`'. There may be only one FEP in a library. An FEP may also be declared `int`, by inserting that word between '`fep`' and the Micro name of the FEP, '`driver`' in this example. The use of an FEP return value is explained in the next section, on Exceptional Conditions.

When Micro is $Run, the master userid, by default the user's userid if not specified otherwise in the directory file, is checked for the file MPFLIB. If the file exists the MPF library it contains is activated, and if the library contains an FEP, that routine assumes control. The Micro command prompt never appears.

The FEP may use Micro command lines, `build` and `execute` in the source code to execute Micro commands and MPF commands. An FEP will probably read input from the terminal and invoke the Micro command interpreter to process it, using the `prompt` macro.

The MPF `stop` macro, if used from anywhere while an FEP is active, returns control to MTS.

As noted above, MPF commands are dynamically loaded as necessary, and by default unloaded when finished. This behavior can be changed to keep commands loaded by setting the global `int _unloadsw` to 0. It is also possible to unload accumulated MPF code by using the macro `unload`, which calls a function that unloads everything down to but not including the FEP. This should be called only from the FEP itself. If called from a module which is loaded after the FEP (or from an MPF command called from the Micro command prompt) it unloads code to which it attempts to return, with predictable results.

Exceptional Conditions

Exceptional conditions can arise from several sources when using the Micro Programming Facility. Attention and program interrupts can occur in Micro or in MPF code. Internally, these interrupts are handled separately, and it is useful to distinguish them when debugging. Execution of Micro and MPF commands, through `execute` or `prompt`, can also result in anomalies distinct from interrupts, including MTS system problems.

All exceptional conditions—attention and program interrupts, and Micro, MPF and system anomalies—cause non-zero values in `_micrortn`, the global return code variable. The macro `execute` thus performs as a function, even though MPF commands are `void` C functions. (An MPF command can also be declared as an `int` function and return its own value in `_micrortn` as explained below.) Zero is the successful return value; non-zero values correspond to the following conditions.

Return Codes in MPF Execution

Term	Value	Condition
`MPF_SUCCESS`	0	Successful Micro or MPF execution.
`MICRO_SYNTAX_ERROR`	65	A syntax error occurred in a Micro command.
`MICRO_ATTN`	66	An attention interrupt occurred in a Micro command (or a user canceled the command when prompted for replacement of an invalid dataset name, etc.)
`MICRO_USER_ERROR`	67	An inconsistency occurred in a Micro command, such as a Combine command where the datasets did not have identical structure.
`MICRO_SYSTEM_ERROR`	68	An MTS system error occurred during a Micro command, typically a GETSPACE failure
`MICRO_PGNT`	69	A program interrupt occurred in a Micro command.
`MICRO_STOP`	70	A Micro Stop command was issued (in response to a `prompt` macro, for instance).
`MPF_SYNTAX_ERROR`	71	A parameter was misspecified in invoking an MPF command.
`MPF_ATTN`	72	An attention interrupt occured in MPF code (or a user didn't reply to a prompt for a required MPF parameter).
`MPF_SYSTEM_ERROR`	73	A system error occured in the MPF, such as failure to load an MPF command.
`MPF_PGNT`	74	A program interrupt occurred in MPF code.

A C function can be designated as a *handler* in MPF code for dealing with these conditions; the word '`handler`' goes in place of '`micro`' or '`fep`'. Unlike `fep` and `micro` functions, there is no MPF name for a handler, just a C name and a single `int` argument, for the type of exception, like a C signal handler. An MPF handler, however, is generated as an `int` function by the MPF preprocessor.

The MPF handler is called after each Micro command generated by an MPF command is executed. It is called if, during execution of an MPF command, an exceptional condition occurs. It is called after each MPF command is executed, even if an exceptional condition occurred, unless the handler jumped to a different location during the processing of the exception, as explained below.

There may be only one handler in an MPF library, though if multiple libraries are active, the handler from the library containing the MPF command is called for that command.

At each handler call, the `int` argument is the value resulting from execution of the Micro or

MPF command, from the table above, 0 for success, or some non-zero value for an exceptional condition. A handler usually consists of a large C `switch` statement on the argument, with cases for the exceptional conditions. The case values should include certain C signals, for which the handler is registered through the C library `signal` function:

C Signals Received by an MPF Handler

Term	Value	Condition
`SIGINT`	2	attention interrupt
`SIGILL`	4	operation, privileged operation, execute, or specification exception
`SIGABRT`	6	abnormal termination
`SIGFPE`	8	data exception, integer or decimal divide by zero, floating exponent overflow, floating point divide by zero
`SIGBUS`	10	protection exception
`SIGSEGV`	11	addressing exception

Processing an exceptional condition in C often requires jumping with the `longjmp` function to a location previously designated with the `setjmp` function. Sometimes more than one jump point at a time is active. The MPF provides C macros for setting jump points and jumping to them.

```
/* macros for jumps */
#define setjump(sig, env) \
  { sig = setjmp(env) ; if (! sig) _mpf_jump(&(env)) ; }
#define longjump(sig)  _mpf_long(sig) ;
```

A jump point is set by `setjump`, which takes an `int` flag `sig`, and a type `jmp_buf` environment variable, `env`. When the jump point is set during normal execution (when the return from `setjmp` is 0) the jump point is registered with the MPF through `_mpf_jump`.

The most recently registered jump point is reached via `longjump`, which takes an `int` argument `sig`, usually the `sig` value passed to the handler. The base of the `env` stack is initialized to a value that, if jumped to, returns to the Micro command prompt, for an MPF command, or to MTS, for an FEP. (This is the location jumped to by the `stop` macro.) So no harm is done if more `longjumps` are used than `setjumps`.

Calls to MPF commands using `build...execute` cause stacking of the command line and the arguments to the MPF command, and activating and deactivating multiple handlers, internally and with C, if present. If jumping takes place across MPF command calls, use of `setjump/longjump` is *required*, not optional, for this bookkeeping to take place.

The `sig` value that is used in `longjump` can be a user-defined one, obviously. A handler that simply returns its `sig` argument for a particular case, without jumping, assigns that value to `_micrortn` where it can be tested, after `execute`. This return code can also be propagated down the MPF call stack, since the handler is called as each MPF command is finished, and an MPF command can be declared as a function and return a value. This is done by inserting '`int`' between '`micro`' and the MPF command name:

`micro int` *MPFname Cname*`(...)`

If present, a handler is called with the function value before control returns to the calling point. An FEP can also be declared `int` and return a value, if desired; such a value is seen by the handler before control returns to Micro and MTS.

The MPF `handler` and `setjump/longjump` facilities provide complete control over exceptional conditions and return codes. There is no reason to use the corresponding C facilities, or to call a `handler` directly. There may be a need to use `raise` to re-signal an attention interrupt, for example, though the propagation of the return code through the MPF function value provides a partial alternative. There are also positive reasons to let the MPF facilities do the remaining work.

As noted above, exceptional conditions can arise from Micro and from MPF code. Internally, interrupts in these areas are trapped separately, and it is useful to distinguish them when debugging. Also, there are several possible anomalous conditions distinct from interrupts. Only an MPF `handler` can process all the exceptional conditions (as well as success) in a single place.

The `build...execute` facility and the `setjump...longjump` macros maintain a call stack for MPF commands, activate and deactive multiple handlers where they are present, and keep data useful for command tracing. The Micro command line is stored in the global `char * _microcmd`. This is declared `extern` in ILIR:MPF.H, which is `#included` in the MPF target code. The MPF command line is stored in the global `char * _mpfcmd`, which is also accessible to MPF code. Since a handler is called after each execution of a Micro command, and after each MPF command, a log of the session can be kept.

The Micro command trace facility records all Micro commands, even those from MPF commands, but does not record MPF commands themselves. The variable `_mpfcmd` is the only access to the MPF command line. Even if a log is not kept, the most recent Micro and MPF commands, available to the handler, are usually useful in debugging.

The file ILIR:PGNT.H, which contains various character variables, C structures and macros, may be `#included` in the handler to help analyse program interrupts. This file is not included automatically in the target code.

```
  char *int_text[] = {"Operation", "Privileged operation",
    "Execute", "Protection", "Addressing", "Specification",
    "Data", "Fixed-point overflow", "Fixed-point divide",
    "Decimal overflow", "Decimal divide", "Exponent overflow",
    "Significance", "Floating point divide"} ;
  struct psw_recd {
    unsigned int ch_mask : 7 ;
    unsigned int e_mask : 1 ;
    unsigned int pkey : 4 ;
    unsigned int cmwp : 4 ;
    unsigned int interrupt_code : 16 ;
    unsigned int ilc : 2 ;
    unsigned int cc : 2 ;
    unsigned int prog_mask : 4 ;
    unsigned int instr_addr : 24 ;
  } ;
  struct save_area_recd {
    struct psw_recd psw ;
    unsigned int grs[16] ;
  } ;
  struct guinfo_words {
    void *pgnt_handler ;
    struct save_area_recd *save_area ;
  } guinfo_rtn ;
/* macros for pgnt handling; reference above data structures */
#define pgnt_info guinfo("PGNTTRP ", &guinfo_rtn) ;
#define pgnt_message \
  fprintf(stderr, "%s exception at MPF address %X.\n", \
    int_text[guinfo_rtn.save_area->psw.interrupt_code-1], \
    guinfo_rtn.save_area->psw.instr_addr) ;
```

The variable `int_types` is a list of interrupt types, ordered according to possible values in the `interrupt_code` of the program status word. The three structures supply the storage that the MTS routine `guinfo` fills. The macros generate the call to `guinfo` and print a message.

If no handler exists, and a program interrupt occurs in MPF code, a message such as

```
Protection exception at MPF address 8762C2.
```

is printed. Control is then returned to the Micro command prompt, or to MTS, for an FEP. The same occurs if an attention interrupt happens in MPF code or a program or attention interrupt in Micro, if no handler exists. Any other anomalous events are reflected in the MPF return code, `_micrortn`.

During MPF execution, the user handler is active only when user MPF code is executing. There is some "dead time" during which the MPF execution processor, and Micro, which have their own exception handlers, are active.

When it is first called, the MPF exec processor signals its own handlers to C; just before it calls an MPF command, it signals the user handler. A user handler will receive the signal `MPF_ATTN` only if an attention interrupt occurs in this interval. One possibility is in response to a prompt for a parameter to an MPF command.

Micro and its handlers are active when `execute` or `prompt` are used; an MPF command constructed with `build` also enters Micro briefly before the MPF exec processor assumes control. If an attention occurs in the Micro portion, a user handler is called with `MICRO_ATTN`.

The same holds for program interrupts, in the MPF exec processor and in Micro. If the MPF itself receives the program interrupt, a handler is called with `MPF_PGNT`. If Micro receives the program interrupt, the handler is called with `MICRO_PGNT`.

The overhead of the MPF execution processor sounds worse than it is. In practice, the MPF takes less time to process the execution of a Micro or MPF command than is required to read a line from a command file, with the buffering, swapping and other complications of I/O in MTS.

The following MPF libraries illustrates some of these concepts. In the first library, the MPF command `test` ultimately calls three other MPF commands and enters an infinite loop which can be exited only by an attention. All the MPF commands are declared `int`, so they can return values.

```
micro int test ctest
  printf("In TEST\n") ;
  build pass1 1  Mississippi
  execute
  printf("Leave TEST\n") ;
  return 0 ;
}
handler hdlr(int sig)
  printf("HDLR, mpfcmd=\"%s\"\n", _mpfcmd) ;
  switch(sig) {
  case MPF_ATTN: case SIGINT:
    printf("sig=%i, jump on env\n", sig) ;
    longjump(sig)
  }
  return sig ;
}
micro looper cloop
  while (1) printf("Looper!\n") ;
}
micro int pass1 p1(int ivar, char *river)
  int jmpflg = 0 ;
  jmp_buf env ;
  printf("PASS1, flag=%i  river=%s\n", ivar, river) ;
  setjump(jmpflg, env)
  if (! jmpflg) {
    build pass2 2 Columbia
    execute
  }
  else
    printf("Jump to PASS1\n") ;
  return 0 ;
}
micro int pass2 p2(int ivar, char *river)
  int jmpflg = 0 ;
  jmp_buf env ;
  printf("PASS2, flag=%i  river=%s\n", ivar, river) ;
  setjump(jmpflg, env)
  if (! jmpflg) {
    build pass3 3 Hudson
    execute
  }
  else
    printf("Jump to PASS2\n") ;
  return jmpflg ;
}
micro int pass3 p3(int ivar, char *river)
  int jmpflg = 0 ;
  jmp_buf env ;
  printf("PASS3, flag=%i  river=%s\n", ivar, river) ;
  setjump(jmpflg, env)
  if (! jmpflg) {
    build looper
    execute
  }
  else
    printf("Jump to PASS3\n") ;
  return jmpflg ;
}
```

Invoking **test** at the Micro prompt, and hitting <attn>, produces the following output:

```
In TEST
PASS1, flag=1  river=Mississippi
PASS2, flag=2  river=Columbia
PASS3, flag=3  river=Hudson
Looper!
Looper!
Looper!HDLR, mpfcmd="looper"
sig=2, jump on env
Jump to PASS3
HDLR, mpfcmd="pass3 3 Hudson"
sig=2, jump on env
Jump to PASS2
HDLR, mpfcmd="pass2 2 Columbia"
sig=2, jump on env
Jump to PASS1
HDLR, mpfcmd="pass1 1  Mississippi"
Leave TEST
HDLR, mpfcmd="test"
```

After the loop is entered, hitting <attn> delivers control to the handler, which prints **_mpfcmd** and does a **longjump**, to **pass3**, the most recent jump point set with **setjump**. **pass3** returns the variable **jmpflg**, and since a handler is present, it is called as **pass3** is exited, with **jmpflg** as the argument. This causes another jump, from within the handler, to **pass2**, and so on, until **pass1** returns 0, from which the handler just returns. (An extra jump at this point, one more than set, would have returned control to the Micro prompt, without further MPF processing, as explained above.) Each time through **hdlr**, the contents of **_mpfcmd** are printed.

The following library has an FEP with a command prompt loop.

```
fep mfep cfep
  jmp_buf env ;
  int flag ;

  while (1) {
    setjump(flag, env)
    if (! flag)
      prompt
  }
}
handler hdlr(int sig)
  switch(sig) {
  case  MICRO_ATTN:
    longjump(sig)
    break ;
  case MICRO_STOP:
    stop
    break ;
  }
  return sig ;
}
```

In the FEP, **setjump** is used inside the loop. If an attention interrupt occurs at the prompt (in Micro), the handler is called with argument **MICRO_ATTN**. It jumps back into the loop with that value, which merely causes another iteration: **setjump** and **prompt**.

Like Micro, the prompting loop can be exited only with a Stop command. The handler is called with `MICRO_STOP`, and uses the `stop` macro to leave Micro for MTS.

Running the Preprocessor

The Micro preprocessor is invoked on MTS by:

`$Run ilir:mpf input=`*sourcefile* [`par=`*microlibfile objectfile*]

Target code is written to file –CTARGET, which is first emptied. If a *mpflibfile* is given in the `PAR` field, data on the MPF commands in the source file is written to that file. Micro uses that information to set up parameters and call the MPF commands. It is activated automatically when Micro is $Run, if named `mpflib` and present on the master userid. Otherwise it can be set from Micro, as shown above.

If two file names are given, the second file is assumed to be the file containing the compiled target code, i.e., the object code produced by:

`$Run *C87 input=-ctarget object=`*objectfile*

This file name is written to the end of the last line in the Micro library file. If the object file name is not present when that file is read by Micro, it is prompted for. The object file can be added manually to the Micro library file with an MTS editor command:

`$edit` *mpflibfile* append`@nv *l ;`*objectfile*`;`

An MTS macro to run the preprocessor, compile the target code with *C87, and add the object code to a library is in the file ILIR:MPF.MAC. It may be activated by

`$Source ILIR:MPF.Mac`

The file contains:

```
>macro MPF Source MPFLib="MPFLib" ObjLib="MPF.Lib" @etc @noprompt
>*   This macro compiles a program written in MPF.
>*   If the source file name is in the form "xxx.MPF", you can leave
>*   off the ".MPF" in {Source}.
>*
  if Source = NULL or Source(1|1) = "?"
    write " Usage:   MPF  source  [MPFLib=mpflib]  [ObjLib=objlib]"
    write "         (defaults:    MPFLIB     MPF.LIB)"
    exit
  endif
>*
  set var  Source = uppercase(Source)
  if  file(Source) doesnt exist
    if  find_string(Source, ".MPF") = 0,  set var  Source ||= ".MPF"
  endif
  if  file(Source) doesnt exist
    write " The source file, {Source}, doesn't exist."
    exit
  endif
>*
  define  Target = "-ctarget"
  define  Object = "-obj"
  define  CCid = SIGNONID || ":"
>*
  if  find_string(MPFLib, ":") = 0,  set var  MPFLib = CCid || MPFLib
  if  find_string(ObjLib, ":") = 0,  set var  ObjLib = CCid || ObjLib
>*
  emit  "$Truncate {Source}"
  if  file(Target) exists,  emit  "$Empty {Target} ok"
  emit  "$Run ILIR:MPF input={Source} par={MPFLib} {ObjLib}"
  if  runrc > 0,  exit
>*
  if  file(Object) exists,  emit  "$Empty {Object} ok"
  emit  "$Run *C87 input={Target} object={Object} par=i=ILIR {etc}"
  if  runrc > 0,  exit
>*
  if  file(ObjLib) doesnt exist
    write " Creating the object library, {ObjLib}."
    emit  "$Create {ObjLib}"
  endif
  emit  "$Run *ObjUtil input={Object} 0={ObjLib} par=SymSave, Lib"
>*
>endmacro
```

The `Source` argument, the file containing the MPF source code, is required; if the file name ends in '`.MPF`', that part can be omitted. The `MPFLib` parameter, defaults to that name if omitted, and `ObjLib` defaults to MPF.LIB. If alternate names are desired, they must be supplied in keyword fashion when the macro is invoked (by the name `mpf`) at the MTS command prompt:

```
mpf kabc:report mpflib=reportlib objlib=report.lib
```

Target Code

C header files <string.h>, <stdlib.h>, <stdio.h>, <signal.h>, <ctype.h>, <setjmp.h>, and <mts.h> are included in the target code, as is ILIR:MPF.H, which includes ILIR:MPF.DEF and ILIR:MPFLIB.H. Declarations, etc., in the latter files are used by the target code.

The following examples show MPF code and the code produced by the MPF preprocessor. The C macros shown here, which govern MPF execution, have been expanded.

```
/* macros for MPF execution */
#define cancel  _microcmd = _EOS ;
#define execute _micrortn = _exec_micro_cmd(_microcmd) ;
#define prompt  { cancel execute }
#define stop    _mpf_exit() ;
```

Here is a declaration of an MPF command:

```
  micro report rgen
    printf("Printing report\n") ;
  }
```

...and the target code:

```
    #line 1
  void rgen(void) {
    printf("Printing report\n") ;
  }
```

The following MPF code

```
  micro report rgen(int ncopies=1, char *title, -
    char *sponsor="XYZ Institute", float version=)
    printf("Printing report: copies=%d  title=%s  sponsor=%s  version=%f\n",
      ncopies, title, sponsor, (double)version) ;
  }
```

produces the following target code:

```
    #line 1
  void rgen(char *argtxt[]) {
    #line 1
    int ncopies = atoi(argtxt[0]) ;
    #line 1
    char *title = argtxt[1] ;
    #line 1
    char *sponsor = argtxt[2] ;
    #line 1
    float version = atof(argtxt[3]) ;
    #line 3
    printf("Printing report: copies=%d  title=%s  sponsor=%s  version=%f\n",
      ncopies, title, sponsor, (double)version) ;
  }
```

Here is an example of MPF code that generates and executes a Micro command.

```
  sort {ds_name} by -
    {field_name}
  execute
```

The first two lines are processed as one, and generate one line of target code, followed by the target code for the third line:

```
#line ...
sprintf(_microcmd, "%s%s%s%s", "sort ", ds_name, " by ", field_name) ;
#line ...
_micrortn = _exec_micro_cmd(_microcmd) ;
```

The `sprintf` function is the standard C library formatting function from <stdio.h>. Command text is formatted into `_microcmd`, a global variable maintained by Micro, assuming `ds_name` and `field_name` evaluate to `char *`.

The function `_exec_micro_cmd` passes the string to Micro for execution, returning the `MPF...` values described above.

The variable `_micrortn` is a global int whose value can be tested after each Micro command.

Following is an example of conditional command construction, using the colon notation.

```
find in Fin-aid where
if (! strcmp(field_name, "source-of-funds"))
  : source-of-funds = industry or UNIV or 3
else
  : Amt is between {lower} and {upper}
execute
```

The preprocessor generates:

```
#line ...
sprintf(_microcmd, "%s", "find in Fin-aid where") ;
if (! strcmp(field_name, "source-of-funds"))
#line ...
sprintf(_microcmd+strlen(_microcmd), "%s", " source-of-funds = industry or UNIV") ;
else
#line ...
sprintf(_microcmd_+strlen(_microcmd), "%s%s%s%s", " Amount is  between ",
#line ...
lower, " and ", upper) ;
#line ...
_micrortn = _exec_micro_cmd(_microcmd) ;
```

The first line from the source code is formatted into the command buffer in the target code. The '`if (...)`' is standard C, passed through. If the condition is satisfied, the text after the first colon is added to the command; otherwise the else code is entered, and the segments of text following the second colon are added.

The following illustrates use of `build` and `cancel`:

```
build report title="Current Research Projects" 5
...
cancel
```

Note that, while '`report`' is not a Micro command, it can placed in the command string by using the '`build`' notation. The cancel preprocessor directive simply assigns the C end-of-string to the Micro command The target code is:

```
#line ...
sprintf(_microcmd, "report title=\"Current Research Projects\" 5") ;
...
#line ...
*_microcmd = _EOS ;
```

Not every source line is prefaced by a `#line` directive, only those generated by the preprocessor. Line numbers increase by 1 after each `#line`, and line numbers in C compiler error messages will be correct if there are no fractionally-numbered lines in the source file, and if the lines in the source file start at 1.

Appendix: Deprecated Features

ILIR:Micro Parameters

This discussion does *not* apply to the Micro Programming Facility, where the same issues are handled automatically.

The following parameters may appear in the **par=** field on the $Run ILIR:Micro command.

`esd={1|2|3|4}  fep=`*objectfile* `feppar=`*string*

Micro allows user programs to be loaded and executed via the various Call commands (Call, Micall, and Extcall) and via a FEP. A mechanism was developed to eliminate collisions between Micro's and users' external symbols (names of subroutines, functions, labeled commons, etc.) However, there are some routines resident in Micro that user programs reference. These Micro routines must be available to user programs while, at the same time, all of the other Micro external symbols must be "hidden" to avoid collisions.

There are two parts to the mechanism. First, the names of all Micro external symbols are changed to something unlikely to be generated by any compiler, thus hiding them from user programs. Second, External Symbol Dictionaries (ESDs, not to be confused with Micro dictionary files) are used to allow user programs to access the resident Micro routines.

Whenever new versions of Micro are installed, the names of all the Micro external symbols are changed to include a "prefix" character, a '!'. For example, PROMPT becomes !PROMPT.

The names of the available Micro routines are listed in the files:

ILIR:ESD/1	safe routines
ILIR:ESD/2	dangerous routines
ILIR:ESD/3	very dangerous routines
ILIR:ESD/4	terribly dangerous routines (don't use)

These files are used to build the ESDs in Micro's loading routine. They represent different "levels" of Micro routines: the higher numbered levels contain the more dangerous routines ("dangerous" in that they modify Micro's internal structures).

If you make no changes at all in your application, the routines in ILIR:ESD/1 are available. These are relatively safe routines which are referenced widely in user programs. You are encouraged to NOT use the routines at levels higher than 1. But, if you must, they can be made available to Call command routines by specifying

`CALL` *subr* `... ESD=`*n*`...`

where n can be 0, 1 (the default), 2, or 3. ESD=0 means that none of Micro's routines are available. For FEPs, you can specify

`$Run ILIR:Micro ...  par=FEP=`*objfile* `ESD=`*n* where *n* can be 1 (the default), 2, or 3. The ESDs are cumulative so that, for example, specifying ESD=2 makes available the routines at levels 1 and 2. The ESD specifications for a FEP and for any Call commands are independent (e.g., ESD=2 for a FEP does not force ESD=2 to be the default for the Call commands).

Currently, the names of the routines listed in these files should be considered reserved—you should *not* have subroutines, etc. with these names in the same applications where you reference the Micro routines.

As noted, specifying fep=*objfile* on the $Run command loads an FEP from that object file. Specifying feppar=*string* passes the string to the FEP routine, which has been declared

```
SUBROUTINE  MICFEP(FEPPAR, PARLEN, SWIT)
  INTEGER*4  PARLEN, SWIT
  LOGICAL*1  FEPPAR(PARLEN)
```

As all the facilities discussed in this section have been replaced by the Micro Programming Facility, this information is provided only for those still using them. No new FEPs of this sort should be written; the MPF should be used.

CALL Micro Command

Purpose: To call a subroutine from the Micro command prompt.

Synonyms: None.

Effect on the Result: None.

Usage

```
CAll subr_name [USING  dataset]
               [LIB =  object_library]
               [KEEP = {YES|NO}]
               [PAR =  <par> [, <par>] ...]
```

where

<par> can be any of:

string
number

Examples

```
Call Rpt  using Result  lib = Call/Lib  keep = yes   -
      par = "University Financial Aid", 1986
```

This calls a subroutine named Rpt, contained in the object library CallLib. Eight parameters are passed: the first six are the dataset parameters from Result, and the last two are from the 'par' phrase. '`University Financial Aid`' is passed to Rpt as a character string, probably a report title; '`1986`' is passed as an integer. After the Rpt subroutine returns to Micro, it remains loaded.

```
Call Time  par=6,1
```

This calls the MTS system routine Time. Since no dataset is used, only the parameters from the 'par' phrase are passed. Both parameters are passed to Time as integers. This call prints the current time and date on the terminal in a format such as:

```
08:14:22
MAY 11, 1992
```

Settings

None.

Modifiers

None.

Description

The Call command allows Fortran subroutines to be called from the Micro command prompt. Note: The Micro Programming Facility allows C functions to be called which can generate and execute Micro commands and access datasets through the Micro Interface Library of C functions. It is the preferred way to program an individual Micro command.

For those who still have a need for the Call command, the `USING` phrase of the syntax indicates the dataset whose parameters are passed to the subroutine (the calling sequence is discussed below). `LIB=` indicates the file containing the object code of the routine. `KEEP=` tells whether the routine is to be unloaded after the call. The default is to keep the routine loaded; the routine is unloaded only if `KEEP` is assigned `NO`. `PAR=` is used to pass extra (character or numeric) parameters to the routine, after the required calling sequence.

The Call command generates the Fortran (IBM S-type) call:

`CALL` *subr_name* `(NWORDS, IFINFO, NCATS, ICINFO, NUMREC, IDATA [, <par> ] ...)`

`NWORDS`	`NWORDS` is a fullword integer containing the width in fullwords of each data record in `IDATA` passed to the subroutine.
`IFINFO`	`IFINFO(6,NWORDS)` is a fullword integer matrix. That is, there are six words of information in `IFINFO` for each data record in `IDATA`. The first four words—`IFINFO(1,I)...(4,I)`—contain the 16-character field name. The fifth word—`IFINFO(5,I)`—contains the number of categories in the associated field. The sixth word—`IFINFO(6,I)`—contains an index into the `ICINFO` matrix which contains category information for the associated field. When the fifth and sixth words are zero, the field has no categories.
`NCATS`	`NCATS` is a fullword integer containing the total number of categories and external data file names, the sum of `IFINFO(5,I)`, where 1 <= `I` <= `NWORDS`.

`ICINFO`	`ICINFO(4,NCATS)` is a fullword integer matrix containing category and external data file information. The first three words of each category `ICINFO(1,I)...(3,I)` contain the category name. The twelfth character is always blank because category names are less than or equal to eleven characters in length. The fourth word `ICINFO(4,I)` is the value of the associated category. The only exception is for external data fields, for which `ICINFO(1,I)...(4,I)` contain the file name associated with the external data field. The file name is in the format *userid:filename*.
`NUMREC`	`NUMREC` is a fullword integer representing the number of records from the dataset passed to the subroutine, i.e., the number of records in the specified dataset.
`IDATA`	`IDATA(NWORDS,NUMREC)` is a fullword integer array of data for the dataset specified in `USING`. Integer values are automatically expanded to fullwords in `IDATA`. All character fields are expanded by padding with blanks until their byte length is a multiple integral of fullwords.
<par>	Each <par> is an argument passed from the Call command. The arguments are converted as fullword integers, if they are numeric, doubleword floating point numbers (`REAL*8`), if they if they are numeric and contain a decimal point, or are passed as strings of up to 24 characters.

EXTCALL Micro Command

Purpose: To call a subroutine from within Micro using the extended calling sequence.

Synonyms: None.

Effect on the Result: None.

Usage

```
EXTCALL subr_name [USING dataset]
                  [LIB = object_library]
                  [KEEP = {YES|NO}]
                  [PAR = <par> [, <par>] ...]
```

where:

<par> can be any of:

string
number

Examples

```
ExtCall RptExt  using Result  lib=ExtCall/Lib  -
        par = "University Financial Aid", 1986
```

This calls a subroutine named RptExt, contained in the object library ExtCall/Lib. Nine parameters are passed: the first seven are the dataset parameters from Result, and the last two are from the 'par' phrase. 'University Financial Aid' is passed to Rpt as a character string, probably a report title; '1986' is passed as an integer. After the RptExt subroutine returns to Micro, it is unloaded.

```
ExtCall Time  par=6,1
```

This calls the MTS system routine Time. Since no dataset is used, only the parameters from the 'par' phrase are passed. Both parameters are passed to Time as integers. This call prints the current time and date on the terminal in a format such as:

```
08:14:22
MAY 11, 1992
```

Settings

None.

Modifiers

None.

Description

The Extcall command allows Fortran subroutines to be called with a more elaborate calling sequence than provided by the Call command. Note: The Micro Programming Facility allows C functions to be called which can generate and execute Micro commands and access datasets through the Micro Interface Library of C functions. It is the preferred way to program an individual Micro command.

For those who still have a need for the Extcall command, the `USING` phrase of the syntax indicates the dataset whose parameters are passed to the subroutine (the calling sequence is discussed below). `LIB=` indicates the file containing the object code of the routine. `KEEP=` tells whether the routine is to be unloaded after the call. The default is to keep the routine loaded; the routine is unloaded only if `KEEP` is assigned `NO`. `PAR=` is used to pass extra (character or numeric) parameters to the routine, after the required calling sequence.

The calling sequence generated is:

```
CALL subr_name(NWORDS, IFINFO, NCATS, IFINCO, NRECS,
        IDATA, DICFIL [, <pars> ] ...)
```

`NWORDS`	`NWORDS` is a fullword integer containing the width in fullwords of each data record in `IDATA` passed to the subroutine.

`IFINFO`	`IFINFO(12,NWORDS)` is a fullword integer matrix. That is, there are twelve words of information in IFINFO for each data record in `IDATA`. `IFINFO(1,I)...(4,I)` contain the 16-character field name. `IFINFO(5,I)` contains the number of categories in the associated field. `IFINFO(6,I)` contains an index into the `ICINFO` matrix which contains category information for the associated field. When the fifth and sixth words are zero, the field has no categories. `IFINFO(7,I)` is a four character field abbreviation. `IFINFO(8,I)` is the field type, 1=U, 2=C, 3=UC, 4=CC, 5=S, 6=E, 7=SC. `IFINFO(9,I)` is the field length. `IFINFO(10,I)` is a notepointer for the field description in `DICFIL`. `IFINFO(11,I)` is the scaling function, 0=none, 1=+, 2=-, 3=*, 4=/. `IFINFO(12,I)` is the scaling factor.
`NCATS`	`NCATS` is a fullword integer containing the total number of categories and external data file names, the sum of `IFINFO(5,I)`, where 1 <= `I` <= `NWORDS`.
`ICINFO`	`ICINFO(6,NCATS)` is a fullword integer matrix containing category and external data file information. `ICINFO(1,I)...(3,I)` contain the 11-character category name. `ICINFO(4,I)` is the value of the associated category. `IFINFO(5,I)` is a four character category abbreviation. `IFINFO(6,I)` is a notepointer to the category description in `DICFIL`. For a type E field, there is one `ICINFO` row. `ICINFO(1,I)...(5,I)` is a 20 character external file name in the format *userid:filename*.
`NUMREC`	`NUMREC` is a fullword integer representing the number of records from the data set passed to the subroutine, i.e., the number of records in the specified data set.

IDATA	**IDATA(NWORDS,NUMREC)** is a fullword integer array of data for the dataset specified in **USING**. Integer values are automatically expanded to fullwords in **IDATA**. All character fields are expanded by padding with blanks until their byte length is a multiple integral of fullwords.
DICFIL	**DICFIL** is the name of the dictionary file for the dataset, in *userid:filename* form, terminated by a blank. The MTS FDUB for the file can be obtained and dictionary information read by the subroutines discussed below.
<par>	Each <par> is an argument passed from the ExtCall command. The arguments are converted as fullword integers, if they are numeric, doubleword floating point numbers (**REAL*8**), if they if they are numeric and contain a decimal point, or are passed as strings of up to 24 characters.

The following routines can be used to obtain field and category descriptions from the dictionary file. These routines are resident in the Micro system (see ILIR:ESD/NEWS for further information).

STATUS = GTFDUB(DICFIL, FDUB)	**GTFDUB** gets an MTS File/Device Usage Block for the dictionary file, which is necessary to read the file. It is passed **DICFIL**, the seventh parameter in the Extcall sequence, and returns **FDUB**, an **I*4** Fortran value, and 0 for an (integer) function value if the operation is successful, else non-zero.
STATUS = RDDESC(FDUB, NTPTR, BUFFER, LENGTH)	**RDDESC** is passed an MTS FDUB as obtained with **GTFDUB**, and a description note pointer, from **IFINFO** or **ICINFO**. It returns **BUFFER**, holding the field or category description, and **LENGTH**, an **I*4** value. The integer function return is 0 for success, else non-zero.
STATUS = FRFDUB(FDUB)	**FRFDUB** release the MTS FDUB. It is passed the FDUB, and always returns 0.

MICALL Micro Command

Purpose: To call a subroutine from within Micro using the Micro Interface calling sequence.

Synonyms: None.

Effect on the Result: None.

Usage

```
MICALL subr_name [USING dataset]
                 [LIB = object_library]
                 [KEEP = {YES|NO}]
                 [PAR =  <par> [, <par>] ...]
```

where:

<par> can be any of:

string
number

Examples

```
Micall Rpt  using Result  lib = Micall/Lib  keep = yes   -
       par = "University Financial Aid", 1986
```

This calls a subroutine named Rpt, contained in the object library Micall/Lib. Five parameters are passed: the first three are the dataset parameters from Result, and the last two are from the 'par' phrase. '`University Financial Aid`' is passed to Rpt as a character string, probably a report title; '`1986`' is passed as an integer. After the Rpt subroutine returns to Micro, it remains loaded.

```
Micall Time  par = 6, 1
```

This calls the MTS system routine Time. Since no dataset is used, only the parameters from the 'par' phrase are passed. Both parameters are passed to Time as integers. This call prints the current time and date on the terminal in a format such as:

```
08:14:22
MAY 11, 1992
```

Settings

None.

Modifiers

None.

Description

The Micall command allows Fortran subroutines to be called from the Micro command prompt with a calling sequence containing the internal data structures used by Micro. Note: The Micro Programming Facility allows C functions to be called which can generate and execute Micro commands and access datasets through the Micro Interface Library of C functions. It is the preferred way to program an individual Micro command.

For those who still need the Micall command, the `USING` phrase of the syntax indicates the dataset whose parameters are passed to the subroutine (the calling sequence is discussed below). `LIB=` indicates the file containing the object code of the routine. `KEEP=` tells whether the routine is to be unloaded after the call. The default is to keep the routine loaded; the routine is unloaded only if `KEEP` is assigned '`NO`'. `PAR=` is used to pass extra (character or numeric) parameters to the routine, after the required calling sequence.

The calling sequence generated is:

`CALL` *subr_name*`(DICFIL, FLDLST, SET` [`,` <par>] `...)`

where `DICFIL` is a 20-character dictionary file name for the dataset in use. This is passed to `GTFDUB` to obtain the MTS File/Device Usage Block for the dictionary file, to obtain descriptions and other information. See the description of the Extcall command. `FLDLST` is the internal field and category structure for the dataset. `SET` is the STDS set for the dataset. These are used by the routines described in ILIR:MICALL.W, which may be called by the Fortran subroutine to access the dataset structure and the data.

USE Micro Command

Purpose: To specify which dataset is to be used in a subsequent Read command. (The Use command is obsolete since the Read command no longer needs it)

Synonyms: None.

Effect on the Result: None.

Usage

`USe` *dataset*

Examples

```
Use Students
```

```
Read from -data
```

This creates a new Result dataset containing the fields from the Students dataset and the records from the Micro data file -data.

```
Read from -data using Students
```

This single command does same thing.

Settings

None.

Modifiers

None.

Description

The Use command is obsolete since you can specify a dataset to use directly in the Read command (see the examples).

Macro Facility

The macro facility has been replaced by the Micro Programming Facility. The MPF allows functions written in the C programming language to be called from the Micro command prompt. Such functions, called MPF Micro commands, can generate Micro commands and other MPF commands for execution, can call Micro Interface Functions to access Micro's run-time internal data structures, and can call the Micro command interpreter itself. The MPF is documented in Part 4.

The following example of a Micro macro library shows how old macros work, for those who need to know. The first line tells the macro processor how many macros are in the library, and the next lines give their names and line numbers in the file. If a line number appears more than once, that macro has more than one name, as with Count and Card in this library.

Count takes a single argument, as shown by '`@1`' after its declaration at line 100. Lines that begin with words are either macro declarations or Micro commands. Lines beginning with '`%*`' are comments. Lines beginning with just '`%`' are functional to the macro definition, like those starting at line 106, which test for the existence of the parameter, and then whether it is a question mark, going to the appropriate label if a test fails. A label ends with '`nop`', as on line 108, or is a prefix to an IF statement, as on line 112. The macro definition ends with '`%...end`'.

The Count macro prints the number of records in a dataset.

```
#list k59f:maclib.t5(1,117)

     1    3
     2    Count     100       Print count of dataset
     3    Card      100       Print count of dataset
     4    Calc      200       Calculate age or month
    98
    99
   100    Count @1
   101    %*
   102    %*  This macro prints the count (cardinality) of the given dataset
   103    %*
   104    %*      COUNT dataset
   105    %*
   106    %       if    @1 eq @null  then  goto syntax
   107    %       if    @1 ne ?  then  goto go
   108    %syntax nop
   109    message '     Usage:'.
   110    message '        COUNT dataset'.
   111    %       stop
   112    %go     if    @count @1 gt 0   then  goto notnull
   113    message '@1 is null!'.
   114    %       stop
   115    %notnull nop
   116    p in @1 count.
   117    %       end
```

The Calc macro takes four arguments. As shown by the '`Usage`' message, the parameters are the word '`IN`', a *dataset*, the word '`AGE`' or '`MONTH`' (or '`MON`'), and a birthdate field in the fourth argument. If the fourth argument is omitted, a field name Birthdate is supplied. If Age is specified in the second argument, Calc first calculates the value 1 into a new field, Age. Then it uses the functions `int`, `date` and `from_yymmdd` to calculate the age. If Month is specified, Calc uses `month` and `from_yymmdd` to determine the birth month. Various statements check the arguments, print error messages, etc.

```
#list k59f:maclib.t5(200)
    200     Calc @3 @1 @2 from @4
    201     %*
    202     %*  Given a birthdate in YYMMDD format, CALC will calculate:
    203     %*    a) The person's AGE in YY format; or
    204     %*    b) The person's birth MONTH in MM format
    205     %*
    206     %         if  @3 ne in    goto syntax
    207     %         if  @2 = age    goto age
    208     %         if  @2 = month  goto month
    209     %         if  @2 = mon    goto month
    210     %*
    211     %*  syntax error
    212     %*
    213     %syntax  nop
    214     message '      Usage:'.
    215     message '         CALC IN dataset AGE [FROM field]'.
    216     message '                     or'.
    217     message '         CALC IN dataset MONTH [FROM field]'.
    218     %         stop
    219     %*
    220     %*  calc age
    221     %*
    222     %age      nop
    223     %         if  @count @1 gt 0  goto  go1
    224     message 'warning:  @1 is null!'.
    225     %go1      nop
    226     set verify off.
    227     calculate in @1 @2 = 1.
    228     calculate in it @2 = int((date() - from_yymmdd(
    229     %         if  @4 eq @null  goto bday1
    230     @4
    231     %         goto  fin1
    232     %bday1   nop
    233     birthdate
    234     %fin1    nop
    235     )) / 365..25).
    236     %         goto  done
    237     %*
    238     %*  calc birth month
    239     %*
    240     %month   nop
    241     %         if  @count @1 gt 0  goto  go2
    242     message 'warning:  @1 is null!'.
    243     %go2     nop
    244     set verify off.
    245     calculate in @1 @2 = month(from_yymmdd(
    246     %         if  @4 eq @null  goto bday2
    247     @4
    248     %         goto  fin2
    249     %bday2   nop
    250     birthdate
    251     %fin2    nop
    252     )).
    253     %         goto  done
    254     %*
    255     %*  tell user it's done
    256     %*
    257     %done    nop
    258     release *problems*.
    259     message 'O@2 has been CALCed..'.
    260     set verify on.
    261     p in it count.
    262     %         end
```

Deprecated Settings and Modifiers

The following global settings have been made obsolete with different settings or enhancements to existing ones. They are documented for the benefit of those who are using them, but may be removed from the program in the future.

Deprecated Settings

Group	Item	Allowable values	Initially
PRINT:	WINDOW	positive integer	20
PRINT:	CAT DESCRIPTIONS CDESC	ON or OFF	OFF
PRINT:	FIELD DESCRIPTIONS FDESC	ON or OFF	OFF
PRINT:	VALUES	ON or OFF	OFF
MACRO:	ECHO	ON or OFF	OFF
MACRO:	LIBRARY LIB	*filename* or OFF	OFF
CMD:	PERIOD	ON or OFF	OFF

WINDOW	The same as **@WIDTH=*x*** command and field modifiers, except **WINDOW** applies only to text items. (default: 20)
CAT DESCRIPTIONS ON|OFF **CDESC**	Use the **CAT DESC** setting. (default: **OFF**)
DESCRIPTIONS **DESC**	Use the **CAT DESC** and **FLD DESC** settings. (default: **OFF**)
FIELD DESCRIPTIONS ON|OFF **FDESC**	Use the **FLD DESC** setting. (default: **OFF**)
VALUES	Use the **CAT VALUE** setting. (default: **OFF**)
ECHO	Sets echoing of command lines emitted by macros. Use the Micro Programming Facility. (default: **OFF**)
LIBRARY *filename*|OFF **LIB**	Activates the macro library *filename*. Use the Micro Programming Facility. (default: **OFF**)
PERIOD ON|OFF	The **PERIOD** setting controls the need for a period at the end of each Micro command. (default: **OFF**)

The following Print command modifiers have been made obsolete with different settings or enhancements to existing ones. They are documented for the benefit of those who are using them, but may be removed from the program in the future.

Deprecated Print Modifiers

Function	Modifier	Command Modifier	Field Modifier
fields, categories as names	`@NAME`	×	×
categories as descriptions	`@DEscription`	×	×
	`@CDEscription`	×	×
fields as descriptions	`@FDEscription`	×	×
text item width	`@WINDOW=`x	×	×

`@NAME`	Same as `@FLD=NAME@CAT=NAME`.
`@DESCRIPTION` `@CDEscription`	Same as `@CAT=DESC`.
`@FDEscription`	Same as `@FLD=DESC`.
`@Window=`x	Same as `@WIDTH=`x, except `@WINDOW` applies to text items only. (default: 20)

Index

Page numbers in bold—**221**—indicate the primary reference for a command or concept. Numbers in italic—*222-3*—indicate an example.

@$ Export modifier, 212

@$ Print modifier, 269

@% Export modifier, 212

@% Xtab modifier, 187

% FACTOR Xtab setting, 187

* Micro command, *see* Comment

@+/- Export modifier, 212

: MPF command construction notation, **389**, *407*

? Micro command, *see* Help

@12HH:MM Export modifier, 213

@12HH:MM:SS Export modifier, 213

@24HH:MM:SS Export modifier, 213

@ABbr Print modifier, 71

abbreviations in Micro command syntax, 70

abs function, 173

Addfield MicEdit VPF key, 52

aggregating data, 77

@ALIGNment Print modifier, 270

ALL Join keyword, 91, 237

ALL RAJ keyword, 91, 299

Alter MicDef command, *31*, **331**, *332*

Alter Micro command, *see* Change

AND connective on Find command, 71

Append MicDef command, *31*, *112*, *333*, **333**

arccos function, 172

arcsin function, 172

arctan function, 172

arithmetic functions, 172

array data format, 280, 326

ASCII export facility, 14, 214

audit facility, *see* command trace facility

AVE FACTOR Xtab setting, 186

average of field, 185, 190

avg function, 173

begin block record boundary, 42

Bottom MicEdit VPF key, 50

@BOTTOMBORDer Print modifier, 270

@BOTTOMMARgin Print modifier, 270

build MPF directive, *153*, 389, 408

C comments in MPF source code, 390

C field type, 23

C signals received by an MPF handler, 398

Calc.Problem field in *Problems* dataset, 171

Calculate Micro command, 7, *79*, *90*, *96*, **167**, *168-169*

  Problems dataset, 79

    records included in, 169

    values for Calc.Problem field, 171

  arithmetic functions, 172

  date functions, 173

  - field created by, 170
  - logarithmic functions, 172
  - miscellaneous functions, 176
  - time functions, 175
  - trigonometric functions, 172
- Call Micro command, **413**, *413*
- `_can_destroy` MPF function, 391
- `_can_replace` MPF function, 391
- `cancel` MPF macro, 389, 406, 408
- capitalization in command syntax, 163
- Cartesian product of two datasets, 95, 236, 239, 298, 301
- case sensitivity
  - in command syntax, 163
  - of categories, 24
- `CAT DESCRIPTIONS` Print setting, 428
- `_cat_abbr` MPF function, 394
- `_cat_cidval` MPF function, 395
- `_cat_cnt` MPF function, 392
- `_cat_cval` MPF function, 395
- `_cat_desc` MPF function, 394
- `_cat_iidval` MPF function, 395
- `_cat_ival` MPF function, 395
- `_cat_lookup` MPF function, 394
- `_cat_name` MPF function, 394
- `_cat_reqd` MPF function, 392
- categorical data, **24**
- *categories required* field attribute, 24
- `CATEGORY` Print setting, 259
- `@CATegory` Export modifier, 209
- `@CATegory` Print modifier, 268
- category, **24**
  - abbreviation, 24
  - adding default category, 107
  - adding from a file, 103
  - adding to existing field, 102
  - attributes, 20
  - case sensitivity, 24
  - changing name, 254
  - creating, 101
  - default, 25, 103, 105
  - name, 24, 67
  - searching on, 84
  - value, 24
- *category*, 164
- Category item menu in MicDef, *29*, *107*
- Category menu in MicDef, *29*, *102*
- CC field type, 23
- `@CENTered` Export modifier, 209
- `@CENTered` Print modifier, 269
- Change MicDef command, *see* Alter
- Change Micro command, *7*, *73*, *90*, **177**, *178*
- changing key record boundary, 42
- changing names of Micro objects, 254
- character searches, 84
- `@#CHaracters` Export modifier, 209
- `CLOCK` Command setting, 309
- Close Micro.Form command, *46*, *371*, **371**
- `@COLumn` Print modifier, 267
- Combine Micro command, *7*, *87*, *181*, **181**
- `@COMMA` Export modifier, 212
- *command*, 164
- command file, 140
- command reference, **163**
- command syntax generic terms, 164
- command trace facility, **21**, 310
  - changing file for, 331
  - MPF command, 399

- Comment Micro command, *16*, *183*, **183**
- computer conference for ILIR:Micro, 2
- CONTCHAR Command setting, 309
- conversion format file, 20, 22, 47, 131, 371, 381
- Copy MicDef command, *31*, **335**, *335*
- copying a dataset from another *userid*, 32, 119, 349
- correcting mistyped Micro commands, 66
- cos function, 168, 172
- COST Command setting, 309
- cot function, 172
- @Count Export modifier, 209
- @Count Print modifier, 266
- Count field from Xtab command, 77
- counting records in dataset, *257*
- Crosstabulate Micro command, *7*, *9*, *77*, *78*, *80*, *86*, *89*, **185**, *185*
- crosstabulated dataset, 9, 77
  - effect of Combine command, 182
  - effect of Join command, 240
  - effect of RAJ command, 302
- @CURRency Export modifier, 212
- custom programming for data entry, 48, 135

- data entry, 206
  - character strings, 37, 39
  - custom programming, 48, 135
  - error correction, 124
  - fixed format, **39**, *130*
    - begin block record boundary, 42
    - BOB record boundary, 42
    - changing key record boundary, 42
    - creating key field, 45
    - data length, 44
    - end block record boundary, 42
    - EOB record boundary, 42
    - filtering records, 42, 131
    - ILIR:Micro.Form prompts, 40
    - input data type, 44
    - missing data value, 44, 132
    - recoding, 44, 132, 377
    - record boundaries, 42
    - sampling records, 42, 131
    - subroutine for field value, 43
    - subroutine for record value, 42
  - free format, 37, **38**, *126*
    - default values, 39
    - end-of-record indicator, 39
    - error records, 39
    - repeat values, 39
  - interactive, **37**, 123
    - abbreviated prompting, 37
    - default value, 38
    - error correction, 38
    - external data, 38
    - MTS commands during, 38
    - multiple values, 37
  - methods, 35
  - Micro commands for, 15
  - Read command, 280
  - scaled data, 37
  - screen definition, 53
  - visual data entry, 49
- data file, 20, 47, 65, 111, 280, 304, 326
- *data required* field attribute, 24
- data security, 25
- data types in Micro, 23, 99
- data updating, 137, 322
- _data_file MPF function, 391

_data_reqd MPF function, 392
dataset, *5*, **20**
  changing name, 254
  conversion format file, *see* conversion format file
  copying from another *userid*, 32, 119, 349
  creating in command mode, 110
  creating in menu mode, 99
  creating with MTS macros, 32, *34*
  data file, *see* data file
  default attributes, 99, 111
  default file names, 22, 111
  destroyable attribute, 22
  dictionary file, *see* dictionary file
  directory file, *see* directory file
  documentation commands in MicDef, 32
  documentation commands in Micro, 15
  documentation formats, 25
  editing commands in MicDef, 31
  management commands in Micro, 13
  moving from another *userid*, 32, 119, 349
  name, 67
  overriding default attributes, 111, 345
  replaceable attribute, 22
  sharing dictionary file
    in ILIR:MicDef, 345
    in ILIR:Micro, 304
  structure, 20
  transferring from another *userid*, 32, 119, 349
dataset, 164
Dataset item menu in MicDef, *28*
Dataset menu in MicDef, *27*, *98*
Dataset switches menu in MicDef, *28*
@DATE Export modifier, 213
@Date Print modifier, 72, 269
date function, *8*, 79, 168, 173
date functions, 173
Datestamp Micro command, 7, **191**, *191*
day function, 174
DCA Print format, 14, 270, 271, 274
@DD-MMM Export modifier, 213
@DD-MMM-YY Export modifier, 213
@DD-MMM-YYYY Export modifier, 213
@DECimalplaces Export modifier, 210
default category, 25, **103**, 105
  filler data for outer join, 238, 300
default values
  filler data for outer join, 238, 300
  in data entry, 38, 39
Delete MicDef command, *31*, **337**, *337*
Delete MicEdit VPF key, 51
Delete Micro.Form command, *46*, **373**, *373*
@DELimiter Export modifier, 210
deprecated features, 411
Describe MicDef command, *32*, *339*, **339**
Describe Micro command, *15*, **193**, *193–194*
@Description Print modifier, 429
DESCRIPTIONS Print setting, 428
descriptive dataset documentation, 14, 25, 194
descriptive statistics, 9, 78, 185
Destroy MicDef command, *31*, *341*, **341**
Destroy Micro command, *12*, 13, *92*, *197*, **197**
destroyable dataset attribute, 13, 22, 111
_dfalt_cat MPF function, 394
_dict_file MPF function, 391
dictionary file, 20, 22, 65, 111, 304
  sharing

in ILIR:MicDef, 345
in ILIR:Micro, 304
DIRECTION Print setting, 260
directory file, 13, 21, 40, 47, 55, 65
Display Micro command, *16*, *199*, **199**
Doc MicDef command, *32*, *343*, **343**
Document Micro command, *15*, *67*, **201**, *202*
Document Micro.Form command, *46*, *133*, *375*, **375**
documentation formats for dataset, 14, 25
_ds_clos MPF function, 394
_ds_desc MPF function, 391
_ds_lookup MPF function, 391
_ds_name MPF function, 391
_ds_open MPF function, 394

e function, 172
E field type, 23, 38, 113, 179
ECHO Command setting, 309
ECHO Print setting, 428
Edit MicDef command, *31*, *111*, *345*, **345**
editing field attributes, 107
encryption of data, *see* scramble key
end block record boundary, 42
End Micro command, *see* Stop
Enter MicEdit VPF key, 51
Enterdata Micro command, *15*, *128*, *205*, **205**
EOB record boundary, 42
error correction
during data entry, 124
in command lines, 66
error records in data entry, 39
exceptional conditions in MPF execution, 396
_exec_micro_cmd MPF function, 407
execute MPF macro, 389, 406, 407
exp function, 172
Export Micro command, *12*, **207**, *207*
_ext_filename MPF function, 392
Extcall Micro command, *417*, **417**
external data file, 38, 85, 113
external field, *see* E type field

% FACTOR Xtab setting, 187
@FDEscription Print modifier, 429
Fdoc MicDef command, *32*, *115*, *347*, **347**
Fdoc Micro command, *see* Fulldoc
FIELD Print setting, 260
@FIELD Export modifier, 210
@FIELD Print modifier, 268
field, **22**
abbreviation, 99
attributes, 20
default values, 113
editing, 107
categories required attribute, 24
changing name, 254
creating, 99
data required field attribute, 24
floating point type, 23
inserting, 104
name, 67, 99
overlapping fields, 24
real type, 23
scale factor, 23
scaling function, 23
size, 23
types in Micro, 23
values by size, 23
field, 164
FIELD DESCRIPTIONS Print setting, 428

Field item menu in MicDef, *29*, *107*
Field menu in MicDef, *28*, *102*
field scaling, 116
file, 164
file names for datasets, 22
@FILL Print modifier, 268
filler data for outer join, 238, 300
filtering records in data entry, 42, 131
Find Micro command, 7, *69–71*, *73*, *83*, *86*, *90*, **219**, *220*
@FIrst Export modifier, 210
@FIrst Print modifier, 266
fixed format data entry, **39**, *130*
_fld_abbr MPF function, 392
_fld_cnt MPF function, 391
_fld_cval MPF function, 394
_fld_desc MPF function, 392
_fld_disp MPF function, 393
_fld_dval MPF function, 394
_fld_fact MPF function, 393
_fld_func MPF function, 393
_fld_ival MPF function, 394
_fld_len MPF function, 392
_fld_lookup MPF function, 391
_fld_name MPF function, 392
_fld_strval MPF function, 394
_fld_type MPF function, 392
floating-point field type, *see* scaled data
FORWARD Print setting, 260
@Forward Export modifier, 210
@Forward Print modifier, 266
@HEADings Export modifier, 210
free format data entry, 37, **38**, *126*
from_h_m_s function, 176
from_hhmmss function, 176
from_julian function, 174
from_y_m_d function, 175
from_y_m_d function, 168
from_yymmdd function, *8*, 79, 168, 174
Front End Processor, 395–396
_frst_cat MPF function, 395
_frst_fld MPF function, 392
full dataset documentation, 14, 25, 226
Fulldoc Micro command, *15*, **225**, *226*

generic terms in command syntax, 163–165
Get MicDef command, *31*, **349**, *349*
Get Micro command, *12*, 13, *64*, *83*, *93*, *229*, **229**, 278
@GF Join modifier, 236
@GF RestrictAndJoin modifier, 298
GoTo MicEdit VPF key, 50
GROWTH FACTOR Join/RAJ setting, 236, 298
growth factor, 239, 301

\halign TEX table, 274
HEADER CC Print setting, 260
@HEADJustification Export modifier, 210
@HEIght Print modifier, 268
Help MicDef command, *32*, *351*, **351**
Help MicEdit VPF key, 50
Help Micro command, *16*, *63*, **231**, *231–232*
error correction with, 233
@HExadecimal Print modifier, 268
hhmmss function, 176
hhmmss12 function, 176
HIRCC Print setting, 260
HORIZONTAL Print setting, 260
@Horizontal Print modifier, 266

horizontal Print layout, 71, 272
`hour` function, 175
hour from internal time, 175
`hour12` function, 175
`@HSIZE` Print modifier, 271, 274

IBM wordprocessor table, 270–271, 274
identifying fields for updating, 36, 137, 322
ILIR:Forum, 2
ILIR:MicDef, **27**, 65, 97
  Category item menu, ***29***, ***107***
  Category menu, ***29***, ***102***
  command mode, 30
  dataset documentation commands, 26
  dataset editing commands, 31
  Dataset item menu, ***28***
  dataset management commands, 31
  Dataset menu, ***27***, ***98***
  Dataset switches menu, ***28***
  Field item menu, ***29***, ***107***
  Field menu, ***28***, ***102***
  menu mode, 27
  Miscellaneous menu, ***30***, ***119***
  MTS macro mode, 32
  parameters, 97
  Show menu, ***30***, ***106***
ILIR:MicEdit, **49**, 55
  parameters, 49, 53, 55
  screen definition, 53
  VPF key definitions, 49
ILIR:Micro
  address of Micro Project, 2
  commands that produce a Result set, 7
  computer conference, 2
  consulting on use of, 2
  course on use of, 2
  dataset documentation commands, 15, 26
  dataset management commands, 13
  family of programs, 1
  ILIR:Forum, 2
  organization of documentation, 1
  overview, 5
  parameters, 161, 411
ILIR:Micro.Add, **35**, ***123***, ***127***
  parameters, 35
  prompts, 36
ILIR:Micro.Cnvrt, **46**, ***135***
  conversion process, 47
  parameters, 46
  prompts, 46
ILIR:Micro.Form, **40**, ***131***
  command mode, 46
  corrections mode, ***133***
  parameters, 40
  prompts, 40
ILIR:Micro.Up, **35**, ***137***, ***138***
  identifying fields, 36
  parameters, 35
  prompts, 36
  update fields, 36
ILIR:PGNT.H, 399
`Include` Calculate setting, 169
`@Increment` Export modifier, 210
`@Increment` Print modifier, 266
inner join, 237, 299
`INPUT` Command setting, 309
input type in data entry, 43–44
Insert MicDef command, ***31***, ***113***, ***114***, **353**, ***353***

int function, *8*, 79, 173
@INTeger Export modifier, 213
internal date, 168, **173**
    from 3-part date, 175
    from Julian date, 174
    from packed date, 174
internal time, **175**
    from 3-part time, 176
    from packed time, 176
interrupts in MPF code, 396
IS keyword on Find command, 84
_is_scrambled MPF function, 391
item, 164
Join Micro command, *7*, *10*, *91*, *93–95*, **235**, *235*
julian function, 174
Julian date, 168, **173**
    from Datestamp command, 191
    from internal date, 174
@Justify Export modifier, 210

key field, 10–11, 45, 76, *93*, 242
    checking uniqueness of, 85
Key Micro command, *7*, *76*, **241**, *241*

@LABEL Export modifier, 213
@LANDscape Print modifier, 271
@LEFTBORDer Print modifier, 270
@LEFTJustify Export modifier, 210
@LEFTJustify Print modifier, 269
@LEFTMARgin Print modifier, 266, 270
@LENgth Print modifier, 268
LIBRARY Macro setting, 428
LINES/PAGE Print setting, 260
link field, 9–11, *86*, 88, *93*, 112, 237, 294, 299
List Micro command, *64*, *69*, *83*, **243**, *243*
ln function, 172
local command trace file, **21**, 310
Lock Micro command, *12*, **245**, *245–246*
locking of dataset, 12, 245–248, 391, 392
log10 function, 172
logarithmic functions, 172
longjump MPF macro, 398
Lotus 1-2-3 export facility, 14, 216
@LPP Print modifier, 265

macro notation for dataset definition, 33
macros, *see* Micro Programming Facility
many-to-many data relationship, 94, 239, 301
master command trace file, **21**, 161, 310
MASTER ID User setting, 308
master userid, **21**, 308, 387
MATCHES keyword on Find command, 84
@MATCHES keyword on Find command, 221
MATCHING Join keyword, 91, 237
MATCHING RAJ keyword, 91, 299
max function, 173
_max_flen MPF function, 392
_max_val MPF function, 392
maximum of field, 185, 190
median of field, 185, 190
Menu MicDef command, *32*, **355**, *355*
Message Micro command, *16*, *249*, **249**
Micall Micro command, **421**, *421*
MicDef, *see* ILIR:MicDef
MicEdit, *see* ILIR:MicEdit
Micro, *see* ILIR:Micro
Micro Interface Functions, *145*, 390–395
Micro Programming Facility, 15, *141*, 385, *402–403*

activating library automatically, 387, 404
activating library with SET command, 386
C comments in source code, 390
C standard header files, 390
C typedefs
  dataset, field, category, record, 390
  field scaling functions, 393
  field types, 393
command tracing, 399
default library location, 308, 331, 387
default library name, 387, 404
dynamic loading, 390
exceptional conditions, 396
`execute` return code, 389
Front End Processor, 395–396
generating Micro commands, 389
include files
  C `#include` files, 385
  C standard header files, 390
  MPF target code header files, 388, 390
interrupt handling, 396, 400
line numbering of source code, 408
Micro Interface Functions, *145*, 390
  argument size definitions, 390
  category attribute functions, 394
  dataset attribute functions, 391
  field attribute functions, 392
  record access functions, 393
MPF command, 15, 385–386, 399
  invocation, 386
  parameter assignment, 386
MPF execution return codes, 397
object code file name, 404
preprocessor
    files used by, 404
    MTS macro for running, 405
    running, 404
    target code generated by, 405
  program interrupts, 400
  reserved words, 388
Micro.Add, *see* ILIR:Micro.Add
Micro.Cnvrt, *see* ILIR:Micro.Cnvrt
Micro.Form, *see* ILIR:Micro.Form
Micro.Up, *see* ILIR:Micro.Up
`_microcmd` MPF variable, 389, 407
`_micrortn` MPF variable, 389, 407
Microsoft Word table, 270, 271, 274
MIDAS, *see* STAT:MIDAS
`min` function, 173
`_min_val` MPF function, 392
minimum of field, 185, 190
`minute` function, 175
minute from internal time, 175
miscellaneous functions, 176
Miscellaneous menu in MicDef, *30*, 119
miscellaneous MicDef commands, 32
miscellaneous Micro commands, 16
missing data value in data entry, 44
`@MM/DD` Export modifier, 213
`@MM/DD/YY` Export modifier, 213
`@MM/DD/YYYY` Export modifier, 213
`@MMM-YY` Export modifier, 213
`@MMM-YYYY` Export modifier, 214
`month` function, 174
Move MicDef command, *31*, **357**, *357*
moving a dataset from another *userid*, 32, 119, 349
MPF command, 15, 385, 386, 399

MTS information, 57

MTS Micro command, *16*, **251**, *251*

@NAME Print modifier, 429

Name Micro command, *12*, *70*, *87*, **253**, *253–254*

names of Micro objects, 21, 67

  changing, 254, 331

Next MicEdit VPF key, 51

_next_cat MPF function, 395

_next_fld MPF function, 392

@NO% Xtab modifier, *see* @NOPERCENT

@[NO]CC Print modifier, 265

@[NO]HEADings Print modifier, 266

@NOPERCENT Xtab modifier, 86, 187

@NOPRint Xtab modifier, 86, 187

@[NO]RULEs Print modifier, 271

not-restriction, 295

number, 164

Number Micro command, *see* Key

@NV Datestamp command modifier, 192

one-to-many data relationship, 93, 239, 301

one-to-one data relationship, 93, 238, 300

Oops MicEdit VPF key, 51

Options MicEdit VPF key, 51

OR connective on Find command, 73

OSIRIS export facility, 14, 215

outer join, 237, 299

OUTLEN Command setting, 310

OUTLEN Print setting, 260

@OUTLEN Print modifier, 267

OUTPUT Command setting, 310

overlapping fields, 24

packed date, 168, **173**

  from Datestamp command, 191

  from internal date, 174

packed time, **175**

  from internal time, 176

@PAD Print modifier, 265

@Page Print modifier, 265

PAGE CONTROL Print setting, 261

PAGECC Print setting, 261

pattern, 164

@PC Print modifier, 265

PDATE Print setting, 261

PERCENT Xtab setting, 187

@PERCent Export modifier, 212

@PERCent Xtab modifier, 187

Percent field from Xtab command, 77

PERIOD Command setting, 428

permanent dataset, **13**, *69*, 288

  and Purge command, 278

  and Release command, 282

  and Update command, 323

  from Save command, 81

Permit MicDef command, *31*, **359**, *359–360*

PgDn MicEdit VPF key, 51

pgnt_info MPF macro, 400

pgnt_message MPF macro, 400

PgUp MicEdit VPF key, 51

pi function, 172

@PITCH Print modifier, 267

pm function, 175

@POINTsize Print modifier, 271

@PORTrait Print modifier, 271

Previous MicEdit VPF key, 51

PRINT Xtab setting, 187

@PRint Xtab modifier, 187

Print Micro command, *12*, *65*, *66*, *69*, *71*, *72*, *77*, *79*, *85*, *86*, *88*, **257**, *257–259*
  command and field modifiers, 262
  settings, 259
Problems dataset, 79
  records included in, 169
  values for Calc.Problem field, 171
prompt MPF macro, *153*, 395, 406
@PROTected Export modifier, 211
Purge Micro command, *12*, 14, **277**, *277*

RAJ Micro command, *10*, *see* Restrictandjoin
RAM Micro command, *see* Restrictandjoin
rand function, 176
Read Micro command, **279**, *279*
READY Command setting, 310
@REAL Export modifier, 214
real field type, *see* scaled data
_recd_cnt MPF function, 391
_recd_lookup MPF function, 394
recnum function, 169, 176
Recode Micro.Form command, *46*, *134*, **377**, *377*
recoding in data entry, 44, 132, 377
record boundaries in data entry, 42
@RECORDSPERPAGE Print modifier, 271
Redo MicDef command, *31*, 115, **361**, *361*
Redo Micro.Form command, *46*, *134*, *379*, **379**
Release Micro command, *12*, 14, *74*, *87*, **281**, *281*
Remove Micro command, *7*, *87*, **283**, *284*
Rename Micro command, *see* Name
repeat values in data entry, 39
Replace Micro command, *12*, 13, *90*, *287*, **287**
replaceable dataset attribute, 13, 22, 111
Resettings Micro command, *16*, **289**, *289*
Restrict Micro command, *7*, *9*, *86*, *88*, **293**, *293–294*
RestrictandJoin Micro command, *88*
Restrictandjoin Micro command, *7*, **297**, *297*
Result dataset, **6**
  Micro commands that produce, 7
REVERSE Print setting, 261
@Reverse Export modifier, 211
@Reverse Print modifier, 266
@RIGHTBORDer Print modifier, 270
@RIGHTJustify Export modifier, 211
@RIGHTJustify Print modifier, 269
@RIGHTMARgin Print modifier, 270
RTF Print format, 14, 270, 271, 274
@RULEs Print modifier, 271
rundate function, 173
RUNFILE User setting, 308
runfile, **21**, 139, 161, 331
runtime function, 168, 175

S field type, 23
sampling records in data entry, 42, 131
SAS export facility, 14, 215
Save MicEdit VPF key, 51
Save Micro command, *12*, *81*, **303**, *303–304*
saving temporary datasets, 282, 304
SC field type, 23
scale factor, 23, 116
scaled data, 23, 37, 116
scaling function, 23, 116, 393
Scan MicEdit VPF key, 52
ScanAgain MicEdit VPF key, 52
@SCIentificnotation Export modifier, 214

scramble key, 22, 36, 47, 66, 81, 111, 280, 288, 326
scrambled dataset, 288
screen definition in data entry, 53
second function, 175
second from internal time, 175
security, 25
Select Micro command, *7*, *75*, *85*, *90*, **305**, *305*
setjump MPF macro, 398
\settabs TEX table, 274
Settings Micro command, *16*, *63*, *80*, *139*, *140*, *307*, **307**
sharing a dictionary file
    in ILIR:MicDef, 345
    in ILIR:Micro, 304
Show menu in MicDef, *30*, *106*
Show MicDef command, *see* Fdoc
Show Micro.Form command, *see* Document
sigdate function, 173
sign function, 173
sigtime function, 175
sin function, 168, 172
Sort Micro command, *7*, *74*, *79*, **313**, *313–314*
sorting records in a dataset, 9, 314
sorting records in datasets, 74
@SPaces Print modifier, 268
spreadsheet export facility, 14, 216
SPSS export facility, 14, 216
sqrt function, 168, 173
@SSn Export modifier, 214
@SSn Print modifier, 72, 269
standard deviation of field, 185, 190
STAT:MIDAS, 78, 326
STAT:Textedit, 14, 270, 271, 275
statistics, descriptive, 9, 78, 185
STD FACTOR Xtab setting, 187
stop MPF macro, 395, 406
Stop MicDef command, *363*, **363**
Stop MicEdit VPF key, 51
Stop Micro command, *81*, *96*, **315**, *315*
Stop Micro.Form command, *46*, *135*, *381*, **381**
@STratification Print modifier, 267, 272
stratification of Print output by field, 77, 272
string, 164
subroutine for data entry field, 43
subroutine for data entry record, 42
sum function, 173
syntax, Micro command
    abbreviations, 70
    capitalization, 163
    case sensitivity, 163
    generic terms, 164
    synonyms, 70
System Micro command, *see* MTS

@TABSKIPglue Print modifier, 271, 275
\tabskipglue for TEX, 275
Tabulate Micro command, *see* Crosstabulate
tan function, 172
Tdoc MicDef command, *32*, **365**, *365*
Tdoc Micro command, *see* Techdoc
Techdoc Micro command, *15*, **317**, *318*
technical dataset documentation, 14, 25, 318
temporary dataset, **13**, *69*, 288, 315
    and Purge command, 277
    and Release command, 282
    and Update command, 323
    saving as permanent, 81, 304
TEX, 14, 270, 271, 275

TEXH Print format, 14, 270, 271, 275
TEXS Print format, 14, 270, 271, 275
    column separators, 275
    column widths, 275
    nominal character size, 275
@TEXT Export modifier, 213
TEXTEDIT Print format, 14, 270, 271, 275
    column separators, 275
    column widths, 275
    multi-page tables, 275
    nominal character size, 276
THEIR keyword on Join and RAJ commands, 94
TIME Command setting, 310
@TIME Export modifier, 214
time function, 168, 175
time functions, 175
TITLE Print setting, 261
@TITLE Print modifier, 265, 274
TITLE CC Print setting, 261
Top MicEdit VPF key, 50
@TOPBORDer Print modifier, 270
@TOPMARgin Print modifier, 270
total of field, 185, 190
TRACE Command setting, 310
transferring a dataset from another *userid*, 32, 119, 349
trigonometric functions, 172
Turn Micro command, *see* Settings

U field type, 23
UC field type, 23
Undelete MicEdit VPF key, 51
UNDERLINE Print setting, 261
@UNDERLINE Print modifier, 267
Undo MicEdit VPF key, 51
unique field in dataset, *see* key field
unit, 164
unload MPF macro, 396
_unloadsw MPF variable, 390, 396
unlocking of dataset, 13, 248
update fields, 36, 137, 322
Update Micro command, *15*, *138*, **321**, *322*
Use Micro command, *423*, **423**
userid, 164

@VALue Print modifier, *85*, 269
VALUES Print setting, 428
VERIFY Command setting, 310
VERTICAL Print setting, 261
@Vertical Print modifier, 71, 267, 272
vertical Print layout, 272
VIFCC Print setting, 262
VIRCC Print setting, 262
visual editing of dataset, 49

weeding of duplicate records, 9, 75, 86, 306
weekday function, 174
@WIDth Print modifier, 268
WINDOW Print setting, 428
@Window Print modifier, 429
WK1 export facility, 14, 216
WL MicEdit VPF key, 50
wordprocessor table, 14, 270, 271
WR MicEdit VPF key, 50
Write MicDef command, **367**, *367*
Write Micro command, *78*, **325**, *325*

Xtab Micro command, *see* Crosstabulate

year function, 174

yr function, 174

yymmdd function, 168, 174

yyyymmdd function, 174

@ZIP Export modifier, 214

@ZIP+4 Export modifier, 214

@ZIP+4 Print modifier, 269

@ZIPcode Print modifier, 269