The

TABLES Memory Manager

Reference Manual

Version 3.2

DISCLAIMER

The information presented is subject to change. Every effort has been taken to ensure that this manual is accurate and up to date. However, Specialized Solutions, Inc. disclaims any liability for any loss or damage resulting from the use of this manual.

May 26, 2013

This edition applies to Release 3.2 of **TABLES/MM**

The TABLES Memory Manager
Reference Manual

TABLE OF CONTENTS

1. INTRODUCTION

Introduction

The **TABLES Memory Manager** is a high performance, easy to use system for managing and accessing data in memory. Its primary function is to allow all applications to very quickly and easily **SEARCH** for and **RETRIEVE** data stored in either common memory or local memory. MVS/ESA common dataspaces are used to allow all online and batch applications to share common data and a region's local memory can be used for application specific data.

In most applications today, there are many files, databases, or simply lists of values that are retrieved very often but updated infrequently or on a scheduled basis (eg. daily, weekly, etc). These types of **tables** are used for a wide variety of purposes and contain information such as rates, manufacturing codes, field validation rules, control codes, scheduling information, conversion tables, state codes, zip codes, etc. By placing these types of tables in memory and making them easier to access, many potential benefits result:

- One of the most important is to reduce the CPU and elapsed time to access data. By using common dataspaces or local memory, data can be accessed with considerably less CPU time and virtually no I/O. As a result, response times and throughput can be dramatically improved, benefiting both the system and its users.

- Another important benefit is to simplify applications. By using TABLES/MM, an application can make a single call to the interface routine to search for and retrieve a row from any table, either in common or local memory. No initialization calls, opens, pre-compile steps, special system gens, binds, etc., are required. Also, since data is retrieved from memory, applications do not require any specific database or access method processing. The data can therefore be easily migrated from one database system to another without affecting the application. This can help to significantly reduce the programming effort and maintenance of applications.

TABLES/MM Components

The TABLES/MM system has four main functional components. Each are very distinct in what they are used for and by whom. This Reference Manual is broken into four additional sections that correspond to the components as follows:

- **Dataspace Setup**
- **Online Facilities**
- **Batch Utilities**
- **Application Programming Interface**

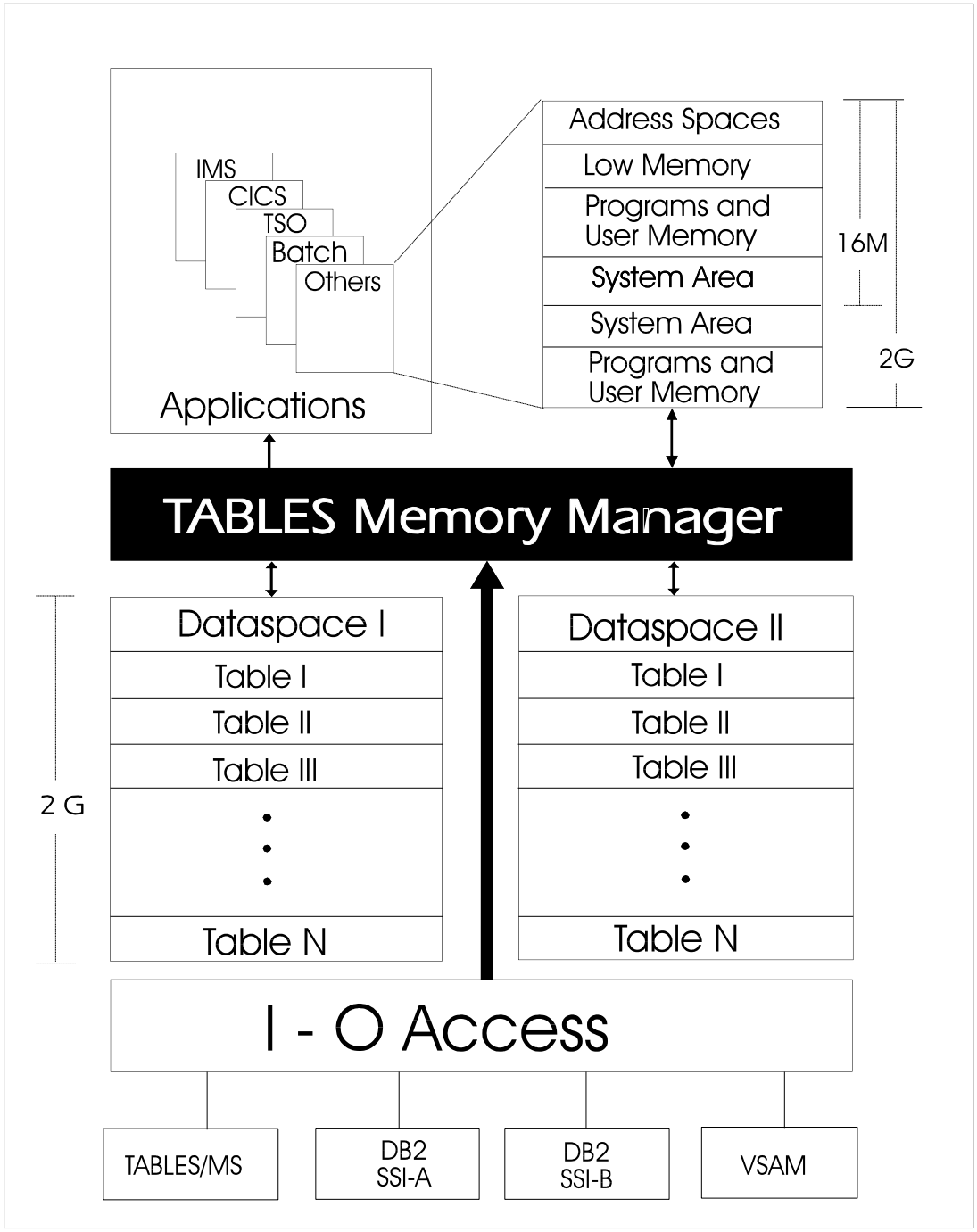
Dataspace Setup is used for creating common dataspaces and making them available to all applications. This will generally be done when the system is started. Once started, a dataspace is normally never terminated unless all systems using it have ended or the system is to be shut down. Section 2 covers Dataspace Setup, including an overview of dataspaces, the structure of TABLES/MM dataspaces, and starting and terminating dataspaces. This section should be reviewed by anyone that needs to start and stop or use TABLES/MM dataspaces.

The **Online Facilities** are used to monitor and control active dataspaces. All information about dataspaces can be displayed, tables can be loaded, freed or browsed, and statistics and other information about the tables is available. This will be most useful for personnel that will be controlling and monitoring the dataspaces as well as for application groups initially setting up tables to be loaded into memory.

The **Batch Utilities** are used for defining VSAM tables, views and indexes, maintaining and reporting on dataspaces and for testing application calls to the interface. By setting up utility jobs, dataspaces can be pre-loaded with all required tables or reloaded at certain intervals. In addition, statistics can be generated to monitor dataspaces on a periodic basis.

The **Application Programming Interface** (API) is used to read and search for data and to monitor and control tables within a dataspace or local memory. A high performance search facility and a flexible set of functions allow data to be easily retrieved in a variety of ways. The data can be searched by specific fields or scanned in both forward and backward directions. In addition, a special call allows for retrieving rows which are time sensitive based on start/stop dates. All functions and requirements for using the API are described in Section 5.

The API is the most complex component used within TABLES/MM and the most important. It allows applications to dramatically improve their performance and simplify table access. As depicted in the following diagram, the API handles all data flow to and from the program. It retrieves data from the I/O interfaces when loading tables, from dataspaces and from local memory. It sends data to the dataspaces and local memory when loading tables and to applications when requested. Maintenance and control of local memory, dataspaces, and I/O for loading tables is all handled internally by TABLES/MM. The applications only requirement is to call the interface to retrieve data in the way it needs it.



In the diagram, arrows show where data flows from and to. While the TABLES Memory Manager can handle the flow from many dataspaces, an application (eg. batch job, online region, etc) can only be setup to retrieve data from **one dataspace** in addition to local memory.

Preparing to use TABLES/MM

Once the TABLES/MM product is installed, it requires very little effort to setup and use. The following steps are a guideline of what should be done next:

(1) Based on your installation and applications, decide what and how many dataspaces are needed. Setting standards for what dataspaces will be used and by whom, is very important to making the system easier to use and maintain. The Dataspace Setup described in Section 2 should be reviewed carefully.

(2) Once the dataspace requirements are defined, use the procedures defined in Section 2 to create the dataspaces.

(3) Execute the Online Facilities to review the dataspaces to make sure they were setup and defined correctly. Additionally, any tables that require date/time controls or sort fields can be defined at this time. Refer to Section 3 when using the Online Facilities or use the online help tutorial.

(4) Using the Batch Definition Utility, define any VSAM tables required, indexes to be used with the VSAM or DB2 tables, and any views to be associated with them.

(5) Setup and run the Batch Utilities to pre-load all tables into the dataspaces. This allows the dataspaces to be fully ready for access before they are needed. The batch utilities can also be scheduled like any production system to initially load tables, re-load tables, generate statistics or to perform any other required function.

(6) Lastly, call the API from the applications.

Once steps 1 through 4 are completed, both the online and batch facilities can be used to monitor the dataspaces and usage. For initially checking out the system and for testing new applications the online facilities are faster and easier to use. Once ready for production, the batch utilities allow for automating these tasks.

2. DATASPACE SETUP

Dataspace Setup

Overview

In order to make data accessible to all online and batch systems, TABLES/MM uses MVS/ESA Common Dataspaces. A dataspace, like any address space or region in the system, uses standard main memory and can be paged and swapped out in the same manner. The memory is also protected in the same manner in that it cannot be overlaid by accident. Like address spaces, dataspaces can be up to the maximum of 2 gigabytes in size. Further, the memory in a dataspace is processed with the same set of assembler instructions (eg, move, compare, etc) as with normal address space memory. However, to access the memory in a dataspace or even another address space, special MVS/ESA processing, called Access Register mode (or AR mode for short) has to be activated and used. This mode is turned on and off by special assembler instructions and requires the use of new ESA registers, called access registers. Once setup correctly, the standard instructions work as normal.

A dataspace has several significant differences from an address space that make them especially useful for managing and accessing large amounts of data. These are summarized as follows:

(1) Programs or instructions can not be executed within a dataspace. They can be stored there as data but not executed. This makes the data in a dataspace much less likely to be adversely affected or overlaid by erroneous programs as is the case where the data is in the same address space.

(2) Since MVS does not allow executing instructions in a dataspace, it does allow the full 2-gigabytes of memory to be used. In normal address spaces, some of the memory is used by the system nucleus and common storage areas like CSA, LPA, etc. In a dataspace, this is not true. All memory in a dataspace from location 0 to 2 gigabytes is directly addressable making it more efficient to allocate and maintain.

(3) Finally, there is a special type of dataspace, called a Common Dataspace. It is special in that MVS/ESA automatically makes it available to all address spaces in the system. This makes it especially useful in sharing data between applications without requiring a lot of resources.

Since the common dataspace is accessible to all address spaces and requires special entries in system control blocks, only a limited number of them can be created. As a result, there are special requirements for creating and maintaining common dataspaces. Therefore, TABLES/MM must be installed with the correct authority to create common dataspaces. It will then automatically take care of any system requirements necessary to maintain them.

Dataspace Structure

To effectively and efficiently use dataspaces, it is important to know how TABLES/MM builds and maintains them. The structure and processing is the same for all dataspaces created by the TABLES Memory Manager.

The basic structure of a TABLES/MM dataspace can be conceptually thought of like a Partitioned Data Set (PDS). There is a main control block that corresponds to the Volume Table of Contents (VTOC) entry for the PDS. This describes the basic information about the dataspace and where the directory is located. Like a PDS, the directory in a dataspace contains information for each table (eg. member) stored in the dataspace and where it's located. As tables are loaded, directory entries are used. Unlike a PDS, once a directory entry is used, it cannot be reused. This is to ensure stability and reliability of the data and allows statistics to be maintained and tracked for all tables until the dataspace is terminated.

As tables are loaded, space within the dataspace is used up. When tables are freed, space is left unusable in the same way when a member of a PDS is deleted. The directory entry for the table remains, but the space it used does not. Also, when a table is reloaded, if there is not enough free space and the table has grown, the old area becomes unusable, as if it were freed, because the table has to be loaded into a new location. Therefore, over time, if a lot of freeing and reloading of tables takes place, a dataspace may run out of available memory. When this occurs, a compress of the dataspace can be done. Like a PDS compress, all tables are moved forward in the dataspace, leaving all unused memory at the end of the dataspace and available for use.

Unlike most PDS's, when tables are replaced (eg. reloaded) in memory, the same area can be used, allowing tables to be reloaded without using additional memory within the dataspace. In addition, a free space value can be specified when loading a table that allows it to be reloaded at the same location even if it has grown in size.

Creating Dataspaces

To create a dataspace, the TMMSTART utility is executed. This program issues the request to the system to create the dataspace, initialize it, and sets up communications to allow the programming interface to connect to it. To be able to create a common dataspace, special authority and requirements must be met. The following must be done for the dataspace to be created and accessed correctly:

- TMMSTART program must be APF authorized. This means it must be linked with AC=1 and the load module must be placed in a special system library that is defined as having APF authorization. This is normally done when the product is installed.

- The job that creates the dataspace must be made non-swappable. When an address space is swapped out, any dataspace created by it is also swapped out. As a result, any other address space would not be able to access it. Therefore, TMMSTART automatically sets itself to be non-swappable.

- For a common dataspace to be always available, the job that creates it must be running at all times. To ensure this happens, the TMMSTART utility does two things. First, it does not allow the system to terminate it due to waiting too long. Second, after completing the dataspace initialization and setup, it goes into a permanent wait, until an operator requests the dataspace to be terminated. Therefore, once created, a dataspace will always be available until the operator terminates it or cancels it.

As a result of these requirements, it is recommended that TMMSTART be executed as a started task instead of a standard batch job. This eliminates the need for special initiators and allows for better control by operations. However, for initial testing of dataspaces or non-production type dataspaces, a batch job is perfectly acceptable and will work exactly the same as a started task.

The JCL in figure 2-1 is the TMMSTART procedure used to create TABLES/MM dataspaces. The DD cards and parameters used for defining the dataspace are described below.

```
//* ******************************************************
//*   TABLES MEMORY MANAGER
//*   (C) COPYRIGHT 2013 SPECIALIZED SOLUTIONS, INC.
//* ******************************************************
//* PROC: TMMSTART
//*
//* DESCRIPTION: START A TABLES/MM DATASPACE.
//* ******************************************************
//TMMSTART PROC  S='*',RGN=2M,          SYSOUT CLASS, REGION
//         DID=00,                      DATASPACE ID
//         SIZE=1,                      SIZE IN MEGABYTES
//         DIR=100,                     DIRECTORY ENTRIES
//         KEY=8,                       STORAGE KEY
//         DESC='TMM DATASPACE',        DESCRIPTION
//         CNTL=NULLFILE                CONTROL CARD INPUT
//*
//TMMSTART EXEC PGM=TMMSTART,REGION=&RGN,
//         PARM='&DID,&SIZE,&DIR,KEY=&KEY,&DESC'
//STEPLIB  DD  DSN=SYS1.LINKLIB,DISP=SHR
//SYSUDUMP DD  SYSOUT=&S
//CONTROL  DD  DISP=SHR,FREE=CLOSE,DSN=&CNTL
```

The DD cards used are as follows:

STEPLIB - The load library that contains the TMMSTART program. It must be defined to the system as being APF authorized.

SYSUDUMP - Optional, and only used if an abend occurs.

CONTROL - Optional, is used to contain control card input to TMMSTART to limit which tables can be loaded into the dataspace.

The RGN= and S= parameters are for JCL purposes and can be set to any normal value. Also, the region size has no affect on the size of the dataspace or limits it in any way.

The other parameters used in the procedure define the specifics about each dataspace. These can be the same or different for each dataspace created except for DID, which must always be different.

DID - This is the Dataspace ID. It is extremely important and should be set with care. It must be two alpha-numeric characters (eg. A-Z or 1-9) and must be different for each active dataspace. A dataspace with the ID='00', is the default dataspace. That is, the programming interface defaults to using dataspace '00' if not told otherwise. Therefore, all online systems and batch jobs will use it unless a special DD card is placed in each job to use an alternate dataspace.

SIZE - This is the size of the dataspace in megabytes. The maximum value is 2000. However, keep in mind that the larger the size, the more system resources (eg. memory, paging space) are required.

DIR - This is the number of directory entries to allow. That is the number of different tables that can be loaded. Remember that even if a table is freed, the directory entry remains. Generally allow for the maximum number of tables ever expected to be loaded.

KEY - This specifies the storage key of the memory the dataspace will use. Only KEY=8 or KEY=9 are valid. The default of KEY=8 should always be used except when the dataspace will be used by CICS transactions running with EXECKEY(USER). CICS transactions running in USER key require the dataspace to use KEY=9 storage. Additionally, specific hardware that supports Sub-system Storage Protection (SSSP) is also required to use KEY=9. If KEY=9 is used, the dataspace is still accessible to all CICS transactions (USER or CICS key) and all batch regions as well.

DESC - This is simply a description of the dataspace. Enter up to 40 characters in single quotes. The description is simply displayed by the online utility and not used otherwise.

CNTL - This defines the control card input file and is optional. It is used to specify the set of tables that can be loaded into this dataspace (See below for more information).

Limiting Tables in Dataspaces

Any TABLES/MM dataspace can be created with a fixed, preset directory of tables that can be loaded into it. By specifying the CNTL= parameter when starting a dataspace, the control cards in the specified dataset are processed by TMMSTART and each table name in the order specified is placed in the directory. After all cards are read, the size of the directory is set to not allow any additional tables to be loaded into the particular dataspace. This can be used to limit a dataspace to a very specific set of tables to control its use and enforce the tables that can be loaded into it.

The dataset specified must have standard 80-byte records and any block size. The control cards themselves must be as follows:

- **An asterisk (*) in column 1 denotes a comment card**
- **A blank card is ignored**
- **Table names must start in column 1 and**
- **The maximum table name allowed is 36 characters**

The control cards do not cause the tables themselves to be loaded. The only affect is to enter the name in the directory and fix the size of the directory. The tables must be loaded later, through the online or batch utilities or directly by a program.

Also be aware that the table names on the control cards are not validated. Any value in columns 1 through 36 can be entered. The table is not checked for invalid characters or format errors. In addition, the table is not checked to see if it exists. However, the only affect an invalid table name would have is to waste a directory entry. (See Section 5 on the API for a description of the table names and specifications).

Finally, since the directory size is set based on the number of tables specified, the DIR= parameter has no affect. It is overridden based on the input and is ignored.

Terminating Dataspaces

In general, once a dataspace is started, it should not be terminated until the system is stopped or all jobs accessing it are stopped. If a job attempts to access a dataspace that has been terminated, results are unpredictable with a system abend likely to occur.

To stop a Dataspace, the system operator must reply correctly to the message issued when the dataspace is started. After TMMSTART correctly initialize and makes a dataspace available, the following message is displayed on the system console:

TMMS001R REPLY "STOP??" TO END D/S-??-JJJJJJJJ

In each case, the **??** is replaced with the dataspace id as entered for the DID= parameter on the start command and **JJJJJJJJ** is the Jobname. The operator must reply as follows:

STOP??

Where the ?? must be replaced with the correct dataspace id. This is to ensure that a reply is not sent to the wrong TMMSTART job and incorrectly terminate a dataspace in use.

Once terminated, the dataspace and all tables loaded into it are no longer available. Any application attempting to access it will get an error code from the interface or abend.

If the wrong dataspace-id is entered in a reply, message TMMS003E is displayed, specifying an INVALID REPLY was entered and no dataspace is terminated. Message TMMS001R is then re-displayed. If an incorrect id was entered, simply reply again with the correct id. If the correct id was entered, but the wrong message was responded to, just reply to the correct message.

Examples

1. To start the default dataspace with 25 megabytes of memory and allowing for 500 tables, the following command would be issued by an operator:

S TMMSTART,SIZE=25,DIR=500

If everything was setup correctly, the reply message should appear on the console allowing the operator to terminate dataspace with ID='00'.

2. To start a special application dataspace with ID='1A', 100 megabytes of memory and allowing for 100 tables, the following start command would be issued:

S TMMSTART,DID=1A,SIZE=100,DIR=100,DESC='Appl D/S'

As a result, the following message would appear on the system console to allow the dataspace to be terminated:

TMMS001R REPLY "STOP1A" TO END D/S-1A-TMMSTART

The operator would then have to reply "STOP1A" to terminate this dataspace.

3. To start a dataspace with a fixed set of tables and 10 megabytes, the operator would issue the following command:

S TMMSTART,DID=1S,SIZE=10,CNTL='TMM.LIB.CNTL(SETDS1S)'

Where the CNTL file might have control cards as follows:

```
*
* TABLES TO BE FIXED IN DATASPACE WITH ID '1S'
*
TMM.TABLE TEST1
TMM.TABLE TEST2
TMM.TABLE TEST3
```

In this case, a special dataspace is setup to allow three tables to be loaded into it and only those three. The standard reply message would also appear on the console once the dataspace is ready.

> ***Note:*** If any errors were detected in these examples, message TMMS002E would have been displayed on the console specifying what the error was. Please refer to the Appendix for a description and actions for this message

3. ONLINE FACILITIES

Online Facilities

Overview

The Online Facilities are used for monitoring and controlling all active TABLES/MM dataspaces and local memory tables, and for defining effectivity for DB2 tables. There are two implementations. The first, uses standard TSO/ISPF full-screen interfaces for displaying information, help panels, scrolling, messages, and PF key processing. The second is an IMS/CICS online transaction that allows a subset of the TSO facility with an additional ability to monitor and control tables in local memory for the IMS MPR or CICS region where the transaction executes.

On TSO

The TSO Online Facility is the primary mechanism to control TABLES/MM Dataspaces. It has two main components. The first is the Dataspace Utility. It allows all dataspaces to be controlled and monitored interactively. It includes the following functions:

- **List all active dataspaces and their descriptions**
- **List active clients for a dataspace**
- **Display current statistics and the directory for a dataspace**
- **Display statistics and other information for a specific table or view**
- **Load, Free, and Browse Tables and Views**

The dataspace utility can be used to monitor such things as how much access there is to a dataspace and its individual tables, how much space is being used, if there is any room left and other things. In addition, tables can be checked for size and accesses, they can be periodically re-loaded or browsed as required and monitored periodically to see what is being accessed.

The other component is Effectivity Definition. It is used to specify date control support for DB2 tables that are maintained with start and stop dates. This allows for rows that fall within these start and stop dates (e.g. the effective dates) to be retrieved with a single call to the API.

> ***Note:*** In prior versions, effectivity definition could also be used to define the key fields or sort order for the table. This is not recommended any more. Instead, the TMMDEFN utility should be used to define a primary index.

In general, the dataspace utility will be used by support staff and operations personnel to monitor and control production dataspaces. Applications personnel can also make use of it when initially setting up and testing access to new tables with test dataspaces. The effectivity definition is generally used by application groups when setting up a table to be loaded and accessed.

On IMS or CICS

The IMS/CICS Online Facility is a subset of the TSO one. It allows for loading, freeing and getting statistics of tables like TSO but only for the one dataspace attached to the region. However, unlike TSO, tables in local memory can be processed as well. In addition, tables can be listed and API functions can be executed (eg. GETF, STGF, etc.). However, tables in memory can not be browsed and Effectivity can not be defined.

The IMS/CICS transaction can be executed in Full-Screen mode and also in Line Mode. In full-screen mode, it can be used to interactively monitor and control tables like the TSO facility. In line mode, it can be used to pre-load tables automatically at CICS start-up or using automated operations procedures at IMS start-up. Tables can be loaded into dataspaces or local memory (that is, in the IMS region or CICS region).

TSO Online Facility

To access the TSO Online Facility, the TABLES/MM Primary Menu must be displayed. To get to the primary menu do the following :

(1) Log on to TSO using your normal procedures.

(2) From TSO READY mode, or ISPF Option 6, enter the following:

%TMM

(3) Then press the ENTER key.

The TABLES/MM Primary Menu shown below should then be displayed. Be aware that many installations change the method to display the primary menu. Therefore, if you have problems getting to it, please check with your local support group.

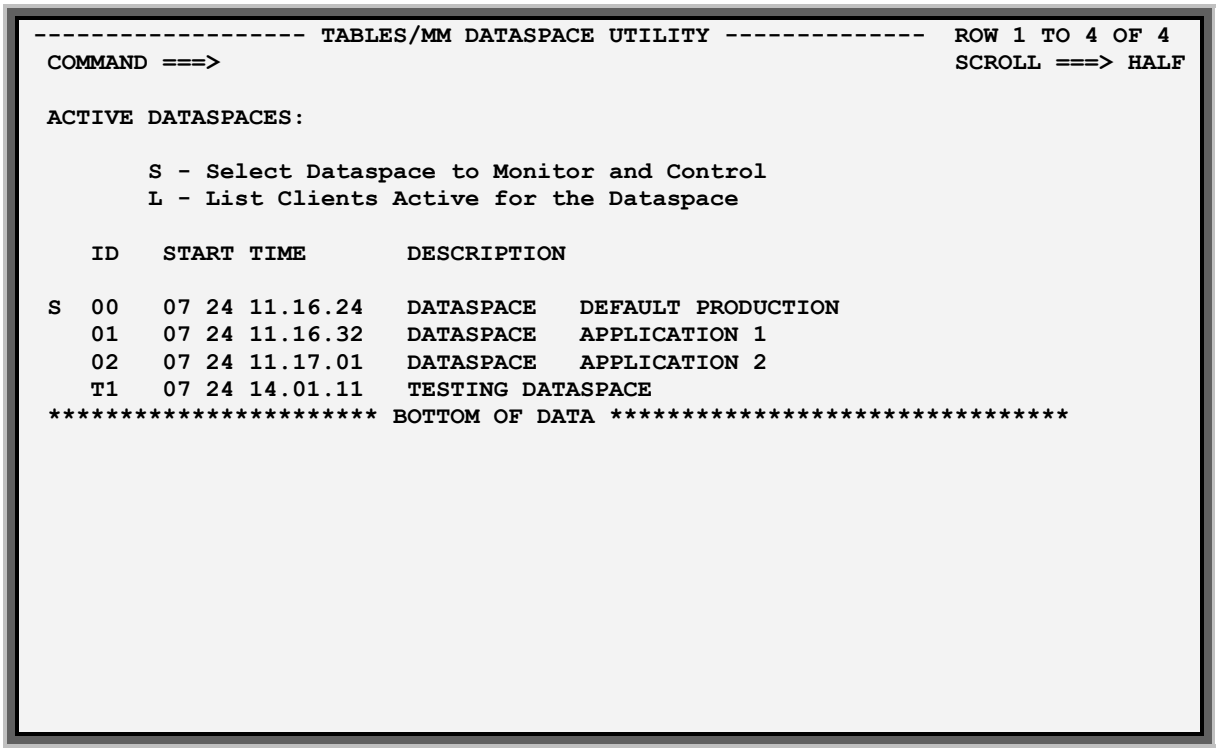
```
--------------------------- TABLES/MM PRIMARY MENU ----------------------------
OPTION ===>

   1 DATASPACE UTILITY        Monitor and Control TABLES/MM Dataspaces
   2 EFFECTIVITY DEFINITION   Maintain Effectivity Records for DB2 Tables


Optionally Enter DB2 Plan and Subsystem :

     DB2 Plan      ===>
     DB2 Subsystem ===>









                    TABLES MEMORY MANAGER - V3.2
            (C) COPYRIGHT 2013 - SPECIALIZED SOLUTIONS, INC.
                          ALL RIGHTS RESERVED
```

Once the primary menu is displayed, select the appropriate option to execute the Dataspace Utility or Effectivity Definition. Pressing the END PF key (PF3/15) will exit from the TABLES/MM Online Facility. Pressing the HELP PF key (PF1/13) will activate the help tutorial.

In addition to the option field, there are two other optional input fields, the DB2 Plan and the DB2 Subsystem. These are used to specify which DB2 system to use and the appropriate plan. These are normally set at installation and should not be required. If there are any problems in loading tables or using Option 2, Effectivity, check with your local support group for specify the DB2 subsystem or plan.

Dataspace Utility

To activate the Dataspace Utility, enter Option 1 on the primary menu and press ENTER. The Dataspace Utility panel shown in the next figure should be displayed. However, if there are no active dataspaces, the primary menu remains and the message "NO ACTIVE DATASPACES" will be displayed.



From the Dataspace Utility panel, you can view the current active dataspaces, select a particular dataspace to monitor or control, list the active clients for the dataspace, press the END PF key to return to the primary menu, or press the HELP PF key to view specific help for this panel and/or view the tutorial.

On the Dataspace Utility panel, for each active TABLES/MM dataspace, the following information is displayed:

ID - The two character dataspace ID from the start-up JCL.

START TIME - The day and time the dataspace was started.

DESCRIPTION - The description from the start-up JCL.

If there are more dataspaces than will fit on one screen, the standard scroll PF keys can be used. The SCROLL field can also be set to any valid ISPF value.

Monitoring and Controlling a Dataspace

To select and process a particular dataspace, enter an **"S"** to the left of the ID on the Dataspace Utility panel as shown in the example above and press ENTER. The Dataspace Directory panel shown in the next figure will be displayed.

```
DID: 00 ----------------- TABLES/MM DATASPACE DIRECTORY ------ ROW 1 TO 5 OF 5
COMMAND ===>                                                 SCROLL ===> HALF

    Table Name  ===>
    Free Space  ===>             (As a Percent, e.g. 10 = 10% free space)
    Sort Fields ===>

Memory: Total Bytes: 5242880        Directory: Total Entries: 100
        In Use     : 162809                    Entries Used : 5
        Available  : 5080071                   Available    : 95

   Table Name                    Load Time Accesses Bytes   RecCnt RecSz
   ----------------------------- --------- -------- ------- ------ -----
S  SSI001.DB2TEST                11.17.55  1        1518    8      167
   SSI001.MS_DEFINITION          11.17.56  22       12464   22     560
   SSI001.APPL1_TBL1             11.18.05  1049     4851    123    35
   SSI001.APPL1_TBL2             11.18.22  2256     42986   356    120
   SSI001.APPL1_TBL3             11.18.41  103      96336   1045   92
****************************** BOTTOM OF DATA ****************************
```

The Dataspace Directory panel is the focal point for monitoring and controlling a specific dataspace. To monitor a dataspace, the memory and directory statistics are displayed along with a list of directory entries.

For the current dataspace, the following information on its memory usage is displayed:

Total Bytes - This is the maximum number of bytes usable in the dataspace as set when the dataspace was started.

In Use - This is the number of bytes currently used by tables that have been loaded and not freed plus the space used by the dataspace control blocks and directory.

Available - This is the remaining bytes that can be used for loading new tables or re-loading tables.

Also, the current directory status for the dataspace is displayed in the following fields:

Total Entries - This is the maximum number of tables that can be loaded into the dataspace. It is set based on the DIR= parameter or control card input when the dataspace is started.

Entries Used - This is the current number of entries that are in use.

Available - This is the number of new tables that can be loaded and should always be the TOTAL ENTRIES minus the ENTRIES USED.

The scrollable list of directory entries shows all tables and views that have been or are loaded in the dataspace. For a table or view that has been freed, only the LOAD TIME (in this case, the time the FREE was issued) and ACCESSES are shown. All other values will be zero. For each directory entry, the following information is displayed:

Table Name - The name of the Table or View.

Load Time - The time when the table was last loaded.

Accesses - The total number of times the table has been accessed since the dataspace was started. The count is retained even if the table is freed.

Bytes - The number of bytes used by this table in the dataspace.

RecCnt - The number of records loaded into the dataspace.

RecSz - The size in bytes of each record within the table. For a view, this is the size of the record as returned to a program when using the API.

On the Dataspace Directory panel, a set of primary commands and a set of line commands can be used to perform specific functions for the current dataspace. The primary commands are entered in the COMMAND field and the line commands are entered on the appropriate line to the left of the table name. **Enter one primary command or one line command only**. If more than one is entered, the primary command will be executed and all line commands ignored. If more than one line command is entered, only the first one is executed. Press the ENTER key to perform the selected command.

Again, as on all panels, the END PF key will return to the previous panel, in this case the primary menu. The HELP PF key will activate the online tutorial.

Dataspace Directory Panel Primary Commands

There are three primary commands that can be entered on the Dataspace Directory panel for performing specific functions. They are:

RESET - Causes all information on the panel to be reset. That is, all fields are updated with the most current values from the dataspace.

COMPRESS - Executes the compress function for the dataspace. This will make all unusable space available again.

LOAD - Causes a table to be loaded or re-loaded into the dataspace. See below for additional fields that must be entered.

Reset Command

The RESET command causes all data on the Dataspace ID panel to be refreshed. This includes all dataspace information and directory entries. After a reset command is entered, the directory list is repositioned so that the first table displayed before the command is the first one displayed after the command.

Also, an automatic RESET is issued after a LOAD or COMPRESS primary command is executed or after a FREE line command. It is not done for a Select or Browse line command.

Compress Command

The COMPRESS command is used to compress the current dataspace. This is only needed if there is little or no available memory left in the dataspace. It will also only work if there is unused space within the dataspace. Unused space is the memory that was used by tables that were freed or reloaded into a different location. This amount can be calculated from the information on the Dataspace ID panel as follows:

Unused Space = TOTAL-BYTES - (IN-USE + AVAILABLE)

When using compress, the dataspace is locked so that no tables can be loaded or freed while the compress is in progress. Therefore, the compress should only be done when this is least likely to occur. Also, for large dataspaces, a compress can take several seconds. On TSO, the address space may be swapped out while the compress is in progress causing longer delays. This can also affect access to a table if the table is in the process of being moved when the swap occurs. Further, a TSO session may abend causing the dataspace to be left in a partially compressed state.

> ***Note:*** It is recommended that for production Dataspaces, compresses should be done using a batch job with a high priority and the maximum TIME parameter to ensure that it completes successfully and quickly

Load Command

The LOAD command is used to load tables and views into the current dataspace. On the Dataspace Directory panel, the following fields are used to load a table or view:

Table Name - Specify the name of the table or view. For a DB2 table, enter the fully qualified name including the auth-id. For a VSAM table, enter the name as defined using the batch definition utility (Note: the VSAM file must be pre-allocated using the TSO ALLOCATE command).

Free Space - Enter the percentage of free space to be used when loading the table. Free space is used to allow the table to be re-loaded at the same location if it expanded since the last time it was loaded. This is ignored for a VIEW.

Sort Fields - Enter the field names the table is to be sorted on. After each name, an 'A' for ascending or 'D' for descending can be optionally entered to specify the sort order. The default is 'A'. Always separate each value by one or more blanks.

After specifying the required information, press ENTER to load the table. A message is displayed with the results. If the load fails, you should make sure the DB2 Plan and Subsystem were correct on the Primary Menu, that you have access to them and the DB2 table, or the VSAM file is pre-allocated.

For pre-loading many tables into one or more dataspaces, it is recommended that the batch utility be used instead. Using control card input, many tables can be loaded into multiple dataspaces with less overhead and when the system is less busy.

Dataspace Directory Panel Line Commands

The line commands on the Dataspace Directory panel perform specific functions against a selected table or view in the dataspace being processed. Enter one of the following commands to the left of the table name to execute its function:

F - Free — Causes the table or view to be freed from the dataspace. The data is no longer available but the directory entry will remain.

S - Select — Selects the entry to show more detailed information. Another panel with a break down of the memory used and additional information is displayed. A sample is shown below.

B - Browse — Is used to browse the data in the table. The rows of the table are passed directly from the dataspace to the ISPF browse facility. All processing is handled by ISPF and all Browse commands and processing can be used. The data is browsed in exactly the same format as a program would get it using the API, including any re-formatting required for a view.

After completing Browse or Select, you should return to the Dataspace Directory panel. The Free command displays a message and the Dataspace Directory panel is reset and re-displayed. Additional commands can then be entered.

Table Information Panel

On the Dataspace Directory panel, selecting a table or view using the 'S' line command causes the Table Information panel to be displayed. The following is an example of the panel:

```
DID: 00 -------------  TABLES/MM Table Information ----------------------------
COMMAND ===>

Table : SSI001.DB2TEST

        Loaded at    ===> 2013-05-01-11.17.55.243742
        Loaded by    ===> SSI005
        Record Size  ===> 167
        Record Count ===> 8
        Accesses     ===> 1
        Effectivity  ===> N
        Columns      ===> 8

        Memory Usage:  Definition ===> 182
                       Indexes    ===> 0
                       Data       ===> 1336
                       Freespace  ===> 0
                                       ----
                       Total      ===> 1518
```

This panel shows more detailed information for the table and a break down of the total memory used by the table. After the table name and dataspace-id, the following fields are displayed:

Loaded At - This is the complete timestamp when the table was last loaded. It is set when the table is moved to the dataspace (ie. becomes available for use).

Loaded By - The userid of the person who loaded the table. It may also be the terminal id or job name.

Record Size - Number of bytes in each record.

Record Count - Number of records loaded.

Accesses - Total accesses to this table since originally loaded.

Effectivity - Y or N specifying if this table is effectivity controlled.

Columns - The number of columns defined for this table.

The BYTES field from the Dataspace Directory panel is the total bytes used by the table or view. The following is a breakdown of that total:

Definition - Bytes required for the table or view definition. This includes table information, statistics, and a variable section for column information.

Indexes - The bytes required for any indexes defined for the table.

Data - The number of bytes required for the data. This should always be the RECSZ * RECCNT. For a View, this will always be zero.

Free Space - Bytes reserved for free space when the table was loaded and extra space to align the definitions.

Total - Sum of the above four values.

After viewing the table information, press ENTER or the END PF key to return to the Dataspace Directory panel.

Listing Active Clients for a Dataspace

To list the active clients for a dataspace, enter an "L" to the left of the ID on the Dataspace Utility panel. The Dataspace Client panel will be shown.

When an application, either online or batch, first calls the API, a connection is made to a Dataspace. This connection remains until the job or region is terminated. The Dataspace Client panel simply shows all jobs that have connected to the Dataspace that are still active.

When stopping a Dataspace, the Dataspace Client panel can be used to check if there are any jobs or regions still connected to it. If there are, these should be checked to see if it is appropriate to stop the dataspace.

On the Dataspace Client panel, scrolling is active if required. Press ENTER or the END PF Key to return to the Dataspace Utility Panel.

Effectivity Definition

Effectivity Definition provides a capability to access multiple versions of rows within a DB2 table in which there is always a start date (called the Break-In date) and optionally an end date (called the Break-Out date). The effectivity definition is stored and used during table processing.

Any table under effectivity control must have a primary or unique index defined to DB2. The break-in date must be the last column in the index. Up to seven columns can make up the index prior to the break-in date.

To activate Effectivity Definition, enter Option 2 on the primary menu and press ENTER. A sample of the Effectivity panel is shown below.

```
---------------------- TABLES/MM - EFFECTIVITY ---------------------------
OPTION ===>

    A - ADD Effectivity Record         D - DELETE Effectivity Record
    C - CHANGE Effectivity Record      R - RETRIEVE Effectivity Data

Enter/Verify Table Name (Fully Qualified with Authorization ID):

   Table Name  ===> SSI001.DB2TEST

Enter/Verify Effectivity Control Information:

   Date Format       ===> L            (L-DB2 format, 8-USA Format)
   Break-In Column   ===> FDATE
   Break-Out Column  ===>

       Key Columns        Order            Key Columns        Order
       ------------------  -              ------------------  -
   01  FCHAR               A          05
   02  FDATE               A          06
   03                              07
   04                              08
```

Maintaining Effectivity Records

To maintain effectivity records, the functions Add, Update, Delete, and Retrieve can be specified. Enter the function code and table name for all functions. If adding or updating a record, enter the control information as well. Then press the ENTER key to perform the function.

To exit back to the primary menu, press the END PF key (PF3/15). For help information or to view the tutorial, press the HELP PF key (PF1/13). Effectivity and the panel fields are described.

When maintaining records, keep the following things in mind:

- Always enter all required fields for the specific function before pressing the ENTER key.
- After any function completes, the data in the fields remains and can be re-used for another function. Therefore, to update a particular record, use 'R' to retrieve the record first, change any required information, and update it using the 'U' function. When adding similar tables, simply change the table name and any other fields and repeat the add function.

- The KEY COLUMNS and ORDER fields are automatically entered when a table is added or updated if they are all left blank. The DB2 catalog is checked for a Primary Index or one Unique Index. If it finds one, the columns that make up the key of the index, including the break-in date, will be used as the key fields. These are then displayed on the panel if the function completes normally. If there is no primary or one unique index, then the key columns and order values must be manually entered.

Effectivity Fields

On the Effectivity Definition panel, the following fields are used to define effectivity. The table name and break-in date are required. All other fields are optional.

Table Name - The fully qualified name of the DB2 table. It can be a DB2 table name, view name or alias. When Adding or Updating a record, the name must exist in the DB2 catalog in order to validate column names and the key fields. For Delete and Retrieve, the DB2 table does not have to exist.

Date Format - This specifies the format of the break-in and break-out dates. Since DB2 allows for installation dependant formats, one of the following must be specified:

L - The format is **YYYY-MM-DD**. This is the standard DB2 default format. This is the default if not entered.

8 - The format is **MM/DD/YYYY**. This is the DB2 USA format. See Appendix-C for additional format codes.

Break-in Col - This is the name of the column in the table that contains the break-in date field. Its format must match the date format code specified or unpredictable results may occur. **It is always required.**

Break-out Col - This is the name of the column in the table that contains the break-out date field. It must have the same date format as the break-in date. This is optional and not required.

Key Columns - The key columns specify the key of the table. They are automatically filled in from the DB2 catalog if available. If entered manually, specify the column names that make up the key of the table. **The break-in date column should always be the last column of the key.**

Order - If specifying key columns, enter A for Ascending or D for Descending order of the columns. The default is A for all fields.

Also, only define Effectivity for tables that require it. That is, when the GEFF function will be used to select records from the table.

IMS/CICS Online Transaction Facility

The IMS and CICS Online Facility can be used in two modes. In full-screen mode, it works like the TSO Facility and allows a subset of the TSO functions. In line mode, it can be used for quick access and for automating TABLES/MM processing.

Full-Screen Mode

The full-screen mode allows for loading, freeing and performing API functions for tables in local memory (in the region the transaction is running in) or for tables in the Dataspace associated with the region. To display the Load Utility panel, do the following:

(1) Log on to IMS or CICS using your normal procedures.

(2) Enter the following transaction code (in IMS, enter a space after it):

TMML

(3) Then press the ENTER key to view the Load Utility panel shown below:

```
------------------------- TABLES/MM LOAD UTILITY V3.2 -----------------------
FUNCTION ===>

   Table Name  ===>

   Location of Table   ===>             (L-Local, D-Dataspace)
   Free Space: Amount  ===>
               Type    ===>             (R-Records, P-Percent)

Results:
================================================================================









================================================================================

PFK:  3/15-Exit   5/17-List   7/19-Scroll Up   8/20-Scroll Down   ENTER-Process
```

On the Load Panel, **specify a Function** to perform and press **ENTER**. A message containing the results will be displayed. Also, the **Result Area** may contain additional information depending on the function requested. The following is a description of each field and its use:

Function - Specify the requested function. Use **LIST** to display a list of tables in the dataspace or local memory. The list will appear in the Results area and can be scrolled. Use **LOAD, FREE** or any other API function (eg. GETF, CMPR, etc.) with a valid table name. This is passed to the TMMINT API and executed. The Result Code and any returned data will be displayed.

Table Name - When doing an API function the table name can be entered. It is required for most API functions. It is not used for the LIST function.

Location of Table - Specify **L** for Local or **D** for Dataspace. For the LIST function, tables will be listed either from local memory or a dataspace, not both. The default is D. For API functions, this field is passed to the interface and will be processed as described in the section about the API (refer to ICA-LOC-FLAG in the Interface Control Area).

Free Space - When doing a LOAD function, specify the AMOUNT of free space and the type. For type, specify **R** if the amount represents Records or **P** if the amount is a Percentage. For example, enter an amount of 10 and type of R to leave 10 records of free space; or enter 50 and P to leave 50% free space.

Results: - The Result area is used to display the list of tables or results from the call to the API. If a GET function is requested, any record will be displayed in the result area. For a STGF (statistics) function, the information is formatted and displayed in the results area.

On the Load Panel, the following function keys can be used:

ENTER - Executes the specified function.

PF3/15 - Terminates the online facility.

PF5/17 - Causes the list of tables to be scrolled up. This is only valid after a LIST function.

PF8/20 - Causes the list of tables to be scrolled down. This is only valid after a LIST function.

All other PF Keys are invalid and if pressed, an error message will be displayed.

Line Mode

In line mode, the TMML transaction on IMS and CICS can be used as a fast way to perform a load or free and can also be used to automate TABLES/MM functions. For example, in CICS, a sequential terminal could be setup to execute TMML in line mode to pre-load all tables required by CICS at start-up.

The following is the format of the line mode transaction :

```
TMML Function table-name  {options}
```

where: **TMML** - is the transaction name. This may be different based on your location and standards.

function - the function to perform. Valid functions are :

LOAD - load the specified table.
FREE - free the specified table.
STGF - display some statistics about the table.

table-name - the name of a table to load, free or get statistics.

options - options are used to affect how the function is performed. The following can be specified:

Local -	process the table in local memory only. Do not use the dataspace associated with the region.
Dataspace -	process the table from the dataspace. (eg. load into or free from the dataspace).
###### -	specify an amount of free space. This is only used when loading a table.
Records -	specifies that the free space amount is the number of records to allow for.
Percent -	specifies the free space amount is a percent of the total space used.

If Local or Dataspace are not specified, then the interface normally checks the dataspace first and then local memory. This however can be affected by other things. Refer to the chapter on the API for more information about how tables are loaded and freed and default values when calling the API.

The following are some examples:

TMML LOAD SSI.TEST_TABLE1 DATASPACE 50 PERCENT

- This will load a table into a dataspace and allows for 50% free space.

TMML FREE SSI.TEST_TABLE1

- This will free a table from the dataspace or local memory.

After each transaction, a one line message is displayed with the results of the function. See the Appendix on message **TMML040I** for more information. The result code and possible reason and sql codes from the function are displayed. For the STGF function, message **TMML041I** is displayed with some statistics if the table is valid.

4. BATCH UTILITIES

Batch Utilities

Overview

The TABLES/MM Batch Utilities are used for defining VSAM tables, views and indexes and for monitoring and controlling all active dataspaces. There are three main utilities, one for definition, one for controlling a dataspace and one for monitoring all dataspaces. They are as follows:

TMMDEFN - The TMMDEFN utility is used to define VSAM tables, views for DB2 and VSAM tables and indexes. It is also used to list the definitions.

TMMUTIL - The TMMUTIL utility is used to setup and execute all functions supported by the programming interface (TMMINT). This includes loading and freeing tables, compressing dataspaces, and searching and retrieving tables. It can be used for initially loading tables into dataspaces as well as for testing the calls used by an application.

TMMRPT - The TMMRPT utility is used for displaying and monitoring information about active dataspaces.

TMMDEFN Utility

TMMDEFN is a utility for defining VSAM tables, simple and complex views, and indexes. By using control card input, TMMDEFN allows tables, views and indexes to be created, replaced, deleted and listed. While DB2 tables can be processed directly from the DB2 Catalog, VSAM tables must be defined using TMMDEFN before they can be processed.

Views are logical representations of physical tables. They are not the same as DB2 Views. There are two types of views. A SIMPLE view is basically an alias of a physical table. That is, it is simply another name. It is recommended that all applications use a view name to access a table. This eliminates all dependencies on the physical table name. **A simple view is defined with no columns.**

A COMPLEX view is a restructuring of a physical table. It is defined using the names of columns in the physical table but with different format specifications. Specific columns may be included, numeric columns may be converted to different types, and dates can be converted between different formats. The complex view can be used to isolate an application from the physical table's structure. If a physical table is changed to a different access method or columns are added, removed or changed, the application can be left intact by simply adjusting the view to the new table's format.

An index, on the other hand, is only defined in relation to a physical table, not a view. The base table name must be a DB2 cataloged table or a defined VSAM table. Indexes are used to improve performance by allowing binary searches on alternate keys and to retrieve rows in different orders. The index is selected dynamically based on the search criteria specified in a GET request. No application logic is required to use an index. When an index is used, the rows are always returned in the order of the index.

After a table, view or index is defined, the TMMUTIL utility can be used to load them. A table, whether a VSAM table defined with TMMDEFN or a DB2 cataloged table, must be loaded explicitly. When the table is loaded, any indexes defined for that table are built automatically. Indexes can not be specified when using TMMUTIL. Once defined for a table, they are always built in memory when the table is loaded.

Like tables, views must be explicitly loaded. However, no data is actually loaded. For a simple view, only an entry in the dataspace directory is used. For a complex view, in addition to the directory entry, the view definition is also stored in the dataspace. In either case, the view's directory entry is setup to point to its base table's directory entry. **Therefore, the base table for a view must always be loaded first.**

Executing TMMDEFN

While TMMDEFN is a batch utility, it can be executed in batch using JCL or from TSO using supplied commands. To execute in batch, the following sample JCL can be used:

```
//         JOB
//TMM      EXEC   TMMDEFN
control cards...
/*
```

Use your standard job card and enter the TMMDEFN control cards after the EXEC statement. Then submit the job for processing. The results are printed to a SYSOUT DD card specified in the TMMDEFN JCL procedure.

To execute the TMMDEFN proc, no parameters are required. However, the following parameters can be overridden if necessary:

S= Specify the print class for the SYSOUT DD card. The default is '*' which causes the MSGCLASS on the job card to be used.

RGN= This is the region size. A value of 2M is sufficient.

TMMLOAD= Specify the name of the load library that contains the TABLES/MM load modules. This is normally set in the procedure during installation.

DB2LOAD= This is the load library that contains the DB2 modules. Again, this is usually set during installation.

DB2SSID= This is ID of the DB2 subsystem that will be used. Only one DB2 subsystem can be used during the job. The default value is set during installation.

DB2PLAN= This is the name of the DB2 plan used by TABLES/MM. The user submitting the job must have execute authority to it. The plan name must be set at installation time and should not normally be changed.

In addition to executing TMMDEFN in batch, there are two commands that can be used to execute it directly on TSO without the need to submit a job. The first command is entered in TSO READY mode or from ISPF Option 6 and is entered as follows:

%TMMDEFN DA(dataset)

where **dataset** is the name of a file that contains the control cards to be input to TMMDEFN. To enter the control cards directly from the terminal instead of a file, enter the word **TERMINAL** instead of a dataset name. The output results from TMMDEFN are displayed using ISPF Browse (if running under ISPF) otherwise they are sent to sysout file and are printed.

An even easier way to execute TMMDEFN is to use ISPF edit on a dataset that contains TMMDEFN control cards. While in edit, the following command can be used to pass the control cards directly to TMMDEFN:

#TMMDEFN

This performs the same function as %TMMDEFN except that the dataset name is automatically set from the file being edited and the output is always displayed using ISPF Browse.

Also, please be aware that when using #TMMDEFN, the dataset or member being edited is automatically saved if any changes were made. Therefore, if you do not want any changes to be permanent, a temporary file or member should be used instead.

Control Card Summary

No matter which method is used to execute TMMDEFN, the control card input always has the same format. The following specifications should be used for the control card input :

- Only columns 1 to 72 are scanned. Columns 73 to 80 are ignored.
- Input is free form (eg. parameters can start in any column).
- Parameters are separated by 1 or more blanks.
- Cards cannot be continued. All parameters must be on one card.
- Parameters shown in brackets below (eg. {...}) are optional.
- Multiple parameters shown separated by slashes means only one of the values can be entered.
- Keyword parameters (eg. TYPE(..)) must not have any blanks before or between the parentheses.
- Parameters in capital letters must be entered as is or can be abbreviated to the first 3 letters in most cases.
- Comment cards can be placed anywhere in the input by placing an asterisk (*) or double dashes (--) in column 1. These cards are ignored and not listed in the output.

The following tables summarize the control cards and possible parameters. Table 4-1 shows the primary function cards. One of these must start each function and may be followed by one or more secondary cards shown in Table 4-2.

Function	Parameters
REPLACE CREATE DROP	INDEX #nn {FROM/FOR/TO/ON} tblname {UNIQUE} {PRIMARY}
LIST	INDEX {FROM/FOR/TO/ON} tblname {NOCOLUMNS}
REPLACE CREATE DROP	VIEW viewname {FROM/FOR/TO/ON} tblname
LIST	VIEW viewname {NOCOLUMNS}
REPLACE CREATE DROP	TABLE tblname
LIST	TABLE tblname {NOCOLUMNS}
COMMIT	

Table 4-1 Primary Function Cards

Function	Parameters
COLUMN	colname {ASC/DESC}
COLUMN	colname {type(len{,decpos})} {FORMAT(F)}
COPY	ASM/COBOL mbrname {{FROM} ddname}

Table 4-2 Secondary Cards

Defining Tables

In Version 2.1 of TABLES/MM, the only tables that can be defined are VSAM tables. DB2 tables are automatically processed from the DB2 catalog. To define a table, use the CREATE statement shown below:

CREATE TABLE tblname

Where **tblname** - is the name of the VSAM table. It must be in the format **VSAM.ddname**. The 'VSAM.' is required as shown and ddname is the 1 to 8 character ddname used to read the VSAM table. When loading a VSAM table, the dataset must be allocated to the ddname specified or defined to CICS in the FCT.

After the CREATE statement, the structure of the table must be defined using COLUMN statements or a COPY statement. In Table 4-2, there are two formats for the COLUMN statement. The first is for use with indexes, and the second for tables, is shown below:

COLUMN colname {type(len{,decpos})} {FORMAT(f)}

Where **colname** - is the name of a column. It can be 1-30 characters.
type - is the type of column and can be one of the following:
CH - Character or Display Data
PA - Packed Decimal Numeric Data
BI - Binary Numeric Data
NU - Zoned Numeric Data
len - is the length of the field in bytes.
decpos - is the number of implied decimal positions. Only valid for numeric data.
f - is a format code for date fields. Specify one of the date format codes shown in Appendix-C. Only specify a date format for character columns.

Notes: The type and length are required except when using a COPY statement.

In addition to the COLUMN statement, the columns can be defined by using a COPY statement. COPY allows a PDS member containing Assembler or COBOL field definitions to be used to define the table. In addition, COLUMN statements can be entered after the COPY statement to override the field definitions (eg. to add DATE specifications). The COPY statement has the following format:

COPY ASM/COB mbrname {{FROM} ddname}

Where **ASM/COB** - is the type of copybook.
mbrname - is the member name to include.
ddname - is the ddname of the library to use to read the member from. This defaults to COPYLIB.

Notes:
- The copybook, whether Assembler or COBOL, must not have any syntax errors or unpredictable results may occur.
- The COPY statement **MUST** be the first card after the primary function card and only one is allowed per CREATE/REPLACE.
- If any COLUMN cards follow the COPY, they are used to **OVERRIDE** information in the copybook. Therefore, the column name must match a name in the copybook.

The following example is a set of control cards used to define a table:

```
--
-- CREATE A VSAM TABLE DEFINITION USING A COBOL COPYBOOK
--
        CREATE TABLE VSAM.RATETAB
         COPY  COBOL  RATETAB  FROM SYSLIB
         COL   RT-EFF-DATE     FORMAT(1)
```

Defining Views

Views are defined like tables with a few differences. A view requires a base table name. That is, the table the view is associated with. The base table must exist and must be a DB2 cataloged table or a VSAM table previously defined. In addition, views can have no columns (eg. a simple view) that is used as an alias name, or as many columns as the base table (a complex view), one used to reformat the rows in the base table. The reformat of the data only takes place when retrieving data from the view. To define a view, use the following statement:

CREATE VIEW viewname {FOR/FROM/TO/ON} tblname

Where **viewname** - is the name of the view to create. This is a 1-36 character name that will be passed to the API to retrieve rows from the table.

tblname - is the name of the base table from which the rows are actually retrieved. This must be a DB2 cataloged table or VSAM table previously defined.

If this is a simple view, then no column statements are required. If this is a complex view, column statements are required to define the structure of the view. The column statement for the view has the same format as for defining a table but work differently. Each column statement specifies a column from the base table to be part of the view. All or some of the base table columns can be defined in the view. In addition, the type and format of the column represent the specifications the column is to have in this view only. If they are different from the base table column, then reformatting will take place when a row is retrieved for this view. If they are the same, the data is moved as is. The type and format are optional and if not specified, the base table column type and format are used.

COLUMN colname {type(len{,decpos})} {FORMAT(f)}

Where **colname** - is the name of a column. The column name must exist in the base table.

type - is the type this column is to have in this view and can be one of the following:

- CH - Character or Display Data
- PA - Packed Decimal Numeric Data
- BI - Binary Numeric Data
- NU - Zoned Numeric Data

len - is the length of the field in bytes for this view.

decpos - is the number of implied decimal positions. Only valid for numeric data.

f - is the date format code for this view. If different from the base table format, the date is reformatted when the row is returned on GET requests. Specify one of the date format codes as shown in Appendix-C. Only specify a format when the base table column is also a date column.

Notes: When simply using a subset of columns, just specify the column name. All specifications are defaulted from the base table columns.

Using the table created in the previous example, we can define views using the following:

```
**
** SIMPLE VIEW
**
          CREATE  VIEW  RATE-TABLE  FOR VSAM.RATETAB
**
** COMPLEX VIEW
**
          CREATE  VIEW  RT-VIEW     FOR VSAM.RATETAB
             COL  RT-CODE
             COL  RT-RATE-AMT   PACKED(4,2)
             COL  RT-EFF-DATE   FORMAT(L)
```

Defining Indexes

Indexes are used to efficiently search through the tables in memory. By defining one or more indexes, a binary search can be used to search a table with significant benefits. If no index is defined, then all searches are sequential. This is true for DB2 or VSAM.

There are two types of indexes. A PRIMARY index and a SECONDARY index (the default). A Primary index defines the order of the rows as they are stored in memory. For a VSAM table, the primary index must be defined as the columns that make up the key field as defined for the VSAM KSDS itself. For a DB2 table, the PRIMARY index can be made up of any columns in the DB2 table and is used to order the rows when loaded into memory. A primary index uses a small amount of additional memory when the table is loaded.

A secondary index is used to define an alternate search path for the table. When retrieving rows and the search criteria specifies the columns in a secondary index, it will be used for searching and retrieving rows instead of any primary index. A secondary index is built dynamically when a table is loaded. It requires 4 bytes of memory for each row in the table and requires sorting the table each time it is loaded. Therefore, secondary indexes use additional memory and resources and should be defined with care.

Overall, while indexes are never required, it is recommended that all tables have at least a primary index for performance reasons. If a table is searched using different keys, secondary indexes may reduce CPU utilization at the expense of additional memory. Their use should be dictated by performance reasons.

To create indexes, use the following statements:

CREATE INDEX #nn {FOR/FROM/TO/ON} tblname {UNIQUE} {PRIMARY}

Where **#nn** - is the index id. The (#) is required in all cases. The nn is the index number to create. It must be a 1 or 2 digit number from 0 to 99. If not specified, the next number available is assigned. Indexes are built and used in the order of the id, not on the order they are created.

tblname - is the name of the base table to have an index. This must be a DB2 cataloged table or VSAM table previously defined.

UNIQUE - an optional keyword. If specified, it defines this index as having all unique keys. If an index will not have any duplicate entries, UNIQUE should be specified to improve performance. It should not be specified if duplicates are possible otherwise results are unpredictable.

PRIMARY - an optional keyword. If specified, it defines this index as the primary index. This means the actual rows of the table are stored in memory in the order specified by this index. For a VSAM table this must specify the fields that make up the physical key of the VSAM dataset. For a DB2 table, the rows will be ordered on the columns specified when the table is read. The first index based on the ID number with the PRIMARY keyword is the one used as the primary index.

After the CREATE statement, the columns that make up the key of the index must be specified. In Table 4-2, the first COLUMN statement is used to define the key fields. This is basically the same as specifying the columns for a table or view, but instead of type and format, just the ordering needs to be specified as shown below:

COLUMN colname {ASC/DESC}

Where **colname** - is the name of a column that is defined in the table this index is for.

ASC/DESC - specifies that the column is sorted in ASCending or DESCending order. Ascending is the default.

Currently, a maximum of 16 columns can make up the key of an index. Any type of column can be part of a key with the exception that a date column (one with a FORMAT specified) can only be of character or numeric type. That is, a date column that is PACKED or BINARY, cannot be part of the key unless it is in century-year-month-day format.

Using the table created in the previous example, we can define an index using the following:

```
--
-- PRIMARY INDEX
--
          CREATE INDEX # FOR VSAM.RATETAB PRIM UNIQ
             COL RT-CODE
```

Replacing Tables, Views and Indexes

To replace an existing entry, simply change CREATE to REPLACE. All other parameters remain the same. The replace is performed as if a DROP and then a CREATE were performed. In the case of indexes, the index ID number is required.

Dropping Tables, Views and Indexes

To delete an entry, use the DROP function for the table view or index. All parameters are the same and all can be specified. For indexes, the ID number is required and the optional parameters are ignored. For a view, the base table name is ignored if specified.

After the DROP statement, no secondary cards (eg. ones shown in Table 4-2) are allowed. The DROP will fail if any are found.

When dropping a table, only the table definition is deleted. Any views or indexes associated with the table are not deleted. Therefore, any views and indexes must be explicitly deleted using a separate DROP statement.

Listing Tables, Views and Indexes

To list entries, the LIST statement can be used for tables, views or indexes. Specify LIST followed by the object type to be listed as shown in Table 4-1. For indexes, all those defined for the selected table or tables are listed (specific ones cannot be listed).

When listing entries, the table name or view name specified can have wild card characters. The name is passed to DB2 using a LIKE function. All entries which match the name are listed. Specifying NOCOLUMNS will limit the output to the basic information. If not specified, all column information for the tables, views or indexes is listed as well.

The following examples, demonstrate LISTing different definitions:

```
-- List a table with its columns
--
        LIST  TABLE  VSAM.RATETAB
--
-- List all views defined but do not list their columns
-- (Note the use of '%' as a wildcard character)
--
        LIST  VIEWS  % NOCOLUMNS
--
-- List all indexes for a table but not its columns
--
        LIST  INDEXES  FOR VSAM.RATETAB NOC
```

Commit and Rollback Processing

If an error occurs during processing of any statements, **TMMDEFN will automatically issue a DB2 Rollback request** before terminating. As a result, all updates done during the job are removed. This is to insure that no incorrect definitions will be left in the TABLES/MM definition database (eg. the MS_DEFINITION DB2 table). In this case, simply fix the control cards that were in error, and execute TMMDEFN using all the control cards again.

This process can be over-ridden by using the **COMMIT control card**. After all secondary control cards for a function, COMMIT can be specified to cause all updates up to that point to be made permanent. **All functions prior to the COMMIT control card will be committed.**

If an error occurs in a function after a commit has been specified, only updates after the last commit will be removed. Therefore, when re-running the job, only those functions that were roll-backed should be re-entered. Functions that were committed may cause errors if re-entered again.

TMMUTIL Utility

TMMUTIL is a utility for setting up the input parameters and executing functions supported by the programming interface. By using control card input, TMMUTIL allows all functions and processing to be performed without any programming. This makes it very easy to automate setup and control of dataspaces without the need for user written programs. It can also help in application testing by simulating the calls made by a program and the results from those calls.

```
//        JOB
//TMM     EXEC   TMMUTIL
control cards...
/*
```

Like TMMDEFN, TMMUTIL can be executed in batch using JCL or from TSO using supplied commands. To execute it in batch, the following sample JCL can be used:

Use your standard job card and enter the TMMUTIL control cards after the EXEC statement. Then submit the job for processing. The results are printed to a SYSOUT DD card specified in the TMMUTIL JCL procedure.

To execute the TMMUTIL proc, no parameters are required. However, the following parameters can be overridden if necessary:

S= Specify the print class for the SYSOUT DD card. The default is '*' which causes the MSGCLASS on the job card to be used.

RGN= This is the region size. A value of 2M is sufficient.

TMMLOAD= Specify the name of the load library that contains the TABLES/MM load modules. This is normally set in the procedure during installation.

DB2LOAD= This is the load library that contains the DB2 modules. Again, this is usually set during installation.

DB2SSID= This is ID of the DB2 subsystem that will be used if any DB2 tables are loaded. Only one DB2 subsystem can be used during the job. The default value is set during installation.

DB2PLAN= This is the name of the DB2 plan used by TABLES/MM. If DB2 tables are loaded, the user submitting the job must have execute authority to it. The plan name must be set at installation time and should not normally be changed.

In addition to executing TMMUTIL in batch, there are two commands that can be used to execute it directly on TSO without the need to submit a job. The first command is entered in TSO READY mode or from ISPF Option 6 and is entered as follows:

%TMMUTIL DA(dataset)

where **dataset** is the name of a file that contains the control cards to be input to TMMUTIL. To enter the control cards directly from the terminal instead of a file, enter the word **TERMINAL** instead of a dataset name. The output results from TMMUTIL are displayed using ISPF Browse (if running under ISPF) otherwise they are sent to sysout file and are printed.

An even easier way to execute TMMUTIL is to use ISPF edit on a dataset that contains TMMUTIL control cards. While in edit, the following command can be used to pass the control cards directly to TMMUTIL:

#TMMUTIL

This performs the same function as %TMMUTIL except that the dataset name is automatically set from the file being edited and the output is always displayed using ISPF Browse.

Also, please be aware that when using #TMMUTIL, the dataset or member being edited is automatically saved if any changes were made. Therefore, if you do not want any changes to be permanent, a temporary file or member should be used instead.

Control Card Summary

No matter which method is used to execute TMMUTIL, the control card input is always the same format. The following specifications should be used for the control card input :

- All control cards start with a 1 to 4 character control code
- The first parameter must start in column 5
- Separate multiple parameters by at least 1 blank
- Blank cards are ignored
- Asterisk (*) in column 1 denotes a comment card
- Control cards cannot be continued
- Uppercase ONLY is allowed, except for values on the DATA control card
- Input record size must be 80 bytes
- Sequence numbers in columns 73 - 80 are ignored

The following table summarizes the control cards, possible parameters and a short description of each code.

Code	Parameters	Description
ABND	ON/OFF	Abend at end if any errors
DATA	Pos Type Len Value	Place data in the record
DCPU	ON/OFF	Display CPU time
DID	Dataspace-ID	Set dataspace to use
DISP	Len	Set length of output line
ECHO	ON/OFF	Display input and results
FSPC	Percent	Set free space percent
FUNC	Code Repeat-Count	Execute interface function
HEX	ON/OFF	Display record in Hex
SFLD	Col-Name A/D	Set sort field and order
SIZE	Size/SET	Set size of record to display
TABL	Table-Name	Set table name to process
WHER	Data	Where Clause

Table 4-3

Control Card Descriptions

For each control code shown in Table 4-1, a detailed description is given along with the possible parameters and values for it. The default value for each parameter is specified when appropriate.

ABND - Causes the job to abend with a U0008 abend if any errors are detected in the control card or function results. This is primarily for operations to make it easier to detect incorrect results. Possible values are:

OFF - Do not abend (the default).
ON - Abend if errors are detected.
blank - Toggle the value between on and off.

DATA - Places a data value into the record area. The record is used for certain functions like GETF to select rows with certain values. This can be used, for example, to display rows with a specific key value or to test application logic. As many data cards can be used as required to set one or all fields in the record. After a FUNC control card, the record area is always reset to blanks. The 4 parameters below are all required:

Pos - Specifies the starting position in the record to place the value (the first position is 1).

Type - The type of data to place in the record as follows:

C - Character data, value is moved as is.
P - Packed Decimal, value must be a number and is converted to a packed field.
B - Binary, value must be a number and is converted to binary.

Len - The length of a character field to move or the size of packed or binary field to create. For character fields, it should be less than or equal to the field size and for packed and binary, it should always be equal to the field size.

Value - The data to place in the record. The first non-blank character after the length is the first character moved or converted. For packed and binary values, the data must be a valid number but should not include a decimal point.

DCPU - Displays the CPU time used in microseconds after each FUNC request. The default is off.

ON - Turns CPU display on.
OFF - Turns CPU display off.
blank - Toggles the value between on and off.

DID - Specifies which dataspace to process. If not specified at all, the default dataspace (ie. ID=00) is used. If a TBLDID?? DD card is specified in the JCL or allocated on TSO, the ID from it (the ??) is used as the dataspace to process and cannot be overridden with the DID control card.

Dataspace-ID - The 2-character ID of an active dataspace to process or blanks. If blanks are specified, then only local memory is used and all dataspace processing is bypassed (useful for testing calls when no dataspaces are active).

DISP - Sets the size of the output display line when displaying table records as a result of retrieval functions. The default size is 75.

Len - The size of the output display line.

ECHO - Turns the echoing of display cards and results either on or off. The default is on. Error results will always be displayed. Valid values are:

ON - Turns echo on.
OFF - Turns echo off.
blank - Toggles the value between on and off.

FSPC - Sets the free space percent value passed to the interface. This is only used when loading a table. A value of 10 for a table of 100 records would cause space to be allocated in the dataspace for 110 records. If set, then the value is passed to the interface for all load functions until changed. A value of blank or zero is valid. The default is zero.

Percent - The amount of free space to leave as a percentage of total records.

FUNC - Causes the programming interface to be executed for the specified function code using values previously set. Parameters are as follows:

Code - The programming interface function to execute (eg. LOAD, GETF, GETN, GETS, FREE, ETC). This value is required as there is no default.

Repeat count - A numeric value specifying how many times to repeat the function. For example, to display 10 records, specify 'GETS 10'. The Get Sequential function is executed 10 times or until the END of data is reached. The repeat count defaults to 1.

HEX - Set to display records in HEX mode. Can be set to the following:

OFF - Do not display hex (the default).
ON - Displays the records in hex mode.
blank - Toggle hex mode between on and off.

SFLD - Specify a column the table is to be sorted on. Enter only one column name per card but use as many cards as necessary. Enter the cards in order of importance (major sort field first, etc.). The sort fields are only used with the table load function. The default is no sort fields. Also, after any FUNC card is processed, the sort field array is set to null. Therefore, they should only be specified immediately before the "FUNC LOAD" card for the table they pertain to. Parameters are:

Column- Name of a column in the table to sort on. It is required.
A/D - Enter **A** for ascending or **D** for descending. The default is **A**.

SIZE - Sets the size of the record area to display for any record returned from a FUNC control card. Specifying a small value can be used to display just the first part of a record. The default is to display the first 75 bytes of the record area. If this value is larger the DISP value, the record is printed on multiple lines.

Size - Specify the size of the record to display or enter 'SET' to have the size automatically set to the record size of the table being processed.

TABL - Specify the table name to process. This must be the complete DB2 table name (eg. Include the AUTH-ID). The table name is required for all functions.

Name - The fully qualified DB2 table name. This can be the table name, a view name or DB2 alias name.

WHER - Specify a where clause to be used **only with the LODW** function. It must start with '**WHERE**' and the data must be in columns **6 through 72**. Enter as many statements as necessary to create the where clause. The data from each WHER statement is appended to the data of any previous one. Use **WHER with no data to clear** the where statement before a second LODW request. An invalid where clause will cause a **TBLINVLD** result code on the LODW request with an appropriate reason and SQL code.

Data - The where clause SQL to use with the LODW request to selectively load rows from a DB2 table. It must be valid SQL, can **only be a where clause** and must be correct for the table being loaded.

Examples

1. This example simply loads three user tables into Dataspace U1. The third table is loaded with only selected records.

```
DID  U1
TABL USER01.TABLE 0001A
FUNC LOAD
TABL USER01.TABLE 0002A
FUNC LOAD
TABL USER01.TABLE_0003A
WHER WHERE DIV = '01' AND
WHER       AND ACCT = 'GL'
FUNC LODW
```

2. This example displays all employee table records with a value of 'SMITH' in the last name field (that starts in column 21). The GETF function gets the first occurrence and the GETN gets up to the next 99 occurrences or until no more are found. Dataspace with ID=00 is searched unless a TBLDIDxx card is found in the JCL. Display is turned off except for the records (ECHO ON/OFF) and the first 100 bytes of each record are displayed (SIZE 100). The records are displayed in hex and EBCDIC (HEX ON).

```
ECHO OFF
TABL APPL.EMPLOYEE TABLE
DATA 21 C 5 SMITH
SIZE 100
HEX  ON
ECHO ON
FUNC GETF
FUNC GETN 99
```

3. This example loads two tables into different dataspaces and displays the statistics for them.

```
DID  1A
TABL RATE.CODE_RATE_001
FUNC LOAD
FUNC STGF
DID  1B
TABL RPT.TITLE TABLE
FUNC LOAD
FUNC STGF
```

Sample Output

The following is a sample of the output from TMMUTIL for the first example above. For each card read, it is displayed with the results. For the FUNC control card, the record area is also displayed if any record is returned.

```
TABLES/MM INTERFACE UTILITY - STARTED
================================================================================

--------------------------------------------------------------------------------
CONTROL CARD: DID  U1
--------------------------------------------------------------------------------
DATASPACE ID CHANGED: U1

--------------------------------------------------------------------------------
CONTROL CARD: TABL USER01.TABLE 0001A
--------------------------------------------------------------------------------
TABLE NAME TO PROCESS: USER01.TABLE 0001A
TABLE TYPE IS SET TO : DB2

--------------------------------------------------------------------------------
CONTROL CARD: FUNC LOAD
--------------------------------------------------------------------------------
TMM INTERFACE RESULT = OK                REPEAT COUNT:00001
RECORD AREA:  *** IS ALL SPACES ***

!
!
!

--------------------------------------------------------------------------------
CONTROL CARD: TABL USER01.TABLE 0003A
--------------------------------------------------------------------------------
TABLE NAME TO PROCESS: USER01.TABLE 0003A
TABLE TYPE IS SET TO : DB2

--------------------------------------------------------------------------------
CONTROL CARD: FUNC LOAD
--------------------------------------------------------------------------------
TMM INTERFACE RESULT = OK                REPEAT COUNT:00001
RECORD AREA:  *** IS ALL SPACES ***


================================================================================
TABLES/MM INTERFACE UTILITY - ENDED, RC = 00000
================================================================================
```

TMMRPT Utility

TMMRPT is a batch utility to generate a report on all currently active dataspaces. It will list all active dataspaces with their memory and directory usage. Also, all used directory entries are listed showing the tables and associated information. TMMRPT is very useful to run on a nightly basis or immediately before all dataspaces are stopped to generate a listing of statistics showing the use and accesses to all dataspaces. This can be used to gauge the benefits of using TABLES/MM and also to monitor and control tables that may not be required or beneficial to keep in memory.

To execute TMMRPT, the following JCL can be used:

```
//         JOB
//TMM      EXEC    TMMRPT
```

The output from TMMRPT is written to the SYSOUT DD card. As for TMMUTIL, the S= parameter can be specified to override the sysout class. The default is '*' which means the MSGCLASS value from the job card is used. The following is a sample of the output:

```
DATE: May 1, 2013                TABLES MEMORY MANAGER                          PAGE   1
DID:  00                            DATASPACE REPORT


================================================================================================
| DESCRIPTION: APPLICATION DATASPACE                    STARTED: 2013-05-01-12.31.44 |
================================================================================================
|                                                                                              |
|     MEMORY: TOTAL BYTES =  2097152           DIRECTORY: TOTAL ENTRIES =   50                 |
|             IN USE      =   550132                      ENTRIES USED  =    2                 |
|             AVAILABLE   =  1547020                      AVAILABLE     =   48                 |
|                                                                                              |
================================================================================================

DIRECTORY ENTRY LISTING:

TABLE NAME       TOT-BYTES ACCESSES LOAD TIMESTAMP        LOAD-BY RECSZ REC-CNT COLS EFF
---------------- --------- -------- -------------------- ------- ----- ------- ---- ---
SSI001 DB2TEST       21432       99 2013-05-01-14.44.01   SSI005    123      21    8 YES
SSI005 ANPGM1 T1    520100      201 2013-05-01-15.03.22   SSI001    100    5201   11 NO


================================================
DATASPACE SUMMARY STATISTICS :
   ACTIVE TABLES        =>       2
   ACTIVE VIEWS         =>       0
   FREED TABLES & VIEWS =>       0
   TOTAL ACCESSES       =>     300
   RECOVERABLE BYTES    =>       0
================================================
```

The same report is generated for each active dataspace. The Dataspace-ID (DID) is printed in the heading on all pages.

5. APPLICATION PROGRAMMING INTERFACE

Application Programming Interface

Overview

The TABLES/MM Application Programming Interface or API, is a very easy to use, high performance interface for executing all functions supported by The TABLES Memory Manager. Its main functions are used for retrieving data from dataspaces and local memory based on the application's requirement. Records can be selectively retrieved based on one or more fields, the table can be scanned both forward and backwards, and records retrieved based on effective dates. In addition, the dataspace can be controlled by an application by loading, freeing, and monitoring tables within a dataspace.

The API is executed by using a standard subroutine call to the interface module with one or more parameters. For all calls, there is one required parameter and several optional parameters. The Interface Control Area (or ICA) is required and is used to pass processing requirements to the API and to return status results back to the application. In addition to the ICA, a Record I/O area, Interface Sort Area (ISA), and other optional parameters may be required. These are described below for each of the function codes.

Dataspace and Local Memory Access

The API, by default, tries to load and access tables from a dataspace with ID=00. This allows any application running anywhere to simply call the API to load and access tables without the need for special JCL or programming. To facilitate access to a different dataspace and/or local memory, this default can be overridden. Using a special DD card in JCL, the dataspace ID can be set so that the API will always access a specific dataspace rather than the default. It can also be set so that the API will not access any dataspace and just use local memory to load and access tables. Further, by setting a flag in the ICA when calling the API, dataspace and local memory access can be inter-mixed. This allows an application to programmatically decide to use local memory for all or some tables and a dataspace for others. **An application cannot however, change or set the dataspace id internally**. This can only be done through JCL.

To specify where an application's tables will be loaded or accessed from, the API uses the following in determining whether to access local memory, a specific dataspace or the default dataspace:

(1) If the following DD card is included in the JCL for the online region, batch job or any other task:

//TBLDID DD DUMMY

then all dataspace access is bypassed. All tables will be loaded and accessed from local memory only. This can be used by batch applications that require direct access to tables in local memory or for testing applications without affecting production dataspaces.

> ***Note:*** The API will always try to connect to a dataspace if this DD card is not included and will fail if none are active. Therefore, always specify it when no dataspaces are active.

(2) If the **LOC-FLAG field in the ICA is set to 'L'**, specifying that local memory is to be used, then no dataspace is accessed. Whatever the function, the specific table is only processed using local memory. This flag can be dynamically set for different tables within the same program to access some from local memory and some from a dataspace. For a specific table, it should always be set to access the table either from local memory or the dataspace, but not both.

(3) If the LOC-FLAG is not set, and a TBLDID card with a dataspace id is found in the JCL, the API always tries to access or load the table into the dataspace specified. For example, if the following DD card was found in the JCL:

//TBLDID99 DD DUMMY

then all table processing would be done from dataspace with ID=99. This allows a particular region (eg. test region) or a batch application to use a dataspace specifically setup for it. The dataspace with the id must be active before it can be used.

(4) If the LOC-FLAG is not set and no TBLDID card is found in the JCL, the API will then try to use the default dataspace (ID=00).

To summarize, the API can be set so that either a dataspace and/or local memory can be used for tables. The control can be done externally using the TBLDID card in JCL or internally, by setting the LOC-FLAG in the Interface Control Area passed to the API. This allows the TABLES/MM API to be easily and flexibly controlled for both online and batch applications.

Auto-Load Mechanism

When tables are processed from local memory, either explicitly or by default, the API will automatically load the table into the local memory the first time the table is processed. This allows an application to be coded independent of where the tables will be accessed from and eliminates the need to explicitly request a table to be loaded when only local memory is used. A table is automatically loaded into local memory based on the following:

- The table is to be processed from local memory because one of the following :
  - The LOC-FLAG in the ICA was set to 'L'
  - The TBLDID DD card was specified with no id, or
  - The dataspace connected to was created with a limited set of authorized tables and the table being processed was not one of them.

- The function specified is a valid GET request that can be issued as the first request. This is true for all GET functions except GET NEXT (GETN) or GET PREVIOUS (GETP).

- And the table was not previously loaded.

If each of them is true, then the table will be automatically loaded into local memory. This can be especially useful in testing applications without the need for dataspaces. Using the TBLDID DD card to bypass dataspace processing, tables can be processed just as if they were pre-loaded into a dataspace. This is true for batch applications as well as online transactions.

The auto-load mechanism is never used with dataspaces. All tables to be accessed from a dataspace must be pre-loaded using the online or batch facilities or explicitly loaded by the application.

Bypassing DB2 Usage

When loading any table, TABLES/MM normally looks first in the MS_DEFINITION DB2 table for information about the requested table. However, if DB2 is not available, then an error will occur and the load request will fail. This is true even when loading VSAM or TABLES/MS tables.

For VSAM tables, the table MUST be defined to TABLES/MM in the MS_DEFINITION DB2 table or the

TBLMSDEF VSAM file. Refer to the Chapter on the VSAM COMPONENT for more information about bypassing DB2 when loading VSAM files without DB2.

To load TABLES/MS tables where DB2 is not available, use the following DD card in the JCL or online region where the load will take place:

//TBLMSDEF DD DUMMY

All DB2 processing is bypassed and only the TABLES/MS databases are used. This also means that ONLY TABLES/MS tables can be loaded by the job or region. Refer to the Appendix on TABLES/MS for additional information about using it with the Memory Manager.

Calling the API

The API can be called from any language or system that supports the standard OS call structure. The following are the basic requirements when calling it:

- Any addressing mode (Amode) can be used (24 or 31) by the calling program. The API always uses 31-bit addressing and switches to and from it as required.
- The calling program can use any residency mode (Rmode). The API itself is setup initially with Rmode=24. This allows programs running in 24-bit mode to call it along with programs running in 31-bit mode.
- Standard subroutine linkage is required with the variable parameter bit set (eg. COBOL CALL works fine).
- The interface module, TMMINT, must be called dynamically, it can not be linked statically into the calling program.

The following example shows the basic call to the API:

```
CALL  TMM-INT-MODULE USING   INTERFACE-CONTROL-AREA,
                             INTERFACE-RECORD-IO-AREA,
                             INTERFACE-SORT-AREA.
```

In the example, an ICA, a record I/O area and an ISA is passed. The ICA is required and the other parameters are optional depending on the function code. TMM-INT-MODULE is a variable defined in working storage as an 8-byte character field with a value of 'TMMINT'. In COBOL, this ensures a dynamic call. Each of the parameters is described in detail below.

> ***Note:*** See the sample programs in the installation SOURCE library for a more detailed example of setting up and calling the API. Also, the SOURCE library has sample COBOL layouts for the different interface areas.

Interface Control Area

The ICA is the main parameter used to pass control information to the API and for the API to return the status of the results to the application. The following is the structure (shown as a COBOL layout) of the ICA:

```
01    INTERFACE-CONTROL-AREA.
      05    ICA-RESERVED                  PIC X(08).
      05    ICA-TABLE-NAME                PIC X(36).
      05    ICA-PROGRAM-NAME              PIC X(08).
      05    ICA-FUNCTION-CODE             PIC X(04).
      05    ICA-RESULT-CODE               PIC X(08).
      05    ICA-SORT-FLAG                 PIC X(01).
      05    ICA-REASON-CODE               PIC S9(8) COMP.
      05    FILLER           REDEFINES    ICA-REASON-CODE.
            10    ICA-SQLCODE             PIC S9(4) COMP.
            10    ICA-RSNCODE             PIC S9(4) COMP.
      05    ICA-LOC-FLAG                  PIC X(01).
      05    ICA-FREESPACE-TYPE            PIC X(01).
      05    ICA-FREESPACE-AMOUNT          PIC S9(8) COMP.
      05    ICA-FILLER                    PIC X(125).

Note: All fields not specifically set by the application should be
      initialized to spaces or low-values to insure
      consistent results and compatibility with future releases.
```

Each field in the ICA is important and should be set appropriately before calling the API. Please review the description of each field below carefully:

RESERVED - The reserved field is used by the API to maintain control information. This field MUST be initialized to spaces or low-values the first time the API is called from a program. For online transactions, this area should be initialized every time the transaction executes. **If multiple ICA's are used, this field MUST be copied from the first one used to all others, before they are passed to the API the first time.**

TABLE-NAME - The name of the table to process. Specify the fully qualified DB2 table name, the name of a defined VSAM table or a defined View.

PROGRAM-NAME - This should be set to the name of the program that is calling the API. It is used for monitoring and debugging and should be set correctly. **It must be NON-BLANK.**

FUNCTION-CODE - This is the API function code to execute. It is a 1 to 4 character value and is required. The function codes are described in detail below.

RESULT-CODE - A 1 to 8 character result returned to the application by the API. OK and END are valid, all others are invalid. Refer to Appendix B for a description of all possible result codes.

SORT-FLAG - This is a flag that denotes whether the SORT-AREA is being passed to the API. Set to 'Y' if a sort area parameter is specified on the call statement. Any other value will mean no sort area is passed.

REASON-CODE - The reason code contains additional error information for certain result codes when loading tables. If the value is negative, then ICA-SQLCODE contains a DB2 sqlcode or an internally generated code and ICA-RSNCODE will contain the reason code. For positive values, ICA-REASON-CODE and ICA-RSNCODE are the same and represent the reason code. See **Appendix E** for a list of all codes and what they mean.

LOC-FLAG - The location flag is used to specify where a table is to be processed from. Leave blank for normal default processing (e.g. a dataspace will be used if available or local memory if not). Valid options are:

- **"L"** – Local mode. Only memory local to the job is used. The tables are not shareable. That is, for a LOAD function, load the table in local memory or for a GET function, only search a local table.
- **"D"** – Dataspace mode. Tables will only be loaded into or searched from a dataspace.
- **"B"** – Special Bypass mode. This is only valid on the FIRST call to TMMINT for the job or process. It will cause bypassing of dataspace usage without the need for the TBLDID DD DUMMY card. Effectively implements Local Mode for all requests.

FREESPACE-TYPE - Set to the percent character (%) to denote that the free space value is to be used. This must be set for the LOAD function, otherwise, the free space value is ignored.

FREESPACE-AMT - The free space value is used only when loading a table into a dataspace. It is used to allow the table to expand and still be loaded into the same memory location within the dataspace. Specify the percent of free space to allocate. For example, if a table has 100 records, and a value of 10 is specified, space for 110 records is allocated in the dataspace (eg. 10% extra).

FILLER - This is a filler area to allow for future expansion of the ICA. It should always be specified to insure older applications will continue to work with future releases. Also, to insure compatibility, it should be initialized to spaces or low-values.

Interface Record I/O Area

The Record I/O Area is used for both input to the API and output from the API. For all **GET requests** except GETQ, the I/O area is used to pass the search values. Specific fields can be set within the I/O area that will be used to match against the fields in the table. Only records where all fields are equal will be returned. Any field in the I/O area not used, **MUST be set to blanks**, except when GETE or GETQ is used. This is true for character and numeric fields. Blanks signify that the field is not used as search criteria. For all GET requests, any record found is returned in the I/O area.

The size of the I/O area must be large enough to contain the full record of any table it's used with. For example, if an I/O area is used to retrieve a record from three tables with record sizes of 100, 200, and 300 bytes, then it must be at least 300 bytes long. For a statistics request, the I/O area must be at least as large as the Statistics Information Block (100 bytes as shown in Figure 5-5).

When processing a complex view (eg. reformatting required), the I/O area only has to be large enough to contain the record as defined for the view, not the base table.

For **statistics functions**, the results are also returned here. Refer to the Statistics Information Block for a layout of the fields returned.

On a **LODW request**, the Record I/O Area must contain a **two-byte binary length** field followed by **the where clause** used to select rows from the DB2 table. The length field must be set to the length of the where clause (not including the length itself) and can be a maximum value of 1000.

Interface Sort Area

The Interface Sort Area (ISA) is no longer required and should only be used in very special circumstances. It is used to specify the sort order when loading DB2 tables into memory from a program. **However, it is recommended that a primary index be defined instead.** Using the TMMDEFN utility, a primary index can be defined which has the same effect. It also eliminates the need to specify the ISA in all programs that load the table and ensures that the table is always loaded in the correct order.

If used, the sort order defined in the ISA becomes the primary index for the table when loaded into memory. Any primary index defined for the table will become a secondary index.

The ISA is used to specify **the column and order** the rows of the table are to be sorted on when loading a table. In addition, the SORT-FLAG field in the ICA must be set to 'Y'. **The ISA should never be passed if the ICA-SORT-FLAG is not set to 'Y'.** The following is the layout of the sort area:

```
01   INTERFACE-SORT-AREA.
     05    ISA-FIELD-COUNT          PIC 9(4) COMP.
     05    ISA-FIELD-ARRAY          OCCURS 1 TO 50 TIMES.
           10    ISA-FIELD-NAME     PIC X(32).
           10    ISA-ORDER-BY       PIC X(01).
```

Within the Sort Area, the following fields are defined:

FIELD-COUNT - The count of the number of fields passed in the field array. This value must be set to the correct number of entries used.

FIELD-NAME - This is the name of a column in the DB2 table to sort on. The fields must be specified in major to minor order. That is the main sort field should be the first entry and so on.

ORDER-BY - Specify 'A' for ascending or 'D' for descending order. The default is 'A'.

The FIELD-ARRAY can contain any number of entries but must contain at least as many as specified by the FIELD-COUNT value. Any entries in the array not used are ignored.

Function Codes

The API supports function codes to maintain and control dataspaces and local memory usage as well as those for searching and retrieving data. The following table summarizes the function codes that can be used:

Function	Description
CMPR	Compress a dataspace
FREE	Free a table from a dataspace or local memory
GEFF	Get Effective row based on break-in, break-out dates
GETE	Get Equal based on search field parameters
GETF	Get First record based on search criteria
GETG	Get Greater than or Equal based on the primary key
GETL	Get Less than or Equal based on the primary key
GETN	Get Next record with specified search criteria
GETP	Get Previous record with specified search criteria
GETQ	Get Qualified based on search parameters
GETR	Get Reverse, the previous record in the table
GETS	Get Sequential, the next sequential record in the table
LOAD	Load a table into a dataspace or local memory
LODW	Load a DB2 table using the specified Where clause.
STGF	Statistics - Get First table
STGN	Statistics - Get Next table

Figure 5-4: API Function Codes

Function Code Descriptions

Each function code is described below with some specific result codes for the function. Common results codes that can occur for many functions are described after them and all result codes are listed in **Appendix B**.

CMPR - Use to compress a dataspace. This is only required when a load fails due to not enough room in the dataspace. It is recommended that this only be executed using a batch job with high priority. If not, and an abend occurs, the dataspace can be left in a partially compressed state.

Result Codes: ERRCMPR - A compress is already in progress for the specified dataspace.

FREE - Use to free a table from a dataspace or local memory. In a dataspace, the memory is no longer usable until a compress is done. A freemain is done for a local memory table releasing and freeing it for other uses.

Result Codes:	OK -	Table was freed ok.

GEFF - GET EFFECTIVE sets the table position to and returns the first record which matches the fields in the I/O area and falls within the break-in and break-out dates. If the I/O area is blank, the first effective record in the table is returned.

Result Codes:	OK -	An effective record was found and returned.
	END -	End of table, no effective records matching the criteria were found.
	EFFERROR -	The table is not defined with effective dates.

GETE - GET EQUAL sets the table position to the first record which matches those fields passed as parameters and returns the complete record in the I/O area. The fields passed as parameters must be in the I/O area and point to the first column of the field. At least one field must be passed as a parameter after the I/O area. See the description and example of GETE later in this section.

Result Codes:	OK -	A record was found and returned.
	END -	End of table, no records matching the criteria were found.

GETF - GET FIRST sets the table position to the first record which matches the fields in the I/O area and returns the complete record in the I/O area. If the I/O area is all blanks, then the first record in the table is returned.

Result Codes:	OK -	A record was found and returned.
	END -	End of table, no records matching the criteria were found.

GETG - GET GREATER searches the table using the first field in the primary index for records with a value greater than or equal to the value specified and any other search criteria that matches. If no primary index was defined at the time the table was loaded, only one field can be placed in the I/O area. The one field is then used to compare against the table for any records with a greater than or equal value. If a record is found that matches, position is set and the record returned.

Result Codes:	OK -	A record matched ok and was returned.
	END -	No record matched the criteria.
	NOTOK -	More than one search field specified and no sort fields exist.

GETL - GET LESS is performed similar to GETG except the search is based on the field being Less than or Equal. Also, instead of the first record in sequence being returned, the last record in sequence is returned. In this case, the GETP will correctly return additional records that match the search criteria.

Result Codes:	OK -	A record matched ok and was returned.
	END -	No record matched the criteria.
	NOTOK -	More than one search field specified and no sort

fields exist.

GETN - GET NEXT gets the next record that matches the search criteria specified on the previous GETF or GETG function. It must only follow a GETE, GETF, GETG or GETQ or an error occurs.

Result Codes:		
	OK -	Another record was found ok that matched the criteria.
	END -	End of table, no more records matching the criteria were found.
	NOTOK -	Table position not set by a previous call

GETP - GET PREVIOUS gets the previous record that matches the criteria specified on the preceding GETF or GETL function. It must only follow a GETL or a GETN that followed a GETF (eg. get records that were previously retrieved).

Result Codes:		
	OK -	Another record was found ok that matched the criteria.
	END -	Start of table was hit before any more matching records were found.
	NOTOK -	Table position not set by a previous call

GETQ - GET QUALIFIED sets the table position to the first record which matches the qualifications of the search parameters and returns the complete record in the I/O area. **The I/O area MUST not contain the search values.** All search values should be separate fields and must not be changed if a GETN request is issued after the GETQ. At least one qualification must be passed as a parameter after the I/O area. See the description and example of GETQ later in this section.

Result Codes:		
	OK -	A record was found and returned.
	END -	End of table, no records matching the criteria were found.

GETR - GET REVERSE is used to get records in reverse order without any regard to search criteria. Based on the current record pointer, the preceding record in the table is returned. The last record in the table is returned if the record pointer was not set by any other function. Any function can be used before GETR (eg. GETF) to position the record pointer to a specific record in the table.

Result Codes:		
	OK -	The previous record was returned ok.
	END -	No more records, the start of the table was found.

GETS - GET SEQUENTIAL is used to get the next sequential record in the table. It is the same as GETR except it works in a forward direction. The first record is returned if the record pointer was not previously set.

Result Codes:		
	OK -	The next record was returned ok.
	END -	The end of table was hit.

LOAD - LOAD a table into a dataspace or local memory depending on the flags in the ICA and any TBLDID cards in the JCL. If the table is already loaded, it will be replaced with the latest records. A table with no records is also valid.

Result Codes:	OK -	The table was loaded ok.
	TBLINVLD -	The load failed because the table was not found or could not be accessed.

LODW - LODW is used to **load selective rows of a DB2 table**. It is the same as LOAD except the Record I/O Area is used to pass a SQL Where clause. The Record I/O Area must contain a two byte binary length followed by the where clause. The clause must start with 'WHERE' and must be valid SQL syntax. The length must be set to the size of the where clause passed in the I/O area (DO NOT count the length field as part of the size) with a maximum value of 1000.

Result Codes:	OK -	The table was loaded ok.
	TBLINVLD -	The load failed because the table was not found, could not be accessed or an invalid where clause was passed.

STGF - STATISTICS GET FIRST function is used to request statistical information for a table. This can be used to check if a table is loaded, to directly access a table in local memory or just to monitor a tables access. The statistics is returned in the I/O area. See below for more information on the statistics information block.

Result Codes:	OK -	Statistics was returned ok.
	END -	No tables loaded.

STGN - STATISTICS GET NEXT function is used to request statistics for the next table in the dataspace or local memory. This is a continuation of the STGF call, which must precede it.

Result Codes:	OK -	Statistics was returned ok.
	END -	No more tables in the directory.

Return, Result and Reason Codes

The **Return Code** from the API is always **ZERO**. For example, in COBOL the RETURN-CODE special register will always be zero after calling the API. Therefore, the ICA-RESULT-CODE should be checked after each call to test for any errors.

The **Result Code** field in the ICA is set by the API for all requests to specify whether the function executed correctly. A result of **'OK'** means the function worked. A result of **'END'** means the GET function worked but no record was found. Any other result means the function did not work and an appropriate action should be taken. See **Appendix B** for a list of all Result Codes.

The **Reason Code** contains additional information when an error occurs during table load processing. It specifies the reason of the failure. If the value is negative, then ICA-SQLCODE or the first two bytes of the reason code will contain the DB2 sqlcode or an internally generated negative value. All codes from -9000 to -9999 is set internally when using the VSAM access method. In batch processing, the two right-most digits represent the File-Status code as set by COBOL. In CICS, the two right-most digits represent the CICS EIBRESP value. See **Appendix E** for a description of all reason codes and actions to take.

Statistics Information Block

For the STGF and STGN function calls, the statistics is returned in the I/O area using a format specified by the Statistics Information Block (SIB). The information is the same for a table in a dataspace or local memory except for the SIB-ADDRESS field. The layout of the SIB is shown in Figure 5-5.

```
01   STATISTICS-INFORMATION-BLOCK.
     05  SIB-TABLE-NAME            PIC X(36).
     05  SIB-RECORD-CNT            PIC 9(9) COMP.
     05  SIB-TABLE-SIZE            PIC 9(9) COMP.
     05  SIB-ACCESSES              PIC 9(9) COMP.
     05  SIB-TIMESTAMP             PIC X(8).
     05  SIB-USERID                PIC X(8).
     05  SIB-RECORD-SIZE           PIC 9(4) COMP.
     05  SIB-TABLE-ADDRESS         PIC X(4).
     05  SIB-COLUMNS               PIC 9(4) COMP.
     05  SIB-FILLER                PIC X(28).
```

Figure 5-5.

The following are the fields in the SIB:

SIB-TABLE-NAME-	The name of the table the statistics are for.
SIB-RECORD-CNT-	The number of records loaded for this table.
SIB-TABLE-SIZE -	The total number of bytes used for this table including definition and data.
SIB-ACCESSES -	The total number of times this table was accessed by a GET request.
SIB-TIMESTAMP -	The timestamp when the table was loaded. This is the 8-byte binary system clock value.
SIB-USERID -	The userid of the job or transaction that last loaded the table.
SIB-RECORD-SIZE-	The length of each record in the table.
SIB-ADDRESS -	For tables in local memory, this is the address of the first record. This can be used to directly reference a table. For a table in a dataspace, it is set to zero.
SIB-COLUMNS -	The number of columns defined in the table.
SIB-FILLER -	Reserved area for future use.

Retrieving Records

In TABLES/MM, there are several different functions for retrieving records from a table. Some are used to retrieve the first record based on some criteria, some to retrieve additional records, and others may be used for both. In addition, records can be retrieved in a forward or backward direction.

The first GET request to a table must be one of the primary functions. All GET functions except GETN and GETP are primary functions. These functions set the initial position within the table. To retrieve additional records after the first request, a secondary function must be used. These include GETN, GETP, GETS and GETR. The GETN/GETP functions retrieve the next record based on any search criteria from the primary function. The GETS/GETR functions simply return the next sequential or previous record in the table without regard to the search criteria.

Which Function to Use

The application requirements should generally dictate which function to use. However, different functions can be used to accomplish the same task. Therefore, to make the most effective and efficient use of TABLES/MM, the guidelines below should be followed:

- If retrieving records with no search criteria, eg. scanning from the first record to some point or the end, use GETS or GETR. GETF with the I/O area all blanks or GETQ with certain criteria can also be used to start the search but are less efficient.
- If searching for specific records with some criteria, use GETQ. It is the most efficient and flexible GET function and should be used instead of GETF, GETE, GETG or GETL.

- If GETQ is not used for some reason, and searching on just a few fields all with equal comparisons, then GETE should be used. With GETE, the search fields, which must be in the I/O area, are specified on the call to the API. This makes it more efficient then GETF as all fields in the I/O area do not have to be checked.

- If GETQ or GETE are not used, and searching on fields with equal comparisons, then use GETF. However, all fields in the I/O area not used as search fields must be set to blanks, including numeric and character fields. **For binary fields, this means values of X'40404040' will not be used in the search.**

- The GETL and GETG functions have limited use and only allow limited searching. It is recommended that GETQ be used instead for efficiency and because the GETQ search criteria are more explicit and lead to less confusion.

GETE Function

The GETE function works exactly the same way as GETF except in the way the search fields are defined. With GETF, all non-blank fields in the I/O area are used. This leads to problems because all other fields, including numeric fields, have to be set to blanks, which can cause problems with packed decimal and binary fields. With GETE, only the search fields specified on the call statement are used. This means that only those fields in the I/O area must be set. Other fields are not used, eliminating the need to clear them to blanks.

The search fields for GETE must be specified on the call to TMMINT. Each search field must be part of the I/O area specified on the call and must be the first position of a field. When a table is loaded into memory, the offset of each field is known. For each field specified on the GETE request, the offset of it from the start of the I/O area must match an offset as defined for a field in the table. If we have the following layout for a table:

```
01    SAMPLE-TABLE.
      05  ST-NAME        PIC X(36).
      05  ST-ACCT        PIC X(8).
      05  ST-DIV         PIC X(4).
      05  ST-SALARY      PIC S9(6)V99 PACKED-DECIMAL.
```

Then to search for all names with ACCT='COMPSRVC' and DIV='10', the following could be used:

```
MOVE 'COMPSRVC' TO ST-ACCT.
MOVE '10'       TO ST-DIV.
MOVE 'GETE'     TO ICA-FUNCTION-CODE.
MOVE 'SAMPTBL'  TO ICA-TABLE-NAME.

CALL TMMINT USING INTERFACE-CONTROL-AREA,
                  SAMPLE-TABLE,
                  ST-ACCT,
                  ST-DIV.
```

In the example code, the ICA is passed first as always, SAMPLE-TABLE is used as the I/O area to pass search values and where any records found are returned, and then the search fields are specified. In this case, the search fields are ACCT and DIV. For GETE, the search fields must be part of the I/O area as is the case for ST-ACCT and ST-DIV. Both are part of the SAMPLE-TABLE area which is passed as the I/O area.

The other fields in SAMPLE-TABLE are not set or initialized. Since GETE will only use the specified fields, this is not required.

GETQ Function

The GETQ function is the most powerful facility for searching a table and the most complex to use. It is also the most efficient function because the parameters more directly specify the search criteria and the application may need to retrieve less records. The parameters specify which fields in the table to use and what operators to use. GETQ allows any type of comparison (eg. equal, not equal, greater, etc.) on all fields in a table. This includes numeric, character and date fields. In addition, GETQ supports a range test using the 'BETWEEN' operator.

Unlike all other GET functions, GETQ does not use the I/O area as the location of the search values. The search values are specified in the same was as for GETE but **MUST NOT be in the I/O area.** All primary GET functions except GETQ save the I/O area internally to allow the search to be continued (eg. using GETN/GETP). With GETQ, since multiple search values and ranges are allowed for the same field, the I/O area would not necessarily hold all search values. Therefore, for GETQ, the I/O area is not saved internally. As such, if the I/O area is used to contain search values and a record is returned, the search values may be changed and cause incorrect results if the search is continued using GETN. **In addition, because the search values are not saved internally, they must not be changed between the GETQ request and any subsequent GETN requests.**

There is another limitation when using GETQ and complex views. With other GET requests against complex views, when the I/O area is saved internally, all fields are converted from the view format to the base table format. This allows the search to be done efficiently. With GETQ however, the I/O area is not saved and no fields are converted. **Therefore, if searching a table that is a complex view, GETQ does not support search fields where the view field has a different format than the base table. View fields that have the same format as in the base table can be used.**

The GETQ function is done like all other calls to TMMINT. However, it requires more complex parameters to specify which fields to search on, what operators to use and the search values themselves. The basic call for GETQ is shown below.

```
CALL TMMINT USING INTERFACE-CONTROL-AREA,
                  IO-AREA,
                  OPER-1, SRCH-VALUE-1,
                  OPER-2, SRCH-VALUE-2,
                  .
                  .
                  .
                  OPER-n, SRCH-VALUE-n.
```

For each search criteria, there are two parts. The first part, shown as the OPER-n fields, specifies the field in the table to search on and the operator to use. The second part specifies the search value. As stated above, the search values should never be part of the I/O area specified in the request. Each search value must be the exact same format and length as the field in the table. For range tests, the search value has a specific layout. It is made up of a 2 byte count field followed by 2 fields with the exact same format as the field in the table. The example below and the sample program SAMPAPP2 in the installation SOURCE library show both types of search values.

The operator fields are simple character types (eg. PIC X(12)). They can be any length but should generally be a minimum of 12 bytes. They are made up of two values, a field number and operator. The field number is the sequence of the field as it occurs in the table. The first field in a table would be field number 1, the second is field number 2, etc. For DB2 tables, doing a DCLGEN shows the correct sequence of the fields. The second part is the operator itself. The following table lists the valid operators:

Operator	Meaning
=	Equal
5=, <>	Not Equal
>	Greater Than
5>, <=	Not Greater Than or Less Than or Equal
<	Less Than
5<, >=	Not Less Than or Greater Than or Equal
BETWEEN	Between a range of values

Therefore, for the operator fields, the following examples show some of the different possible values that could be used:

OPER-1 = '01 = ' ... FIELD 1 IS EQUAL TO
OPER-2 = '06 > ' ... FIELD 6 IS GREATER THAN
OPER-3 = '12 BETWEEN ' ... FIELD 12 IS BETWEEN
OPER-4 = '2<> ' ... FIELD 2 IS NOT EQUAL TO

Field numbers can be up to 4 digits with zeroes optional on the left. The field number can be separated from the operator by a space but it is not required. However, a space should always follow the operator.

To use the example from GETE above, it can be converted to a GETQ request as follows:

```
========================================== Working Storage
  01  SEL-VALUE.
      10    SEL-ACCT          PIC X(8).
      10    SEL-DIV           PIC X(4).
  01  OPERATORS.
      10    OPER-1            PIC X(12).
      10    OPER-2            PIC X(12).
========================================== Procedure Division
  MOVE '02 = ' TO OPER-1.
  MOVE '03 = ' TO OPER-2.
  MOVE 'COMPSRVC' TO SEL-ACCT.
  MOVE '01'       TO SEL-DIV.
*
  CALL TMMINT USING     INTERFACE-CONTROL-AREA,
                  SAMPLE-TABLE,
                  OPER-1, SEL-ACCT,
                  OPER-2, SEL-DIV.
```

If we then needed to search the table for all names with ACCT='CORP' and SALARY BETWEEN 1000.00 and 2000.00, the following could be used:

```
========================================== Working Storage
  01  SEL-VALUE.
      10    SEL-ACCT          PIC X(8).
      10    SEL-SALARY.
            15  SEL-SAL-CNT   PIC S9(4)    BINARY.
            15  SEL-SAL-START PIC S9(6)V99 PACKED-DECIMAL.
            15  SEL-SAL-END   PIC S9(6)V99 PACKED DECIMAL.
  01  OPERATORS.
      10    OPER-1            PIC X(12).
      10    OPER-2            PIC X(12).
========================================== Procedure Division
  MOVE '02 = '       TO OPER-1.
  MOVE '04 BETWEEN ' TO OPER-2.
  MOVE 'CORP'        TO SEL-ACCT.
  MOVE 2             TO SEL-SAL-CNT.
  MOVE 1000.00       TO SEL-SAL-START.
  MOVE 2000.00       TO SEL-SAL-END.

  CALL TMMINT USING     INTERFACE-CONTROL-AREA,
                  SAMPLE-TABLE,
                  OPER-1, SEL-ACCT,
                  OPER-2, SEL-SALARY.
```

In this example, the SALARY is searched based on a range. The search values for a range must always be passed in a structure like SEL-SALARY. The first field is the count of values and is required. It must be set to 2 signifying that 2 values follow. Currently, only 2 values are allowed. The SEL-SAL-START and SEL-SAL-END fields are the exact same format and size as ST-SALARY. This is required for the range values and all search values.

For more dynamic searches, the field numbers and operators can be set based on what fields are to be used. In addition, by re-defining the selection field area, the appropriate values could be set also based on what

fields are to be used. This would allow for totally flexible searches dynamically changed at execution time based on application needs.

Preparing an Application

In most applications that will be accessing tables from dataspaces, there is virtually no preparation required to use the API other than to make sure the API load modules are available to the online region or batch job. For some applications, however, that will be loading DB2 or VSAM tables, either explicitly or implicitly (through auto-load), a DB2 bind may be required and some DB2 authority may be needed.

When a table is loaded, the API uses dynamic SQL to access the table itself and static SQL to access the TABLES/MM MS_DEFINITION table and certain DB2 catalogs. Therefore, any time an application loads a table, a DB2 plan is needed. The plan must include all the DBRM's, if any, for the application and the following TABLES/MM DBRM's from the installation DBRMLIB:

- **TBLIODB2**
- **TBLMSDEF**
- **TMMDEF1C**

These should only be included if the application will be loading tables into memory. **If the application is only retrieving records using the GET functions, these DBRM's should not be included.**

In addition, because the table being loaded is accessed using dynamic SQL, any user running the application will need SELECT authority defined to DB2 to be able to load it.

For all online applications and for batch applications that also use DB2 or run under the IMS region controller, a bind including the DBRM's is required. For batch applications that do not use DB2, no bind is required. However, for the API to connect to DB2, it needs to know what DB2 plan and subsystem to use. It does this through special JCL cards as follows:

```
//DB2PLAN     DD  QNAME=plan-name
//DB2SSID     DD  QNAME=subsys-id
```

Where: **plan-name** - is the DB2 plan to use. Any plan that includes the TABLES/MM DBRM's can be used including TMMUTIL (the plan used for the batch utilities).

subsys-id - is the DB2 subsystem.

Therefore, when running batch applications that do not use DB2 but will be loading DB2 tables, the above JCL cards can be used to inform the API about DB2 requirements. No other JCL changes are required.

6. TRANSIENT TABLE INTERFACE

Transient Table Interface

Overview

The Transient Table Interface is an optional component of TABLES/MM, implemented through an extended set of function codes to the API. With it, an application can build and maintain in-memory tables for its private use and save those tables to a dataspace or local memory. **It can also be used to update tables that have been pre-loaded into a dataspace or local memory.**

If the Interface is not available, then a result code of **FNCINVLD** is returned if any of the function codes described below are used.

When updating in-memory tables, the Transient Table Interface should be used with care because of the additional complexities the application must take into account. However, there are several uses which can justify its use:

- By eliminating or reducing the need to reload updated tables. This is especially helpful where the in-memory table must be in sync with the original table at all times and the updates occur reasonably frequently.
- Assisting in operating 24 hours, 7 days per week.
- Provides the capability to perform read-only access to DB2 and VSAM on-line while performing batch update applications.
- Provides a capability to build temporary tables in memory and save in the Dataspace or Local memory for multiple applications. This allows for transient type tables that may be used to summarize or pass data from one job or transaction to another

Features and Capabilities

When using this interface, the following features and capabilities are very important. Each of these should be understood and considered in any application that will be updating a table that has been loaded into a dataspace or local memory:

1. The updates are made only against the in-memory copy of the table. **The application is responsible for updating the original DB2 or VSAM table and making sure both copies remain synchronized.**
2. Updates to a table are always made to a temporary copy of the table. That is, the TMAK function gets the memory required and copies the table from the dataspace or local memory into it. This temporary copy is only accessible from the program or transaction that issued the TMAK function. Also, during the time the temporary table is being accessed, the program or transaction cannot access the original table.
3. The table in the dataspace or local memory remains available for concurrent read-only access by other programs during the time the temporary table is being updated. **However, any updates made by the program will only take affect after the table is saved back to the dataspace or local memory using the TSAV function.**

4. A table that is updated by CICS transactions **MUST NOT** be updated by any other non-CICS program. The TLOK/TUNL locking mechanism only locks a table within a particular CICS region or all non-CICS regions. Therefore, updates can take place from one CICS region only or any IMS region or batch job. **Under no circumstances allow a table to be updated from a CICS region and a non-CICS region at the same time.**

5. If a table to be updated is not locked using TLOK/TUNL, then it is the applications responsibility to serialize the updates to the table using its own mechanisms.

6. If a table is for transient purposes only and never saved back to the dataspace or local memory copy, then there is no requirement to lock it.

7. A complex view (eg. where columns are re-formatted) cannot be used with the transient table functions. **Only a base table or a simple view can be used.** If a complex view is used, the results will be unpredictable.

Transient Table Function Codes

The API function codes used for transient table processing and updating tables are summarized as follows:

Function	Description
TLOK	Lock a table to serialize it
TMAK	Make a temporary copy of a table
TADD	Add a row to the temporary copy of a table
TDEL	Delete a row from the temporary copy of a table
TUPD	Update a row in the temporary copy of a table
TSAV	Save the table back to the dataspace or local memory
TUNL	Unlock a table the was previously locked

Figure 6-1: Transient Table and Update Function Codes

Function Code Descriptions

Each function code is described below with some specific result codes for the function. Common results codes that can occur for many functions are described after them and all result codes are listed in Appendix B.

TLOK - Use to lock a table prior to updating. TLOK should always be issued before a TMAK if the table will be updated. The lock stops any other program from issuing TLOK for the same table in the same dataspace.

Result Codes: TBLINVLD - Table was not found.

TMAK - Make a temporary copy of a table that was loaded in a dataspace or local memory to be used exclusively by the program issuing it. Any functions executed by the program against the table will be directed to the temporary copy and not the original one. **TMAK is required before any update functions can be performed.**

Result Codes: TBLINVLD - Table was not found.
GMERRTT - Error getting memory for the transient table.

TADD - Add a row to a temporary table. The row specified in the I/O area passed is added as a new row to the table. Any indexes for the table are updated as required.

Result Codes: NOTOK - The row being added is a duplicate of another row. This only occurs when the key of a unique index already exists.

TDEL - Delete a row from a temporary table. The row specified in the I/O area passed is removed from the table. Any indexes for the table are updated as required. The row passed in the I/O area must be the complete row as stored in the table and match exactly.

Result Codes: NOTOK - The row being deleted was not found in the table.

TUPD - Update a row in a temporary table. The row specified in the **Second I/O** area passed is replaced with the row in the **First I/O** area passed. Any indexes for the table are updated as required. The row passed in the second I/O area must be the complete row as stored in the table and match exactly. The matching row found is then replaced with the updated record.

Result Codes: NOTOK - The row being updated was not found in the table or the update would cause a duplicate entry in a unique index.

TSAV - Save the temporary copy back to the dataspace or local memory making any changes permanent. That is, the changes are now available to all programs accessing the table in the dataspace or local memory. This is the opposite of TMAK. Once saved, the temporary copy is deleted and any references to the table are now made against the updated table in the dataspace or local memory.

Result Codes: TBLINVLD - Table was not found.

TUNL - Unlock the table to allow other programs to update it. **This reverses the TLOK function and MUST be executed**. TUNL should always be issued if TLOK was originally issued for the same table no matter what other functions are done.

Result Codes: TBLINVLD - Table was not found.
NOTOK - No TLOK was issued for the table.

Calling the API

The following example shows the basic call to the API when using the transient table and update function codes:

```
CALL  TMM-INT-MODULE  USING  INTERFACE-CONTROL-AREA,
                             RECORD-IO-AREA,
                             OLD-RECORD-IO-AREA.
```

In the example, an ICA, a record I/O area and an old record I/O area are passed. The record I/O area is only required for TADD, TDEL and TUPD. The old record I/O area is only required for TUPD. For TLOK, TMAK, TSAV and TUNL only the ICA is required.

Please refer to Chapter 5 (Application Programming Interface) for a layout and description of the ICA and other requirements when calling the API.

Also, the sample program SAMPAPP3, in the install SOURCE file, is a complete routine that can be used to update a table in memory. It contains examples of all the transient table and update functions.

Using Transient Table Functions

TLOK

The TLOK function is used to serialize the update process. Only one application can be updating a table at the same time. To insure this, the TLOK function must be issued for the table being updated and must occur before TMAK is issued.

To execute the TLOK function, call the API passing the ICA as a parameter with the name of the table to be locked. The ICA is the only parameter required, the I/O area is optional and not used.

The TLOK function **works differently in CICS regions versus non-CICS** regions. When a transaction in a CICS regions issues TLOK, the API internally does a **CICS ENQ** function to lock the table. In non-CICS regions, the API internally executes an **MVS ENQ macro**. As a result, the following limitations apply to the update process:

- If a table is updated by a CICS transaction, then it cannot be updated by any non-CICS program. In addition, it can only be updated within one specific CICS region. That is, if multiple CICS regions are running, to insure the table does not get corrupted, only transactions in one region should be updating any particular table.

- If a table is updated by non-CICS programs, the update can be done from IMS regions, batch jobs or TSO sessions but cannot be updated by any CICS transaction.

> ***Note:*** Be aware that the above limitations are not enforced by the API. It is up to each location to insure only CICS transactions in a specific region or only non-CICS programs update a specific table.

If an attempt is made to update a table in a dataspace from a CICS and non-CICS program at the same time, the functions may appear to work correctly. However, it is possible, that some of the updates will be lost if one program issued a TMAK while another program is updating the table. **The program that issues the TSAV last will have its updates saved but the first program's updates will be lost.**

TMAK

The TMAK function is used to create a temporary copy of a table from a dataspace or local memory to be used explicitly by the program executing the TMAK. Once TMAK is executed, any accesses to the table are made to the temporary copy. The program can also no longer access the dataspace or local memory copy of that specific table. However, no other program can access the temporary copy of the table. This is what allows tables to be updated while the original table is still fully accessible by other programs. The temporary table is the one that is always updated while the original is always available.

To execute the TMAK function, call the API passing the ICA as a parameter with the name of the table to be copied. **The table name MUST be for a base table or a simple view. If a complex view is specified, the results will be unpredictable**. The ICA is the only parameter required, the I/O area is optional and not used. Also, the ICA-FREESPACE TYPE and AMOUNT fields and ICA-LOC-FLAG can be used. Setting the freespace type and amount will allow for expansion of the table with less overhead. If not used, and additional rows cause expansion of the table, it will be moved to a larger block of memory that has to be allocated in addition to the current area already allocated. The ICA-LOG-FLAG specifies where the original table was loaded. See Chapter 5 on the API for a more detailed description of the fields in the ICA.

Once TMAK has completed, then TADD, TDEL and TUPD can be used to actually update the table (again, the temporary table is the one being changed). After all changes are made, then TSAV should be executed to save the temporary table back to the dataspace or local memory.

TADD, TDEL, TUPD

To actually update a temporary table, TADD is used to add new rows to the table, TDEL is used to delete rows and TUPD is used to update a row.

To execute TADD, TDEL or TUPD, call the API passing the ICA as the first parameter with the name of the table to be updated, a record I/O area, and for TUPD, an old record I/O.

When adding, the new row must be passed in the record I/O area. All column values should be specified in the correct format for the table.

When deleting, the row to be deleted is also passed in the record I/O area. The complete row must be passed as all columns of the row are compared. If all columns do not match, no record is deleted. Therefore, before deleting a row, a GET request should be made to insure the row to be deleted is found and passed on the TDEL request.

When updating, the new row is passed in the record I/O area and the old row is passed in the old record I/O area as shown in the above example. The update process searches for an exact match of the old row, and if found, replaces it with the new row.

In all cases, if there are any indexes for the table, all of them will be updated as required. If a row is updated and the key for an index is changed, the old index entry is deleted and a new index entry is added.

TSAV

The TSAV function is used to save the temporary copy of a table back to the dataspace or local memory. Once TSAV is executed, the temporary copy is freed and any accesses to the table are made to the updated copy in the dataspace or local memory. TSAV is equivalent to re-loading a table from the original DB2 table or VSAM file after it has been updated, but uses much less resources and requires no I/O.

To execute the TSAV function, call the API passing the ICA as a parameter with the name of the table to be saved. The ICA is the only parameter required, the I/O area is optional and not used.

TUNL

The TUNL function is used to release the lock for the table locked using a TLOK request. Only issue TUNL if the TLOK was successful. It should be the last function executed for the table after any updates and TSAV have completed.

> ***Note:*** If TLOK is issued, TUNL should always be issued. If not, a table could remain locked permanently until the region or job that issued the lock is terminated.

To execute the TUNL function, call the API passing the ICA as a parameter with the name of the table to be saved. The ICA is the only parameter required, the I/O area is optional and not used.

Summary

The Transient Table processing is a very powerful, but complex component of the MEMORY MANAGER. If used correctly and for the right reasons, it can save enormous resources for keeping the in-memory copy of a table synchronized with the original table. However, it should be considered and planned for with great care to insure it is not misused and that applications are thoroughly tested before being implemented.

7. VSAM Component

VSAM Component

Overview

The VSAM Component is an optional part of TABLES/MM that eliminates the need for DB2 when defining and loading tables. TABLES/MM normally uses a DB2 table to maintain table, view and index information (eg. in the MS_DEFINITION table). This requires that any program or transaction that will be loading tables into memory, be correctly setup to access DB2 (eg. binds, plans, authorizations, etc.), even if the tables being loaded are not DB2. However, with the VSAM Component, TABLES/MM will allow either a DB2 table or a VSAM KSDS to be used for storing table definitions. In addition, when using the VSAM KSDS instead of the DB2 MS_DEFINITION table, all other DB2 access is bypassed so that no connection to DB2 is needed, messages will correctly identify VSAM problems where appropriate and utilities will run with either type.

Functions and Capabilities

The VSAM component is useful when DB2 is not installed at a specific location or not available on a particular CPU. By using a VSAM KSDS for definitions, DB2 is not needed, allowing the Memory Manager to run under these circumstances. Additionally, the following should be considered:

- **Only VSAM or TABLES/MS tables can be loaded**. If DB2 tables from another system need to be loaded, they must be copied to a VSAM file, defined to TABLES/MM and then loaded into memory.

- **Regions which do not allow access to VSAM files are not supported**, including IMS MPR regions. However, any normal batch or TSO region or any CICS region can be used. VSAM access is done through normal VSAM macros in non-CICS regions and through standard CICS interfaces when running under CICS.

- The VSAM component is **NOT REQUIRED** to load VSAM tables. They can still be loaded without it but require DB2 to be active and available.

- Normally, if DB2 is not available or down, tables cannot be loaded into memory, no matter what type of tables they are. With the VSAM component, however, if the tables are defined using it, then VSAM tables could be loaded even if DB2 is not available.

- **As a backup facility**, the DB2 MS_DEFINITION table can be periodically unloaded to a sequential file and repro'ed into a VSAM KSDS to be used when DB2 is not available. This would always allow VSAM files to be loaded into memory, even without DB2. User DB2 tables could also be unloaded and repro'ed to VSAM allowing them to be loaded into memory anytime also. (**Note**: TABLES/MM does not supply a utility to unload the MS_DEFINITION table or load it into VSAM. However, sample DB2 utilities can be used to unload DB2 tables to sequential files and IDCAMS can be used to load them into VSAM files using the REPRO command.)

Using the VSAM Component

When included with the Memory Manager, the VSAM component is integrated into the product and requires no changes in applications or programs. Also, there are no additional requirements to install it, beyond setting up the VSAM KSDS. In addition, since the VSAM component is activated at the job or region level, it allows for using either the VSAM KSDS or the DB2 table as required. One region can be running with it and another without it.

To activate the VSAM Component is a very simple process. In a batch job, CICS region or TSO session, simply include the following JCL card:

```
//TBLMSDEF  DD   DSN=your.vsam.ksds,DISP=SHR
```

This is true for any job or region that will be loading tables or defining tables, whether the TMMUTIL utility, TMMDEFN, a user written application, or CICS transaction. In addition, **for CICS, the FCT must also include an entry for TBLMSDEF** or use RDO to define it.

The TBLMSDEF DD card represents the MS_DEFINITION table where all TABLES/MM table, index and view information is stored. **It is used to replace the DB2 table and tells TABLES/MM not to use DB2 at all.** If the TBLMSDEF DD card is present whenever a table is loaded, TABLES/MM will always attempt to use the associated VSAM dataset and will bypass DB2. **Therefore, always make sure there is no TBLMSDEF DD card present in any job or region when loading tables if it is not required.** In CICS, also make sure the TBLMSDEF file is not defined in the FCT or an error will occur.

Setting Up a VSAM Definition Dataset

IN TSO

Before using the VSAM component, a VSAM KSDS must be defined and initialized. This can be done on TSO or in batch using the TMMVIMSD clist supplied in the installation CLIST library. On TSO in READY mode or using an ISPF Command Line, execute the following clist using the appropriate parameters as described below:

%TMMVIMSD dataset-name RESET NEW VOLUME(volser)

Where:

dataset-name - The name of the VSAM file to create and/or initialize. The name must be fully qualified with or without quotes.

RESET - Initialize the dataset even if it was already done. This will cause all existing records to be deleted. **ONLY specify this when an existing dataset is to be reset back to its initial state (eg. empty and ready to use). BE CAREFUL.**

NEW - Specifies that a new KSDS is to be defined with the specified name. The VOLUME parameter is required if NEW is specified.

VOLUME - Specify the name of the volume to use when defining the new KSDS. This is required when NEW is specified.

Examples:

%TMMVIMSD USER1.MM.MSDEF NEW VOL(UVOL01)

- Will create and initialize a new VSAM KSDS on volume UVOL01.

%TMMVIMSD USER2.MM.MSDEF RESET

- Will re-initialize an existing VSAM file deleting all existing definitions from it.

IN BATCH

To do the same thing in batch, use IDCAMS to define a VSAM KSDS based on the sample definition in member VSAMDEF of the installation SOURCE library. After the define, execute TMMVIMSD program to initialize the KSDS. In the installation JCL library, the TMMVIMSD procedure can be used to execute it or use the following sample JCL with appropriate changes for your installation :

```
//STEP             EXEC  PGM=TMMVIMSD
//STEPLIB   DD     DSN=mm.loadlib,DISP=SHR
//TBLMSDEF  DD     DSN=msdef.vsam.ksds,DISP=OLD
//SYSOUT    DD     SYSOUT=*
```

This will initialize the VSAM file specified on the TBLMSDEF DD card. The TMMVIMSD procedure in the installation JCL library has more details and describes some optional parameters that can be used.

Other Changes

When using the VSAM component there are changes to some messages that reflect the use of VSAM versus DB2. This can occur when using TMMUTIL, TMMDEFN and the on-line utilities. The changes only affect messages when errors occur and for a few informational messages. Return codes and reason codes may be different when processing VSAM and these are described for the appropriate messages. Please refer to Appendix B for all messages and descriptions. Refer to Appendix E for reason codes and special SQL-like codes that are generated by TABLES/MM when accessing a VSAM definition file. The SQL-like codes are passed back when calling the interface in the ICA and allow for consistency when checking results and reporting errors in applications.

Summary

The VSAM component is a very easy facility to make use of and allows TABLES/MM to be used when DB2 is not available. If not activated, it will not impact TABLES/MM in any way and requires no application changes if it is activated.

APPENDICES

Appendix A - Messages

The TABLES Memory Manager components use the same basic format for all messages. Refer to the following breakdown when reviewing messages:

TMMf###t - message text

Where **TMM**= the prefix for all TABLES/MM messages.

f= The functional area, as follows:
- D - Definition Utility (TMMDEFN)
- I - Interface Messages (TMMINT)
- L - Load Utility (TMMLOAD)
- R - Report Utility (TMMRPT)
- S - Dataspace Setup (TMMSTART)
- U - Batch Utility (TMMUTIL)
- V - VSAM Batch Utility (TMMVIMSD)

###= The message number.

t= The message type, as follows:
- E - Error occurred
- I - Informational message
- R - Response required

TMMDEFN - Batch Definition Messages

TMMD000E - DB2 CONNECTION FAILED, PLAN OR AUTH ERROR

The connection to DB2 could not be made. This is due to an invalid plan name, the user running the job is not authorized to execute the plan, the DB2 subsystem is invalid, or DB2 is not running. If the installation of TABLES/MM was done correctly, this should normally occur only when the user is not authorized for the DB2 plan.

TMMD001E - INVALID FUNCTION CODE

The first word on each control card input must be a valid function code. Check the card in question for a misspelled function code. Valid functions include REPLACE, CREATE, COLUMN, COPY, DROP, COMMIT, ROLLBACK, LIST and abbreviations for them.

TMMD007E - CREATE FAILED, DUPLICATE ENTRY

When creating a new entry, an existing entry with the same name was already found to exist. Either delete the old one, use REPLACE or change the name.

TMMD008E - RECORD NOT FOUND

The specified entry could not be found when replacing or dropping it. Use LIST to see if the entry exists or use CREATE if this is a new entry.

TMMD009E - UPDATE/DELETE FAILED, RECORD NOT FOUND

When replacing or dropping a record, the entry was not found. Use LIST to see if the entry exists or use CREATE if this is a new entry.

TMMD010E - ROLLBACK: ALL UPDATES REMOVED

An error occurred in processing a function. All changes are removed when this occurs. Fix the error that occurred and re-run the job with the same input. The COMMIT function can be used to keep changes from being removed.

TMMD011E - ROLLBACK: UPDATES REMOVED AFTER FUNCTION nnnnn

An error occurred in processing a function. All changes after the function nnnnn have been removed. This occurs if COMMIT was used to keep changes. Fix the error that occurred and re-run the job with the input starting after function nnnnn.

TMMD012E - NO INPUT TO PROCESS

No input was found. Check to make sure the file with DDNAME of CONTROL points to valid input. Also make sure that all input was not commented out (eg. all cards starting with '*', '--').

TMMD013E - TOO MANY INPUT RECORDS FOR THIS FUNCTION

For any one function (eg. CREATE, DROP, etc), there is a maximum of 500 cards. This should normally never occur. Check the input data to make sure it is valid.

TMMD014E - INVALID INPUT, TOO MANY PARAMETERS

There were too many parameters specified for this function. Check the first card for the function to make sure no extra values were specified. Review the documentation for the particular function to see what parameters are valid.

TMMD015E - INVALID OBJECT TYPE (2ND PARAMETER)

The object specified is invalid. The object is the value after the function and must be one of the following: INDEX, TABLE, VIEW, or a pluralized version.

TMMD016E - FUNCTION NOT AVAILABLE FOR THIS OBJECT

For the object, the specified function is not available. Review the documentation for the particular object to see what functions are available.

TMMD017E - SQL ERROR, ?

An SQL error occurred processing the specified function. This should not normally occur. Check with your DB2 administrator for the SQL error in question or call SSI for support.

TMMD020E - ERROR IN CARD nnnn.mmmm: ???????????

In function nnnnn, card mmmm is invalid for the reason shown. Review the card in question and adjust as required. Review the documentation for the type of card for valid parameters.

TMMD030E - COPY MEMBER NOT FOUND IN THE LIBRARY

For a COPY/INCLUDE request, the member could not be found in the library with the DDNAME specified or in the default library with DDNAME of COPYLIB.

TMMD031E - COPY MEMBER EMPTY OR NO VALID STATEMENTS FOUND

The member specified was either empty or had all blank or all comment cards. Check the member name to make sure it is correct and the DDNAME is specified if not using COPYLIB.

TMMD032E - OPEN FAILED FOR COPY LIBRARY

The COPYLIB or ddname specified could no be opened. When copying COBOL or Assembler copybooks, the DDNAME must be defined in the JCL or allocated correctly. If no DDNAME was specified on the COPY statement, make sure a COPYLIB DD card is defined in the JCL pointing to the dataset containing the member specified.

TMMD033E - COPY MEMBER TO LARGE TO PROCESS

The member being copied was too large to process. The maximum size a member can be is 500 records. Check the member and the library to make sure they are correct.

TMMD034E - UNKNOWN ERROR READING THE COPY MEMBER

When reading the member from the specified dataset, an I/O or other error occurred. Check to make sure the member name and the dataset are correct.

TMMD035E - MEMBER NAME REQUIRED ON COPY CARD

When copying a COBOL or Assembler copybook, a member name is required.

TMMD036E - LANGUAGE TYPE REQUIRED ON COPY CARD

Either COBOL or ASSEMBLER must be specified after COPY to denote the type of copybook to process.

TMMD037E - TOO MANY PARAMETERS ON COPY CARD

When scanning the COPY card, more parameters than should be were found. Remove any invalid parameters and check the documentation for the COPY card for the parameters that are valid.

TMMD038E - INVALID STATEMENT IN COPYBOOK, STMT ####

When processing the copybook member, statement #### was invalid. Review the copybook member and make sure it can be correctly compiled. Also, the statement before or after the specified number should be checked.

TMMD039E - ERROR PROCESSING COPYBOOK, ERROR CODE = ####

When processing the copybook, an error occurred. This should only occur if the copybook could not be successfully compiled. Error Code is the return code from the Assembler or one of the following:

101 - In a COBOL copybook, an occurs was found on a group level. These are not supported.

TMMD040E - (#) REQUIRED BEFORE THE INDEX ID

When processing an INDEX, the '#' is required after the INDEX keyword. For REPLACE and DROP, the '#' and the index number are required. For create, the '#' is required to insure that the number is not inadvertently forgotten.

TMMD041E - INDEX ID INVALID

A valid 1 or 2-digit ID is required after the '#' when processing an index. No spaces should be between the '#' and the ID.

TMMD042E - TABLE NAME REQUIRED FOR INDEX FUNCTIONS

When processing an index, a table name is always required.

TMMD043E - INDEX ID REQUIRED FOR THIS FUNCTION

For the REPLACE and DROP function, an index ID must be specified. The ID must immediately follow the '#' with no spaces in between.

TMMD044E - INVALID KEYWORD FOR INDEX FUNCTION

An invalid keyword was found for the INDEX function. Review the documentation for the valid parameters that can be specified on the INDEX card.

TMMD045E - TOO MANY COLUMNS SPECIFIED (16 ALLOWED)

An index can currently have up to a maximum of 16 columns in the key. Remove all extra column cards so that only 16 remain.

TMMD046E - INVALID TABLE NAME OR TABLE DOES NOT EXIST

The table name was invalid or does not exist. An index can only be created for a DB2 table defined in the DB2 catalog or for a VSAM table previously defined using TMMDEFN. Check to make sure the table specified exists.

TMMD047E - AT LEAST ONE COLUMN REQUIRED FOR INDEXES

An index requires at least one column. Make sure that a COLUMN card follows the INDEX card. COPY cannot be used to define columns for indexes.

TMMD048I - ID SET, NEW INDEX CREATED WITH ID = nn

Informational message issued when an index is created and the ID was not specified. TMMDEFN automatically assigns the next ID number as one plus the previous highest existing index for the table. The nn is the index number that was assigned.

TMMD060E - VIEW NAME REQUIRED

When processing a view the VIEW name is always required. No name was found following the VIEW keyword.

TMMD061E - BASE TABLE NAME REQUIRED

When creating or replacing a view, the base table name is required. The base table is the physical table the view is defined for. It can be a table in the DB2 catalog or a VSAM table defined using TMMDEFN.

TMMD062E - NO PARMS ALLOWED AFTER THE BASE TABLE NAME

On a view function, parameters followed the base table name. None are allowed. Review the VIEW function documentation for valid parameters.

TMMD063E - INPUT FOLLOWING DROP, NONE ALLOWED

When Dropping a view, table or index, secondary cards followed the function card. That is, a COLUMN, COPY or other card was found. There should be no secondary cards for the DROP function. Remove any such cards.

TMMD064E - BASE TABLE NOT FOUND

The base table name specified for the view does not exist. The base table must be defined in the DB2 catalog or be defined in the TABLES/MM MS_DEFINITION table as a VSAM file before a view can be created for it.

TMMD065E - TOO MANY COLUMNS DEFINED, MAX IS 300

The maximum columns that can be defined for a view is 300.

TMMD080E - TABLE NAME REQUIRE

A table name is always required when processing a table. Specify the table name following the TABLE keyword on the function card.

TMMD082E - NO PARMS ALLOWED AFTER THE TABLE NAME

Parameters were found after the table name. None are allowed. Review the documentation for valid parameters with table functions.

TMMD083E - INPUT FOLLOWING DROP, NONE ALLOWED

When Dropping a view, table or index, secondary cards followed the function card. That is, a COLUMN, COPY or other card was found. There should be no secondary cards for the DROP function. Remove any such cards.

TMMD084E - DB2 TABLE NOT FOUND

The table being created was not a VSAM table (eg. starts with VSAM.) and the name could not be found in the DB2 catalog. Check to make sure the name is correct. If defining a VSAM table, the name must start with VSAM.

TMMD085E - AT LEAST ONE COLUMN REQUIRED FOR TABLES

A table must have at least one column. Specify the column definitions for the table after the CREATE/REPLACE TABLE card or use the COPY card to define the columns. Also, make sure the following card is correctly entered.

TMMD098E - ERROR: FUNCTION TERMINATED, RC = nnnnn

An error occurred processing the function. Review the previous messages for the error that occurred. The return code will specify what type of error occurred:

08 - Error occurred processing the function
12 - Invalid parameter
16 - Severe error

TMMD099E - INTERNAL LOGIC ERROR. **(?)**

A situation was found that should not occur. The (?) specifies the internal error code. This should be reported to SSI immediately.

TMMD101I - TMMDEFN PROCESSING RESULTS:

This is the first message displayed from TMMDEFN.

TMMD102I - TMMDEFN ENDED, RC = nnnnn

This is the last message displayed and shows the highest return code. This is also the return code for the JOB/STEP.

TMMD103I - FUNCTION COMPLETED NORMALLY, RC = 0

This message is displayed for each function that completes successfully.

TMMD104I - COMMIT SUCCESSFUL, PROCESSING COMPLETED

If all functions complete successfully, a DB2 COMMIT is issued. This message is displayed if the COMMIT is successful.

TMMD105I - PROCESSING THE FOLLOWING FOR FUNCTION NO. nnnnn

This message specifies that a function was found and is about to be processed. All cards associated with this function (eg. the function card and any secondary cards) are displayed following the message. The function number, nnnnn, is just a count of the functions that have been processed so far and may be displayed in other messages.

TMMD106I - NAME NOT FOUND, NO ENTRIES LISTED

For the LIST function, no entries were found to list. The function completes normally.

TMMD017I - PROCESSING DEFINITION TABLE: creator-id.MS_DEFINITION

All definitions are maintained in a DB2 table with a name of MS_DEFINITION and a creator-id specified when the system is installed. This is the DB2 table that will be processed.

TMMD108I - LIST COMPLETE

The list function has completed. One or more entries may have been listed. The list function always completes normally even if no entries are found.

TMMD109I - COPY MEMBER PROCESSED, RECORDS READ = nnnnn

A copy member was processed successfully and nnnnn was the number of records read from the member.

TMMINT - Interface (API) Messages

TMMI001I - CLIENT CONNECTED TO DATASPACE ID = **??**

The first time an application calls the API, the interface sends this informational message to the JES2 job log of the on-line region or batch job if the connection was made. It allows for validating that a region or job has connected to the correct dataspace. If dataspace access is turned off using the TBLDID card, then this message will not be sent. The ?? is the id of the dataspace connected to.

TMMI002E - ERROR INITIALIZING CLIENT, RC = rc-errc

If the interface fails to connect to a dataspace, it sends this message to the JES2 job log of the on-line region or batch job. The return code (RC) value specifies what error has occurred. The following is a list of possible values:

08-0004	The dataspace specified on the TBLDID card in the JCL or the default one is not active. Start the appropriate dataspace and re-run the job or transaction. An on-line region does not have to be re-started. The application issuing the request will receive a result code of ERRNODS in the ICA. For testing, use a 'TBLDID DD DUMMY' to bypass dataspace access.
21-????	A request (GETMAIN) for memory failed during the interface initialization. The ???? is the return code from the GETMAIN macro. The interface always requests memory from above the line and as a result, this should never occur. If it does, the system default region size for above the line memory may have to be increased.
24-0000	An invalid dataspace id was detected when processing the TBLDID dd card from the JCL. The dataspace id must be two non-blank, alpha-numeric characters following the TBLDID (eg. TBLDID01 would be used to specify dataspace 01).
Other	All other values are internal errors and should be reported to technical support.

TMMLOAD - Online Transaction Messages

TMML000E - TMMLOAD TRANSACTION ENDED.

The TMML full screen transaction was ended by the user.

TMML001E - SEE REFERENCE GUIDE FOR HELP.

While using the TMML full screen transaction, PF1 was pressed to request help. Currently, TMML does not support help. See the TABLES/MM Reference Guide for help.

TMML002E - SCROLL UP IS NOT SUPPORTED.

While using the TMML full screen transaction, PF7/19 was pressed to scroll up. Either there is nothing to scroll or scroll up is not supported. Re-issue the same request to start from the beginning again.

TMML003E - INVALID OR NOT ACTIVE PFK PRESSED.

While using the TMML full screen transaction, a PF Key that is not valid or not functional at the current time was pressed. Use a different PF Key or Enter to perform a function.

TMML004E - PLEASE ENTER A FUNCTION TO EXECUTE.

While using the TMML full screen transaction, ENTER was pressed but no function was specified. Enter a valid request (eg. LIST, LOAD, etc.) in the FUNCTION field.

TMML005E - INVALID LOCATION, ENTER L, D OR SPACE.

While using the TMML full screen transaction, an invalid value was entered in the Location of Table field. Valid values are **'D', 'L' or no value** (eg. leave blank).

TMML006E - FREE SPACE AMOUNT MUST BE NUMERIC.

While using the TMML full screen transaction, an invalid value was entered in the Free Space: Amount field. Enter a valid number or leave blank.

TMML007E - FREE SPACE TYPE MUST BE R OR P.

While using the TMML full screen transaction, an invalid value was entered in the Free Space: Type field. Valid values are **'R' for Records** or **'P' for percent.**

TMML008E - END OF LIST OR NO TABLES TO LIST.

While using the TMML full screen transaction, scroll down was requested but there are no more entries; or when entering LIST, there are no tables to list. Specifying the Location of Table will affect where the list of tables is retrieved from.

TMML009E - UNKNOWN PARAMETER PASSED TO TMMLOAD.

While using the TMML transaction in line mode, invalid parameters were found. Refer to the Reference Guide for valid parameters that can be passed to TMML in line mode.

TMML010E - FUNCTION COMPLETED OK.

While using the TMML transaction, the function was processed and completed normally (eg. if LOAD was selected, the load completed ok).

TMML040E - FUNCTION FAILED. RESULT = **?**, REASON = **?**, SQLCODE = **?**

While using the TMML transaction, a function passed to TMMINT failed. The result code, reason and sql codes from TMMINT are displayed. Refer to the Appendix on Result and Reason Codes for more information or check the SQL Code for the cause of the problem.

TMML041- STATS FOR table: RECS = ###

While using the TMML transaction in line mode, a STGF (Stats Request) was issued. The table name and

number of records loaded in memory is displayed.

TMMRPT - Batch Report Messages

TMMR001E - NO DATASPACES ACTIVE OR ERROR RETRIEVING INFORMATION

When TMMRPT utility finds no active dataspaces or an invalid return code is received from the dataspace interface module, this message is displayed in the SYSOUT print dataset. If no dataspaces are active, start any that should be. If dataspaces are active, call for support.

TMMR002E - DATASPACE NO LONGER ACTIVE OR ERROR RETRIEVING INFORMATION

While processing a specific dataspace, TMMRPT found the dataspace had been deleted. This should only occur if the dataspace job is terminated while TMMRPT is running.

TMMR003E - ERROR RETRIEVING INFORMATION FOR A TABLE

When retrieving information for a specific table, TMMRPT found the table to no longer be available. This should only occur if the dataspace was stopped and restarted while TMMRPT was running.

TMMSTART - Dataspace Setup Messages

TMMS001R - REPLY "STOP**??**" TO END D/S-**??-jjjjjjj**

After a dataspace is created and available to use, this message is displayed on the system console by TMMSTART. The **'??'** is the Dataspace ID as entered for the DID= parameter in the start-up JCL and 'jjjjjjj' is the job or task name. This message is used to allow a specific dataspace to be stopped. When a dataspace is no longer needed, the operator should reply 'STOP??' to terminate the dataspace with an id of '??'.

TMMS002E - ERROR, RC = **???** FOR FUNCTION = **????????**

If an error occurs during dataspace creation or initialization, this message is sent to the system console by TMMSTART. Clean-up is performed and the dataspace job is ended with a return code = 8. The following is a list of FUNCTION values and associated return codes:

DSPSERV -	Error creating a common dataspace. RC is the return code from the DSPSERV macro. Possibly too many common dataspaces created. See DSPSERV macro for possible return code values.
TESTAUTH-	The TMMSTART is not authorized to create common dataspaces. Make sure TMMSTART is in an APF authorized library and the module is linked with parameter AC=1.
ENQ-ID -	The dataspace with the ID specified for the DID= parameter in the start-up JCL is already active. Change the ID or terminate the active one. RC is the return code from the ENQ macro and can be checked if the dataspace was not active.

GETPARM - An error was detected in the start-up parameters. Refer to Section 2 on Creating a Dataspace for parameter descriptions and requirements. Fix the parameter and re-start the dataspace. RC is always set to 8.

KEY9-INV - The dataspace was started using KEY=9 but the computer being used does not support it. Only use KEY=8 when starting dataspaces on hardware that does not support Sub-system Storage Protection (SSP).

Any Other - Any other function codes returned are internal errors and should be reported to technical support.

TMMS003E - INVALID REPLY

This message is issued by TMMSTART if the operator replies incorrectly to message TMMS001R. If STOP is not entered and followed by the correct dataspace id, this message is displayed. No action is taken by TMMSTART other than to re-issue TMMS001R. The operator should respond with the correct dataspace id or to the correct STOP message.

TMMUTIL - Batch Utility Messages

TMMU001E - INVALID CONTROL CARD CODE

A control code in column 1 to 4 was invalid. All control codes are 4 characters, padded on the right with blanks. Fix the invalid code and resubmit the job.

TMMU002E - DATA PARAMETER MISSING OR BLANK

On a DATA card, there are four required parameters. Each must be separated from the other by 1 or more blanks. The first one must start in column 5. Refer to the DATA control code for a description of each parameter.

TMMU003E - DATA TYPE INVALID (C, N, B, OR P)

On the DATA card, the type parameter must be C, N, B, or P. Review the parameters and fix the type value.

TMMU004E - INVALID POSITION SPECIFIED

On the DATA card, the position value was invalid or not specified. Make sure the position is numeric and within allowable ranges for the record.

TMMU005E - INVALID LENGTH SPECIFIED

On the DATA card, the length was invalid or not specified. Make sure the length is numeric and in the range of 1 to 32.

TMMU006E - INVALID VALUE OR TYPE

On the DATA card, the value could not be converted to the specified type. Make sure the value is a valid number with no decimal places and is not larger than the type will allow.

TMMU007E - TABLE NAME MUST BE SPECIFIED FIRST

A FUNC card was specified before a TABL card was processed. A table name is required before any interface function can be executed.

TMMU008E - INVALID REPEAT COUNT

The REPEAT-COUNT value specified after the function code on the FUNC card was invalid or too large. Make sure the value is numeric and less than 9999.

TMMU009E - INVALID SIZE OR COULD NOT SET

In attempting to set the SIZE of the record to display, the value entered was not numeric or invalid or if SET was specified, statistics could not be retrieved for the table in order to get the record size. Fix the value or make sure the table is loaded before specifying the SIZE card.

TMMU010E - INVALID SORT NAME (BLANK)

A SFLD card was entered but no column name was specified. The column name is required before the A/D order code.

TMMU011E - SORT SEQUENCE MUST BE A OR D

On the SFLD card, only one column name is allowed with an optional order value. The order must be A for ascending or D for descending.

TMMU012E - INVALID DISPLAY LINE SIZE

The value entered for the DISP parameter was not numeric, not specified or invalid. It must be in the range of 1 to 204.

TMMU013E - INVALID FREE SPACE VALUE

On the FSPC card, the free space parameter was not numeric. Enter a valid numeric value for the percent of free space to use when loading tables.

TMMU015E - INVALID LOCATION FLAG VALUE

For the LOC card, an invalid value was specified. Enter 'L' to set the flag to load a table into local memory or blank to load the table into a dataspace.

TMMU016E - ID INVALID OR THE SAME, NOT CHANGED

On the DID card, to process a different dataspace, the ID must be different than the current value and must be a valid dataspace id (eg. 2 alpha-numeric characters).

TMMVIMSD - VSAM Component Utility Messages

TMMV001E - FILE ALREADY INITIALIZED. USE RESET PARM IF NEEDED.

The VSAM KSDS being initialized as an MS_DEFINITION file has already been initialized. If the file was already initialized, it is all set to be used. If the file is to be re-initialized, then specify the RESET parameter. **Be aware, using RESET will delete all definition records currently in the file.**

TMMV002E - ERROR OPENING FILE FOR OUTPUT (*)

The VSAM file to be initialized could not be opened for processing. Check to make sure a TBLMSDEF DD card is correctly defined in the JCL or allocated. It must specify a valid VSAM KSDS dataset that was created using IDCAMS or the TMMVIMSD clist. Also, review the file status code and VSAM error information for the cause of the problem.

TMMV003E - ERROR WRITING INITIAL RECORD (*)

When writing a record to initialize the VSAM MS_DEFINITION FILE, the write failed with a VSAM error. Make sure the dataset specified on the TBLMSDEF DD card is a valid VSAM KSDS and is correctly defined. Also, review the file status code and VSAM error information for the cause of the problem.

TMMV004E - ERROR OPENING FILE TO DELETE RECORD (*)

When re-opening the VSAM MS_DEFINITION file for I-O processing to delete the initial record that was inserted, the OPEN failed. This should not generally occur. Review the file status code and VSAM error information for the cause of the problem.

TMMV005E - ERROR DELETING INITIAL RECORD (*)

The DELETE of the initial record failed. This should not generally occur. Review the file status code and VSAM error information for the cause of the problem.

TMMV007I - FILE RESET SUCCESSFUL

The VSAM MS_DEFINITION file was reset. All existing records were deleted and the file is available for use.

TMMV008I - FILE INITIALIZED OK

The VSAM MS_DEFINITION file was initialized correctly and is available for use.

TMMV009I - ENDED, RC = ####

The TMMVIMSD program has ended with the specified return code. The following return codes are valid:

0000 - Normal, program ran successfully.
0004 - VSAM file was already initialized. No changes made.
0008 - Error occurred in processing. Review error messages for the problem.

(*) **The TMMVIMSD program displays the file status code and VSAM error information when appropriate. Please refer to the COBOL II manuals for a description of file status codes. For VSAM error information, refer to the appropriate VSAM manuals.**

Appendix B - API Result Codes

The API always returns a result code in the Interface Control Area to the calling program. There are two results that mean the function was executed correctly and the others mean the function was not executed correctly. The following summarizes all result codes:

Valid Codes - Function completed

OK - The function executed normally. For a GET request a record was returned in the record I/O area.

END - A GET request executed normally, but no record is returned. A record matching the criteria was not found, the end of the table was reached for a forward request or the start of the table was reached for a backward request. For STGF/STGN, there are no more tables.

Invalid Codes - Function did not complete

DUPREC - When adding or updating a row using TADD or TUPD, a duplicate row was found. For tables that do not allow duplicates, this is an error.

EFFERROR - Effectivity error, occurs when a GEFF function is requested for a table that was not defined for effectivity.

ERRCALLM - The interface module, TMMINT, was called incorrectly. This occurs if the interface module is statically linked with the calling program. The API must be called dynamically at all times.

ERRCMPR - Error during compress function. This is caused if a compress is already in progress for the dataspace being processed.

ERRNODS - Error accessing a dataspace. The default dataspace or the one specified on the TBLDID DD card is not active. Either start the dataspace or turn off dataspace processing using a '//TBLDID DD DUMMY' card with blanks for the id. This result only occurs on the first call to the API when it tries to connect to the dataspace, even if the LOC-FLAG is set to 'L'.

ERRELOAD - Error during table reload. This occurs for a GETN, GETP, GETS, or GETR that is executed for a table that has been reloaded after the original call that set the table position. After the reload, the table position may be invalid due to records being added prior to the current position.

ERREXPTT- When adding a row to a transient table using TADD, the table could not be expanded to add the row. Make sure free space was specified on the TMAK requerst to allow expansion.

ERRGTMEM - A request for memory (CICS or OS Getmain) failed.

ERRLDINT- Error when issuing a load for module TMMINT to make it resident. This should only occur if the module is not available and it was statically linked with the calling module (which would be invalid also).

ERRSAVTT- When saving a transient table using TSAV, the table could not be saved back to the dataspace. Make sure the dataspace has enough space left or was not stopped.

ERRVIEW - Error creating a view dynamically. This result code occurs in R2.0 only.

ERRVWCNV- When converting columns in a complex view, the conversion failed. This occurs if invalid data is passed in the I/O area (eg. non packed data for a packed decimal column or an invalid date).

FNCINVLD - The function code passed to the API was invalid. Refer to the list of valid function codes in Section 5. Some function codes are part of optional components of TABLES/MM and may not be installed or available.

GMERRSAV- A request for memory (CICS or OS Getmain) failed when trying to allocate a save area for a table.

GMERRTT - A request for memory (CICS or OS Getmain) failed when trying to allocate memory for a transient table during TMAK processing.

NOTOK - The function could not be processed for several reasons. The current table position was invalid. This occurs for example if a GETN or GETP function is executed before a function that sets the position (eg. GETF, GETL, GETP or GEFF). For a GETL or GETG, if the table has no sort fields and more than one search field was specified, NOTOK is returned. Also, for transient table processing, NOTOK is returned when a record is not found when deleting or updating or when a duplicate key is found when adding or updating a row.

TBLINVLD - The table specified could not be found. It was not loaded or possibly loaded into a different dataspace than the task was set up to access. For the LOAD function, it means that the table was not found or an error occurred in accessing it.

Appendix C - Date Formats

When defining tables or views, the FORMAT parameter requires a date format code and on the Effectivity Definition panel, the Date Format field is used to enter the code that specifies the format of the break-in and break-out dates. The format codes allowed are shown in the following table with the date layout and an example. For effectivity, any format is acceptable, but the date column itself must be a DB2 date field or a character field that can be sorted correctly (ie. the date must be in year-month-day format).

Code	Date Format	Example
1	MMDDYY	123113
2	MMDDYYYY	12312013
3	DDMMYY	311213
4	DDMMYYYY	31122013
5	YYMMDD	131231
6	YYYYMMDD	20131231
7	MM/DD/YY	12/31/13
8	MM/DD/YYYY	12/31/2013
9	DD/MM/YY	31/12/13
A	DD/MM/YYYY	31/12/2013
B	YY/MM/DD	13/12/31
C	YYYY/MM/DD	2013/12/31
D	YYDDD	13365
E	YYYYDDD	2013365
F	YY/DDD	13/365
G	YYYY/DDD	2013/365
H	DD-MMM-YY	31-DEC-13
I	DD-MMM-YYYY	31-DEC-2013
J	MON DD,YYYY	DECEMBER 31, 2013
K	MMM DD,YYYY	DEC 31, 2013
L	YYYY-MM-DD	2013-12-31
M	DD.MM.YYYY	31.12.2013
S	TIMESTAMP	2013-12-31-00.00.00.000000

Figure C-1: Date Format Codes

Appendix D - TABLES/MS Interface

TABLES/MS is a component of the TABLES Management System from SSI that manages, controls and maintains tables in IMS/DLI databases, VSAM databases, or user VSAM files. TABLES/MM allows tables maintained by TABLES/MS to be loaded into dataspaces and/or local memory. Once loaded, the tables can be processed by the API exactly in the same way as for DB2 tables.

Transactions running online under IMS or CICS can load and process all allowable tables by using the standard API function codes. Tables can be loaded, freed, searched, etc. by calling the interface module. The only difference in the call is the table name.

Applications running in batch, that do not need to load tables, can run without any JCL changes and use the API in exactly the same was as for DB2 tables. That is, DB2 tables, TABLES/MS tables or User VSAM tables can be searched and processed by calling the API with no application changes. However, if the tables are to be loaded by the application, into a dataspace or local memory, JCL changes may be required so that the API has access to the databases and/or files needed to load the tables.

In all cases, when a table is to be loaded into memory (dataspace or local), the API automatically detects what type of table to load based on the format of the table name. It will then load the correct I/O module used to read the table's definition and data. It decides what type of table based on the following specifications:

DB2 Tables - The table name can be from 3 to 36 characters and must be a valid DB2 table, alias or view. In addition, it must contain the AUTH-ID (eg. creator id of the table) followed by a period. For example, SSI.TALBLE1 or USER.RATETAB are valid names.

TABLES/MS Tables - The table name must be 1 to 8 alpha-numeric characters. For example, TABLE1 and RATETAB are valid names. The first character must be alphabetic. TABLES/MS tables are the only table names that do not have a period in them.

User VSAM Tables - The table name is 6 to 36 characters and must be formatted as follows:

VSAM.table-name

where 'VSAM.' is required to distinguish it from a DB2 table and 'table-name' is the name of a table defined into TABLES/MS. The table-name may be the DD name of the VSAM file or the FCT name from CICS for online transactions. Refer to the TABLES/MS Reference Guide for more information on defining VSAM tables.

Compatibility with TABLES/MS TABLP Modules

The TABLP modules (eg. DLITABLP, VISTABLP, etc.) that came with the TABLES/MS package will still work correctly. However, they do not interface or communicate to the TABLES/MM API or dataspaces. Therefore, to allow older applications to continue to work and use dataspaces, the compatible TABLP modules or the routines they call are included with TABLES/MM. By simply including the TABLES/MM load library before the TABLES/MS library in any STEPLIB of JCL that uses TABLES/MS, the new routines will be invoked. As a result, all processing is handled by TABLES/MM as if the applications were calling it directly. However, when implementing TABLES/MM this was, the following should be taken into consideration:

- All tables must be processed from either local memory or dataspaces. By using the TBLDID DD card, only local memory is used, otherwise a dataspace is used. Since the TABLES/MS routines have different parameters, they can not specify the LOC-FLAG to use local memory for some tables and dataspaces for others.

- If loading tables from applications, the FREE SPACE options can not be used.

- The TABLES/MS TABLP modules are less efficient than the TABLES/MM API. The API is more streamlined and requires less overhead.

- The TABLES/MM API is a consistent interface no matter how it's called. The parameters are the same no matter what program calls it, making it easier to use.

Overall, it is recommended that any new applications or old ones being changed be converted to use the TABLES/MM interface rather than any of the TABLES/MS modules.

Online Facilities

The Online Facilities can be used for monitoring and controlling TABLES/MS tables that are loaded in a dataspace. However, TABLES/MS tables can not be loaded with the online facilities and the Effectivity Definition can not be used. These features are limited to DB2 tables and will not work with TABLES/MS tables. However, once TABLES/MS tables are loaded into a dataspace, they can be browsed, freed or selected.

Batch Utilities

The Batch Utilities, TMMUTIL and TMMRPT are fully compatible and support TABLES/MS tables. TMMRPT will list all TABLES/MS tables in all dataspaces in the same way as for DB2 tables. Once in a dataspace, no distinction is made between the tables.

The TMMUTIL utility will also work with any TABLES/MS supported tables. The only difference is JCL used to run it. The installation JCL library has several sample JCL members that can be used to process TABLES/MS tables in the same way as DB2 tables are processed. All control codes are supported except SFLD (TABLES/MS tables are always sorted by the Sequence Field). The following sample JCL members are supplied:

JCLUTILB - Executes TMMUTIL using IMS BMP processing. This allows TABLES/MS-IMS tables and DB2 tables to be loaded.

JCLUTILD - Executes TMMUTIL using IMS DLI processing. It calls the DB2 program to allow either TABLES/MS-IMS or DB2 tables to be loaded.

JCLUTILI - Executes TMMUTIL using IMS DLI processing with no DB2 support. This allows just TABLES/MS-IMS tables to be loaded.

JCLUTILV - Execute TMMUTIL using the standard TMMUTIL proc with the addition of TABLES/MS-VSAM DD cards. This allows TABLES/MS for CICS and user VSAM files to be loaded.

Application Programming Interface

As recommended above, the TABLES/MM API should be used in place of the TABLE/MS pre-load interface routines. The API fully supports all TABLES/MS tables (IMS, CICS, VSAM), is more efficient, and has more capabilities. Calling the API for TABLES/MS tables is exactly the same as shown in Section 5 with one exception. For applications that will be loading TABLES/MS IMS tables either explicitly or implicitly (using auto-load), then the four TABLES/MS IMS PSB's must be passed (the IMS LT-PCB is no longer required). The call format in this case would look like the following:

```
CALL  TMM-INT-MODULE  USING  INTERFACE-CONTROL-AREA,
                             INTERFACE-RECORD-IO-AREA,
                             INTERFACE-SORT-AREA,
                             PCB-1, PCB-2, PCB-3, PCB-4.
```

Where PCB1 through PCB4 are the TABLES/MS pcb's and the same as passed to the TABLP routines (excluding the LT-PCB). For TABLES/MS VSAM tables and any call that will not be loading tables, the standard call can be used. Also, the SORT-AREA is still optional in all cases depending on the ICA-SORT-FLAG field.

Appendix E - Reason Codes

The TABLES/MM API, TMMINT, returns a reason code in the Interface Control Area after each request. Currently, only those for the LOAD and Transient functions have been defined. The following is a list of each reason code, a description of what causes it, and possible actions to take.

LOAD Function Reason Codes

004 - The table or view was not found.

Action: Make sure the name is spelled correctly. If not a DB2 table, make sure it is correctly defined to TABLES/MM as a table or view.

201 - A getmain failed trying to initially allocate memory for the table. Tables are loaded into the local region before being moved to a dataspace.

Action: Make sure the region size is large enough.

202 - The connection to DB2 could not be made. This should only occur in batch or TSO when not running under the DSN command (eg. using DB2 Call Attach).

Action: Make sure the correct DB2 plan is specified and that you have authority to execute it and check to make sure that the plan was bound correctly.

203 - A view could not be created in the dataspace or local memory.

Action: Check the dataspace to make sure there is enough directory entries and space available. Also, possibly a memory problem. Make sure the region size is large enough.

204 - The WHERE Clause passed on a LODW function was too large. The maximum size allowed is 1000 bytes.

Action: Make sure the length is correctly set for the WHERE clause. The length is a two byte area preceding the where clause. Also, make sure the parameters passed to TMMINT are correct.

205 - A getmain failed while loading the table. For large tables, memory is reallocated after a certain size.

Action: Make sure the region size is large enough.

206 - A getmain failed while building an index.

Action: Make sure the region size is large enough.

207 - While building a unique index, a duplicate entry was found.

Action: Check the index to make sure it was defined correctly and check the table to make sure it does not have any incorrect rows.

208 - The table could not be loaded into the dataspace.

Action: Check the dataspace to make sure there is enough directory entries and space available. Also, make sure the region size is large enough.

220 - Error trying to load an I/O interface module. Only occurs for TABLES/MS tables. If a table is not found in the DB2 catalog or in the MS_DEFINITION table, then a check is made to see if it is an old style MS table.

Action: If loading an MS table, make sure the TABLES/MM load library is available to the job. Also, check to make sure the table name is correct.

221 - No IMS PCB's were passed and attempting to load a TABLES/MS table. (See reason code 220 for additional information).

Action: If loading an MS table, make sure the PCB's are passed and the job is run under the IMS region controller. Also, check to make sure the table name is correct.

222 - The table was not found or an error occurred trying to get the TABLES/MS definition for the table being loaded.

Action: Check to make sure the table exists. If loading an IMS TABLES/MS table, make sure the correct PCB's are passed and the databases are available. If loading a VSAM TABLES/MS table, make sure the TBLDD06 - 09 files are allocated correctly and are available (eg. not allocated and open to CICS).

223 - A getmain failed trying to allocate memory for the TABLES/MS table.

Action: Make sure the region size is large enough.

226 - Error trying to read a TABLES/MS record.

Action: The **ICA-PROGRAM-NAME field passed in the ICA must be non-blank**. This is required by TABLES/MS processing. Also, if loading an IMS TABLES/MS table, make sure the correct PCB's are passed and the databases are available. If loading a VSAM TABLES/MS table, make sure the TBLDD06 - 09 files are allocated correctly and are available (eg. not allocated and open to CICS).

301 - Internal error trying to access MS_DEFINITION.

Action: Call technical support.

302 - SQL Error trying to read the MS_DEFINITION table.

Action: Check the SQL CODE for the cause of the problem.

401 - SQL Error trying to read VIEW definition records from MS_DEFINITION.

Action: Check the SQL CODE for the cause of the problem.

402 - SQL Error trying to read VIEW column records from MS_DEFINITION.

Action: Check the SQL CODE for the cause of the problem.

403 - The view being loaded is inconsistent with the base table.

Action: Make sure the table has not changed since the view was defined. List the view using TMMDEFN and compare the columns to the base table. Redefine the view to match the table.

404 - A column in the view being loaded does not match any column in the base table.

Action: Make sure the table has not changed since the view was created. List the view using TMMDEFN and compare the columns to the base table. Re-create the view to match the table.

405 - SQL Error trying to read an EFF record from MS_DEFINITION.

Action: Check the SQL CODE for the cause of the problem.

406 - SQL Error trying to read an ORD record from MS_DEFINITION.

Action: Check the SQL CODE for the cause of the problem.

407 - The Break-in or Break-out column defined in the effectivity definition record was not found in the table.

Action: Make sure the table has not changed since effectivity was defined. Using the on-line System, check the B/I and B/O column names to make sure they are correct. Re-define effectivity for the table.

408 - SQL Error trying to read INDEX records from MS_DEFINITION.

Action: Check the SQL CODE for the cause of the problem.

409 - SQL Error trying to read INDEX columns from MS_DEFINITION.

Action: Check the SQL CODE for the cause of the problem.

410 - A column in the index for the table being loaded does not match any column in the table.

Action: Make sure the table has not changed since the index was created. List the indexes for the table using TMMDEFN and compare the columns to those in the table. Re-create the index to match the table.

411 - An error occurred trying to setup the key compare instructions for an index.

Action: Check to make sure the region size is large enough.

412 - SQL Error trying to read table column records from MS_DEFINITION.

Action: Check the SQL CODE for the cause of the problem.

413 - SQL Error trying to build the internal table definition when processing the DB2 catalog or the MS_DEFINITION table.

Action: Check the SQL CODE for the cause of the problem.

414 - The base table for the view being loaded was not found.

Action: Make sure the view is defined correctly and that the table it is associated with is correct and exists. This should only occur if a base table is deleted after a view was defined.

451 - Internal Error trying to load a VSAM table.

Action: Call Technical Support.

452 - Error trying to create an ACB for a VSAM table.

Action: Make sure the region size is large enough or call Technical Support.

453 - Error trying to create an RPL for a VSAM table.

Action: Make sure the region size is large enough or call Technical Support.

454 - Error trying to open a VSAM file.

Action: Make sure the VSAM file is correctly allocated to the job. The DDNAME for the file should be the same as the table name suffix. Also, make sure the file allocated is a valid VSAM KSDS.

455 - Internal Error trying to load a VSAM table.

Action: Call Technical Support.

456 - Internal Error trying to load a VSAM table.

Action: Call Technical Support.

457 - A VSAM Logical Error occurred reading a VSAM file or a CICS READNEXT failed.

Action: Make sure the VSAM file is correctly allocated and is a valid VSAM KSDS dataset. For CICS make sure the file is available and defined correctly.

458 - A VSAM Physical Error occurred reading a VSAM file.

Action: Make sure the VSAM file is correctly allocated and is a valid VSAM KSDS dataset.

459 - An Unknown Error occurred reading a VSAM file.

Action: Make sure the VSAM file is correctly allocated and is a valid VSAM KSDS dataset.

461 - A CICS Inquire failed for the VSAM file being loaded.

Action: Make sure the VSAM file is correctly defined to CICS and that the user loading the table is authorized to access it.

462 - The VSAM file being loaded was not defined to CICS.

Action: Make sure the VSAM file is correctly defined to CICS.

463 - When loading a VSAM file under CICS, the file was not open and an attempt to open it failed.

Action: Make sure the VSAM file is correctly defined to CICS and that the user loading the table is authorized to access it. Also, make sure the file has not been disabled and that it is not in use by another job.

464 - A CICS STARTBR failed when trying to read a VSAM file.

Action: Make sure the VSAM file is correctly defined to CICS to allow BROWSE and that the user loading the table is authorized to access it.

471 - Internal Error trying to execute dynamic SQL.

Action: Call Technical Support.

472 - SQL Error trying to prepare a dynamic SQL statement.

Action: Check the SQL CODE for the cause of the problem. For a LODW function, make sure the WHERE clause is correct.

473 - SQL Error trying to execute a dynamic SQL statement.

Action: Check the SQL CODE for the cause of the problem. For a LODW function, make sure the WHERE clause is correct.

474 - SQL Error trying to open a cursor to fetch the DB2 table.

Action: Check the SQL CODE for the cause of the problem. For a LODW function, make sure the WHERE clause is correct.

475 - SQL Error trying to fetch a row.

Action: Check the SQL CODE for the cause of the problem. For a LODW function, make sure the WHERE clause is correct.

476 - Error trying to process the columns for the DB2 table.

Action: Check the DB2 table to make sure it was defined correctly and supported by TABLES/MM or call Technical Support.

Transient Function Reason Codes

604 - The table specified is not currently loaded into memory.

Action: Make sure the name is spelled correctly. Check the dataspace-id on the TBLDID DD card in the JCL to make sure the correct dataspace is being accessed or if processing a local table, make sure the table has been loaded.

605 - Invalid table for TMAK.

Action: When using TMAK, the table name can not reference a view or alias (simple view) of a table. Also, make sure the table has not been freed from the dataspace or local memory.

610 - TADD failed because there is no free space for the specified table.

Action: If adding records, then a free space value must be specified in the ICA on the TMAK request issued prior to the TADD.

611 - TUPD failed because the record was not found.

Action: When updating a record, the first (old) Record IO area passed to TMMINT must contain the exact record to be updated. Use a GET request to find the correct record and return it on the TUPD function.

612 - TDEL failed because the record was not found.

Action: When deleting a record, the Record IO area passed to TMMINT must contain the exact record to be deleted. Use a GET request to find the correct record and return it on the TDEL function.

613 - Transient Function specified but no prior TMAK.

Action: A TMAK function must be executed prior to any other Transient function. Make sure a TMAK is issued and the result code returned was OK.

Negative Reason Codes

A negative reason code means that the reason code field in the ICA has two values in it. The first two bytes represent an SQL code and the last two represent the actual reason code. Refer to the description of the ICA in Section 5.

For the SQL code, the value is a result of a call to DB2 or a VSAM access method request. If the value is from -9000 to -9999, the SQL code represents an error when processing a VSAM file in batch or through CICS. Otherwise, the value is a DB2 SQL code and should be checked in the DB2 Messages and Codes manual.

For VSAM errors, only the two right-most digits are meaningful (eg. ignore the -90). In the case of batch processing, the two digits represent the COBOL or COBOL II file status code. Refer to the appropriate COBOL manual for a description of what the value means. In the case of CICS processing, the two digits represent the CICS EIBRESP value. Refer to the appropriate CICS manual for a description of the possible EIB response values.

INDEX

9

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W