SUBJECT

> Description of the Multics Transaction Processing Subsystem for Performing Data Base Related Operations

SPECIAL INSTRUCTIONS

> This manual is a preliminary edition which describes a basic transaction processing capability for use on a Multics system.

SOFTWARE SUPPORTED

> Multics Software Release 7.0

PREFACE


Transaction processing in the Multics system can be performed by the subsystem described in this manual. By employing the transaction processing (TP) subsystem, the individual user can process transactions against an extensive database by invoking a set of commands defined by the site. This manual describes the Multics TP subsystem, describes the administrative commands and their usage, and furnishes the practical details of subsystem operation.


Although most TP users will not need to avail themselves of the many other facilities of the Multics system, additional information regarding Multics software concepts and organization as well as specific usage of Multics commands and subroutines can be obtained from the volumes of the Multics Programmers' Manual (MPM). These volumes are:

MPM Reference Guide, Order No. AG91

MPM Commands and Active Functions, Order No. AG92

MPM Subroutines, Order No. AG93

MPM Subsystem Writers' Guide, Order No. AK92

MPM Peripheral Input/Output, Order No. AX49

MPM Communications Input/Output, Order No. CC92

CONTENTS

ILLUSTRATIONS

SECTION 1

INTRODUCTION

## MULTICS TRANSACTION PROCESSING

The Multics transaction processing (TP) subsystem provides a specialized environment for applications requiring much activity with a few centralized databases. Within the subsystem only a limited, well-defined set of simple commands are available. Most of these are application programs that interact with a database. The TP user is isolated from the general Multics interactive environment. In this manual, a user is a person who enters transactions from a terminal. He is not expected to know about computers. People who write the application programs are Multics users but not necessarily TP users. Many aspects of the subsystem are table-driven or specified in modifiable modules, so that features can easily be added, changed or deleted. Several independent TP subsystems may run concurrently on a single Multics system.

This manual describes how the Multics TP subsystem is organized, how to run it, how to meter it, and how to write application programs for it. However, both administrators and application programmers may still need to consult various volumes of the MPM, as well as other appropriate manuals for further information concerning the Multics system and its use. Specific TP user requests are site-dependent and therefore are not included.

## LIST OF FEATURES

Features available in the Multics TP subsystem include the following:

- locking of individual database records to allow concurrent database access

- commitment and rollback facilities which allow handling of both concurrency control and restoration after an error or other interruption

- availability of most Multics languages for application programs

- modular construction to facilitate tailoring of the system

- separation of input/output processing and computation to allow better tuning

- a deadline mechanism for scheduling transactions

- easily established parallel environments for testing application programs

For the purpose of discussion within this manual, a transaction is defined as the processing of a single user command, from receipt of the input message by the TP subsystem to execution completion. A transaction processing routine (TPR) is an application program. It usually is one that references a large database.

A transaction is a series of modifications to a database that appear to occur atomically and simultaneously to other processes. To ensure that all database changes will appear at the same time, the changes made by a TPR are not visible to other processes until the transaction is finished. When a transaction is finished, the TP subsystem automatically performs an operation called a commitment. A commitment means that this transaction is finished, and all changes to the database since the last commitment should be made visible to other processes. This scheme ensures database consistency. Occasionally a commitment by one process changes data that a transaction in another process depends upon. This is called an asynchronous change. The second process may then have an inconsistent view of the database. A commitment by the second process under these circumstances will fail. When this happens, the TP subsystem rolls back the transaction. This means that all changes made to the database since the last commitment are erased. The transaction usually will be retried.

The Multics TP subsystem is organized to make efficient use of resources by taking advantage of the special characteristics of TP. Typically, many users are all performing similar operations on a fixed set of databases, using a closed environment of their own rather than the standard Multics command environment.

Allocating a process per user would entail much duplicated address space overhead and would be difficult to schedule efficiently. Therefore, the TP subsystem's processes are organized differently. They are divided according to function into the following three types:

● Master process--coordinates and communicates with the other processes

● I/O process--manages input from and output to terminals

● Worker process--executes the application programs

A TP subsystem consists of one master process, one or more I/O processes, and one or more worker processes.

The I/O and worker processes communicate with each other via the input and output queues as well as through a shared table. The input queue contains commands to be processed, and the output queue contains output from TPRs. Figure 1-1 shows a simplified diagram of the relationships of the various processes. These processes are described in more detail below.
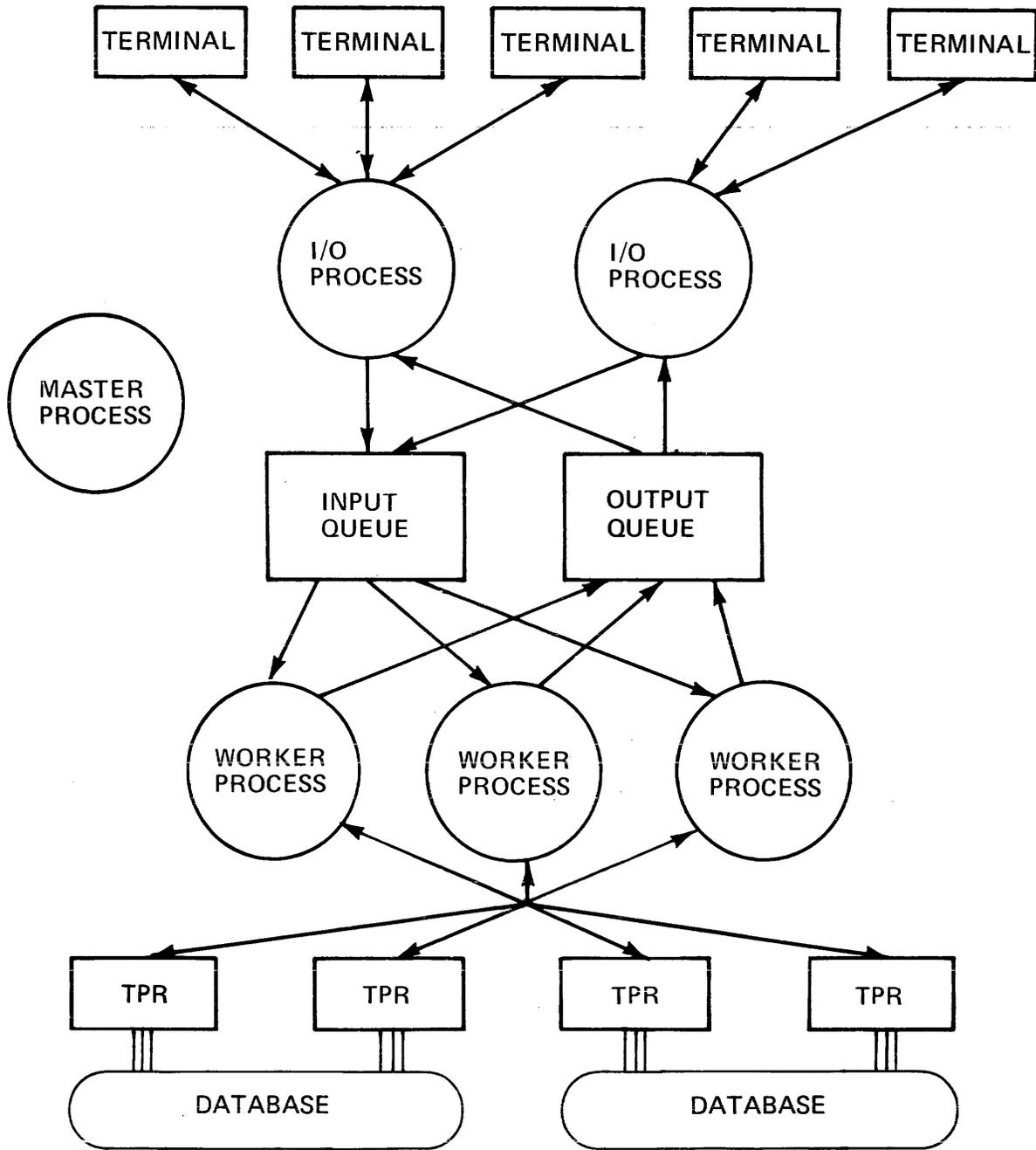
Figure 1-1.  Structure of the Multics TP Monitor

The master process is used by the TP administrator to begin transaction processing. First the master table is initialized. Then the master process logs in all of the other processes. No explicit action is taken if the TP subsystem is being restarted after an abnormal interruption. In that case, any database modifications that were incomplete are automatically rolled back as they are encountered in normal processing. The master process is responsible for coordinating TP shutdown. This includes ensuring that all transactions in progress are completed.

An I/O process handles input/output for a group of terminals. The lines handled may be dedicated or attached to the I/O process by the dial facility. This process also queues and schedules input messages. Transactions are scheduled by deadlines. A deadline is a date-time associated with the transaction. Of two transactions in the input queue at the same time, the one with the earlier deadline will be started first. No attempt is made to execute a transaction by the time indicated as its deadline. Some commands that require little processing and do not involve database references may be executed by the I/O process. These are called immediate commands.

A worker process executes one transaction at a time. Output messages to be printed on the user's terminal are placed into the output queue. Databases are accessed through the standard file I/O module, vfile_, in a mode that allows changes to reversed if a transaction is interrupted because of an error or system crash. Also, if a transaction fails because of a concurrent update to the database by another process, the changes are rolled back and the transaction is re-executed. All databases to be referenced are attached and opened at the beginning of the process to reduce overhead for individual transactions.

SECURITY

I/O and worker processes may have different User_ids so that Multics access control may be used to restrict database access to worker processes. Individual users, however, are not logged into Multics while using the TP subsystem; in fact, they need not be registered Multics users at all. They only need to be registered as TP users by the TP administrator. Signing on to the TP subsystem is handled by the I/O processes and involves only the TP subsystem's person-name table. Since the set of commands users can invoke is restricted (e.g., editors or compilers are generally excluded), individual TP users are unable to perform malicious acts.

USER INTERACTION

The TP user types input request lines, which cause TPRs to be run. The TPR's output is printed on the terminal. The input line is read by an I/O process and the TPR is executed in a worker process. Figure 1-2 shows conceptually how a transaction is processed. The transaction input is read from the terminal and placed into the input queue. Some worker process then finds the transaction in the input queue and invokes the appropriate TPR. Any output written by the TPR onto the user_output I/O switch is placed into the output queue. An I/O process then prints output from the output queue on the user's terminal.
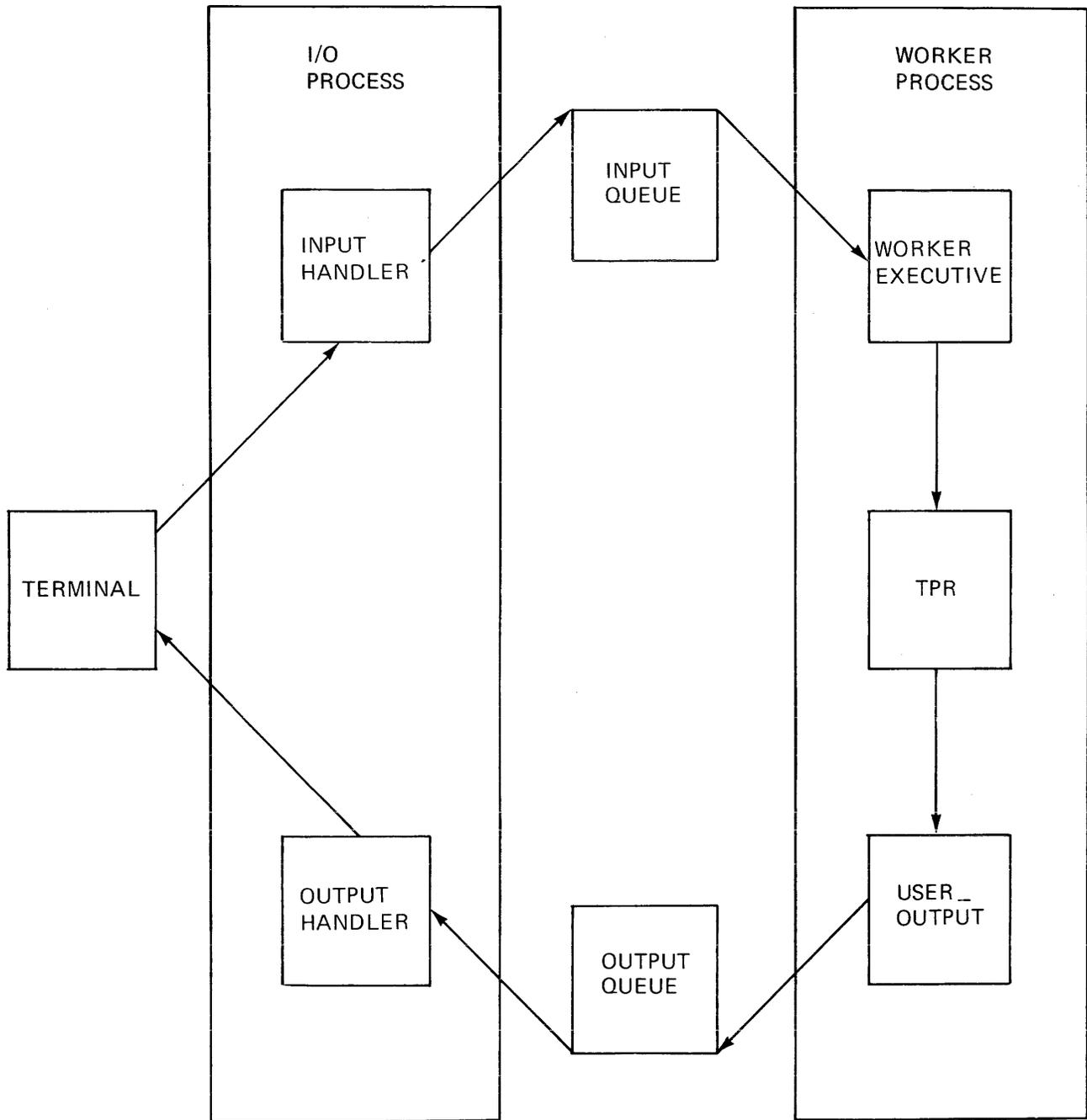
Figure 1-2. How a Transaction is Processed

A user may wait for the complete output from a transaction to be printed before typing the next transaction. However, the TP subsystem does not enforce serial execution. If the user enters transactions without waiting for previous ones to complete, the following things may happen. The output from an immediate command usually appears before output from any other pending transaction. This is because immediate commands are executed as soon as they are read rather than queued for a worker process. The output from a regular transaction might not be contiguous and might appear before the output from a transaction entered earlier. Aside from deadline considerations, this may occur because the TP subsystem can execute more than one transaction from a single user simultaneously. Although execution of transactions is in deadline order, a later transaction may finish before an earlier transaction. In addition, the output messages are queued as soon as they are generated, not when the TPR finishes.

Since output messages from one transaction may be interspersed with output messages from other transactions, each output message is labeled with the transaction number. The transaction number is printed when a transaction is queued.

SECTION 2


RUNNING AND OPERATION



A Multics TP subsystem is defined by a directory and the control segments contained in it. Following is a list and brief description of the control segments.


| Control Segment | Description |
| --- | --- |
| tp_master_table_ | contains most of the dynamic information used by the TP processes. This includes information about each terminal, each process, and interprocess communication information. |
| tp_command_table_ | lists the valid command names for the TP subsystem and their associated attributes. See the description of the tp_cvsct command. |
| tp_init_database.ec | contains commands to open and initialize MRDS databases. This segment is needed only if MRDS is used. |
| tp_person_name_table_ | contains the names of registered TP users and their encrypted passwords. See the description of the tp_user command. |
| tp_start_up.ec | starts transaction processing and starts up the I/O and worker processes via Multics enter_abs_request commands. |
| worker_start_up.absin | is the worker process absentee control segment which initializes the non-MRDS databases to be used by the worker and then turns the process into a worker. |
| io_start_up.absin | is the I/O process absentee control segment which turns the process into an I/O process, passing it slave channel names and/or a dial_id. |
| tp.tcf | is the transaction control file and contains information indicating whether database operations have been committed. See Section 4. |
| tp.tpinq | is the input queue and contains, for each queued transaction, its input command line, meters, information about its output destination, and its current state. |
| tp.tpoutq | is the output queue and contains the transaction output messages to be displayed on terminals. |

SETTING UP A TP SUBSYSTEM

The following discussion presents the considerations required to set up a TP subsystem, and lists the necessary procedural steps. To illustrate each step, an example of a typical transaction processing situation, called the Sample_TP project, is also included. In this discussion, reference is made to access control lists (ACLs) and setting of access to segments. See "Access Control" in the MPM Reference Guide for a discussion of the various kinds of access.

## The TP Administrator

TP administrator is the term used in this manual for the person who performs the following tasks:

● creating subsystem directories and tables

● starting, stopping, and monitoring transaction processing

● making sure databases are consistent after crashes

## User ids for a TP Subsystem

All I/O processes should use one given User_id. Likewise, all worker processes should use some other User_id.

The master, I/O, and worker processes should all have User_ids different from each other for security reasons. The Sample_TP project has the following User_ids:

| Process | User id |
|---------|---------|
| Master Process | Master.Sample_TP |
| I/O Processes | IO.Sample_TP |
| Worker Processes | Worker.Sample_TP |
| TP administrator | PCJones.Sample_TP |

## TP Process Names

Each I/O and worker process is given a unique name to identify it within the TP subsystem. This name is independent of the User_id of an I/O process or a worker process. The process name is also the first component of the absentee control segment for an I/O or worker process. In the Sample_TP project, there will be three I/O processes named IO_1, IO_2 and IO_3. There will be two worker processes named Worker_1 and Worker_2.

Following is a list of the steps necessary to establish a TP subsystem:

1. Register the User_ids for the TP administrator, master, I/O and worker processes. The I/O and worker processes' User_ids should have max_foreground set to the maximum number of concurrent I/O and worker processes, respectively. If the dial facility is used, the I/O process User_id should be given the dialok attribute. (See the Multics Administrators' Manual -- Project Administrator, Order No. AK51, for information about registering User_ids.) For the Sample_TP project, the following User_ids are registered: PCJones.Sample_TP, Master.Sample_TP, IO.Sample_TP and Worker.Sample_TP.

2. Arrange to have the master process be given execute (e) access to the proxy ACS. This enables it to log in absentee processes for the other processes. For the Sample_TP project, the system administrator issues the following command:

   sa >system_control_1>proxy>absentee_proxy.acs e Master.Sample_TP

3. Create the TP subsystem's directory. This is the directory in which all the control segments reside. The master process and TP administrator should have sma access to this directory. The I/O and worker processes should not have modify access, but should have status and append access to it. In the Sample_TP project, the TP administrator issues the following commands:

   create_dir >udd>Sample_TP>tp_dir
   change_wdir >udd>Sample_TP>tp_dir
   set_acl [wd] sma Master.Sample_TP sa IO.Sample_TP
     sa Worker.Sample_TP

   The following commands assume the working directory is >udd>Sample_TP>tp_dir.

4. Create a subdirectory for the commands. This must be named "commands". The I/O and worker processes should have s access to it. The master process should have sma access. In the Sample_TP project, the TP administrator issues:

   create_dir commands
   set_acl commands sma Master.Sample_TP s IO.Sample_TP
     s Worker.Sample_TP

5. Create the master table with appropriate access. All processes should have rw access to it. The TP monitor will enlarge it. For the Sample_TP project:

   create tp_master_table_
   set_acl tp_master_table_ rw Master.Sample_TP rw IO.Sample_TP rw
     Worker.Sample_TP

6. Create the transaction control file (TCF) with appropriate access. All processes should have rw access to it. The tp_pre_create exec_com makes sure the transaction control file is a multisegment file. The TCF should never be deleted.

   ec >unb>tp_pre_create tp.tcf
   set_acl tp.tcf rw Master.Sample_TP rw IO.Sample_TP rw
     Worker.Sample_TP

7.  Create the input queue with appropriate access.  All processes should
    have rw access to it.  For the Sample_TP project:

```
ec >unb>tp_pre_create tp.tpinq
set_acl tp.tpinq rw Master.Sample_TP rw IO.Sample_TP rw
    Worker.Sample_TP
```

8.  Create the output queue with appropriate access.  All processes should
    have rw access to it.  For the Sample_TP project:

```
ec >unb>tp_pre_create tp.tpoutq
set_acl tp.tpoutq rw Master.Sample_TP rw IO.Sample_TP rw
    Worker.Sample_TP
```

9.  Create the I/O process absentee control segment, io_start_up.absin, in
    the TP directory.  Each I/O process should  have r access to it.  For
    each I/O process name,  IO_process_name, io_start_up.absin should have
    the  additional  name  IO_process_name.absin.  The  absentee  segment
    requires  one  argument, the  pathname of  the TP  directory.  The I/O
    process  absentee control  segment should be  similar to the following
    one for the Sample_TP project:

```
change_wdir &1
&
& To protect this process against being logged out due to
& inactivity, uncomment the next three lines.
& memo -pathname &ec_name -on
& memo -delete -match nothing
& memo -time "20 minutes" -repeat "20 minutes" -alarm -call
    nothing
&
&goto &ec_name
&
&label IO_1
tp_io_start &ec_name [wd] channel1 channel2 channel3
&quit
&
&label IO_2
tp_io_start &ec_name [wd] -registered_dial Sample_TP_system
&quit
&
&label IO_3
tp_io_start &ec_name [wd] channel4 channel5 -dial TP_system
&quit
```

10. Create  tp_init_database.ec.  This  segment  is  needed  only  if  the
    Multics  Relational Data  Store (MRDS)  is used.  Each  worker process
    must have r access to  this segment.  This  exec_com contains commands
    to prepare a MRDS  database for use.  It is  invoked when the run unit
    in which TPRs  execute is  started.  The  following is the format of a
    typical tp_init_database.ec.

```
& Prepare MRDS databases
mrds_call open dsm_paths
mrds_call set_tcf_switch [mrds_get_dbi (dsm_paths)] tcf_
mrds_call ready_file [mrds_get_dbi dsm_path1]
        file_name1 rdy_mode1 ... file_namei rdy_modei
mrds_call set_collection_delay_time [mrds_get_dbi dsm_path1]
        file_name1 delay_time1
    .
    .
    .
mrds_call set_collection_delay_time [mrds_get_dbi dsm_path1]
        file_namei delay_timei
    .
    .
    .
mrds_call ready_file [mrds_get_dbi dsm_pathn]
        file_name1 rdy_mode1 ... file_namej rdy_modej
```

```
        mrds_call set_collection_delay_time [mrds_get_dbi dsm_pathn]
                file_name1 delay_time1
        .
        .
        .
        mrds_call set_collection_delay_time [mrds_get_dbi dsm_pathn]
                file_namej delay_timej
        &quit
```

To ensure repeatability if an asynchronous change occurs, only the monitor_retrieve or update ready modes should be used. The collection delay time should be longer than the real time any transaction that uses the file will require. The following is tp_init_database.ec for the Sample_TP project:

```
        & Prepare MRDS databases
        mrds_call open <databases>people_db <databases>inventory
        mrds_call set_tcf_switch [mrds_get_dbi (<databases>people_db
                <databases>inventory)] tcf_
        mrds_call ready_file [mrds_get_dbi <databases>people_db]
                people monitor_retrieve
        mrds_call set_collection_delay_time [mrds_get_dbi <databases>
                people_db people 90]
        mrds_call ready_file [mrds_get_dbi <databases>inventory]
                inventory update
        mrds_call set_collection_delay_time [mrds_get_dbi <databases>
                inventory inventory 300]
        &quit
```

11. Create the worker process absentee control segment, worker_start_up.absin, in the TP directory. Each worker process should have r access to it. For each worker process name, worker_process_name, worker_start_up.absin should have the additional name worker_process_name.absin. The absentee control segment requires one argument, the pathname of the TP directory. All segments in the commands directory are explicitly initiated to reduce overhead. All files accessed through language I/O facilities or through iox_ directly should also be attached and opened with io_call to eliminate this overhead from TPRs. The worker process absentee control segment should be similar to the one for the Sample_TP project:

```
        change_wdir &1
        initiate commands>([segments commands>**] [links commands>**])
                -all -force
        tp_worker_init_tcf [wd]
        &
        & Attach and open I/O switches
        io_call attachparts vfile_ <databases>parts -stationary
                -transaction tcf_ -share
        io_call open parts keyed_sequential_update
        io_call control parts set_wait_time -collection_delay_time 300
        &
        & To protect this process against being logged out due to
        & Inactivity, uncomment the next three lines.
        & memo -pathname &ec_name -on
        & memo -delete -match nothing
        & memo -time "20 minutes" -repeat "20 minutes" -alarm -call
                nothing
        &
        tp_worker_start &ec_name [wd]
        &quit
```

For a file that is attached above to be protected by the commitment mechanism, the vfile_ attach description must include "-stationary -transaction tcf_ -share". The collection delay time should be longer than the real time any transaction that uses the file will require.

12. Create the TP subsystem's start_up exec_com, tp_start_up.ec. The master process should have r access to it. The exec_com requires one argument, the pathname of the TP directory. The tp_start_up.ec segment should be similar to the one for the Sample_TP project:

```
&command_line off
&if [nequal &n 1]
&then &goto start
&print Usage: ec tp_start_up tp_dir
&quit
&
&label start
change_wdir &1
answer yes -brief do """rename &(1) [strip_entry &(1)].old.absout"""
    ([segments *.absout])
tp_start [wd]
&
ear IO_1 -foreground -proxy IO.Sample_TP -arguments [wd]
ear IO_2 -foreground -proxy IO.Sample_TP -arguments [wd]
ear IO_3 -foreground -proxy IO.Sample_TP -arguments [wd]
&
ear Worker_1 -foreground -proxy Worker.Sample_TP -arguments [wd]
ear Worker_2 -foreground -proxy Worker.Sample_TP -arguments [wd]
&quit
```

3. Create tp_command_table_. See the description of the tp_cvsct command. The master, worker and I/O processes should have r access to it. This segment lists all commands, including I/O process immediate commands, that TP users may type.

4. Write the transaction processing routines (TPRs). See Section 6. Put all TPRs, or links to them, in the "commands" directory contained in the TP directory. I/O processes must have re access to commands that they can execute. Worker processes must have re access to commands that they execute.

5. Create tp_person_name_table_. The master process should have rw access to it, the worker and I/O processes should have r access to it. For the Sample_TP project:

```
create tp_person_name_table_
set_acl tp_person_name_table_ rw Master.Sample_TP
    r IO.Sample_TP r Worker.Sample_TP.
```

6. Prepare and distribute TP user documentation.

7. Use the tp_user command to enter TP users and their passwords into tp_person_name_table_.

8. Create all databases needed by the worker processes. All keyed sequential vfiles must be multisegment files before transaction processing is started. The tp_pre_create exec_com creates an empty multisegment keyed sequential vfile. Worker processes must have access to the databases they will use. See the MRDS Reference Manual, Order No. AW53, for information about initializing MRDS databases.

## A Complete TP Subsystem

After setting up the TP subsystem for the Sample_TP project, a listing of the TP directory is as follows:

```
Segments = 7, Lengths = 9.

r w    3   tp_person_name_table_
r w    2   tp_command_table_
r w    1   tp_start_up.ec
r w    1   worker_start_up.absin
           Worker_1.absin
           Worker_2.absin
r w    1   tp_init_database.ec
r w    1   io_start_up.absin
           IO_1.absin
           IO_2.absin
           IO_3.absin
r w    0   tp_master_table_

Multisegment-files = 3, Lengths = 21.

r w    7   tp.tpoutq
r w    7   tp.tpinq
r w    7   tp.tcf

Directories = 1.

sma    commands
```

## OPERATING A TP SUBSYSTEM

This section describes how to run a TP subsystem once it has been set up. It also describes how to attach terminals to the TP subsystem.

## Starting Transaction Processing

To start transaction processing, log in the master process and issue the following command in the master process:

    ec tp_start_up tp_dir

where tp_dir is the pathname of the TP directory. For the Sample_TP project the command would be:

    ec tp_start_up >udd>Sample_TP>tp_dir

In tp_start_up.ec, the tp_start command turns the process into a master process. The I/O and worker processes are then logged in.

## Managing Transaction Processing

If another I/O process or worker process is required after transaction processing has started, an enter_abs_request command similar to the ones in tp_start_up.ec can be issued in the master process. The I/O or worker process name of the new process must not be the same as an existing process's name.

If the master process should be destroyed while transaction processing is running, invoke the tp_start command with the -new_master control argument to establish another process as the master process. Periodically a program automatically runs in the master process and makes sure all the I/O and worker processes still exist. If one has been destroyed, a message is printed and an enter_abs_request command is automatically issued to restart it. If an I/O process is destroyed and restarted, all TP users that were connected to it must sign on again. Check the I/O and worker process's absentee output segments for error messages.


## Shutting Down Transaction Processing


Transaction processing is shut down by issuing the tp_stop command in the master process; the rest is done automatically in two stages. In the first stage, the I/O processes stop accepting input and the worker processes either finish all pending transactions or, if the -immediate control argument was specified, just finish their current transactions. The worker processes then log out. In the second stage, the I/O processes finish processing all accumulated output and log out. Thus, when a shutdown is complete all the output from completed transactions has been processed. If the -immediate control argument was specified, there may be some transactions left in the input queue to be processed when transaction processing is resumed. A message is printed by the master process when the shutdown has completed.


## Input Queue Maintenance


The input queue may be placed on a different logical volume than the databases used by the TP subsystem. In this case, the TP directory should contain a link to the input queue.

As transactions are entered into the TP subsystem, they are placed into the input queue where worker processes obtain transactions to execute. When a transaction is finished, its entry is not deleted from the input queue. Instead, the entry is changed to indicate that the transaction has completed. The input queue therefore keeps getting larger. Several things can be done to keep the input queue from growing indefinitely:

- The tp_shrink_q command may be used to delete records from the input queue. This may be done while the TP subsystem is running. Records may be copied to any device. Records of successful transactions may be separated from those of unsuccessful ones.

- The input queue may be renamed. This should only be done when the TP subsystem is not running and there are no pending transactions. Using this method, a different multisegment file may be used each day as the input queue. The tp_meters command may be used to obtain statistics from a previous input queue. The old input queues may be deleted or copied to a backup device as needed. A new input queue must be created before resuming transaction processing.

- The input queue may be deleted. This should only be done when the TP subsystem is not running and there are no pending transactions. This method naturally leaves no record of processed transactions. A new input queue must be created before resuming transaction processing.

## Dial and Terminal Use

In Multics TP, each I/O process manages a group of terminals. It is not possible for a particular terminal to be used by more than one process at a time. An I/O process handles all the input and output for its terminals including user signons. Terminals are connected to an I/O process either through the dial facility or through specified slave channels. The I/O process controls the terminals and therefore controls the users' access to Multics and to the TP subsystem. A TP user need not be a registered Multics user.

## Use of the Dial Facility

The Multics dial facility allows several terminals to be attached to the same process. A terminal is connected to an existing process by the dial access request. (See MPM Commands, Section 4 for a description of the dial access request.) In order for an I/O process to accept dials, the tp_io_start command must be given the -dial or -registered_dial control argument. See the description of tp_io_start in Section 3 of this manual.

The dial facility should be used when there are login service type channels that use the TP subsystem, i.e., when some channels need to be able to access the TP subsystem at some times and to log in to Multics at other times.

## Use of Slave Channels

An I/O process may manage channels that are specified when the command tp_io_start is invoked. These channels must be of the slave service type. See the Multics Administrators' Manual -- Communications Order No. CC75, for a description of the channel definition table (CDT), the slave service type, and for instructions on using the slave service type.

## RUNNING TP IN TEST ENVIRONMENTS

Parallel TP subsystems may be used for testing. Setting up a test subsystem requires creating a directory and the necessary control segments, just as for a real TP subsystem. If databases are used in common with another TP subsystem, the TP subsystem's transaction control file must also be used via a link in the test TP directory.

## Establishing Test Processes

Once a test environment has been created, test processes must be set up. Any process with access to the test TP directory and control segments can be used. One way is to use special test Person_ids within the TP project. Once tp_start has been invoked in the test TP subsystem's master process, the I/O and worker processes can be logged in. A process becomes an I/O or worker process by invoking the tp_io_start or tp_worker_start commands, respectively. The test processes can be interactive or foreground absentee processes. However, an interactive worker process will require a terminal that it ignores.

One process may, with restrictions, serve more than one function within the TP subsystem. If one process is serving as both the master process and an I/O process, and the terminal is connected to the TP subsystem, then the command table must contain any desired master process commands as immediate commands.

Combining the master process and a worker process is not recommended because a worker process ignores all input from the terminal. An interactive process may serve as an I/O process and a worker process. In this case, the tp_io_start command must precede the tp_worker_start command, and the terminal may not be connected to the TP subsystem.


Test Mode

   Any user may use a TP subsystem in test mode. Transactions are processed and produce output in test mode but databases are not changed. To enter this mode, use the tp_io_enter_test_mode immediate command. To return to normal TP processing, use the tp_io_exit_test_mode immediate command.

   See Section 3 of this manual for descriptions of the transaction processing commands mentioned in the preceding paragraphs.

SECTION 3


TP COMMANDS AND SUBROUTINES



The commands and subroutines available to the user and TP administrator for
transaction processing  are described in this  section.  They are presented in a
format consistent with that described in the Multics Programmers' Manual (MPM).


The TP commands and subroutines may  be classified according to the context
in which they may be invoked.  The  contexts in which a TP command or subroutine
may be invoked are:

- in the master process

- in the worker process

- in the I/O process

- outside the TP subsystem or in the master process


The commands and  subroutines are described in  four subsections, according
to the context in which they are  invoked.  Within each subsection, the commands
and subroutines are presented  alphabetically.  The following is an alphabetical
list of all the commands and subroutines described in this section together with
designation of the context in which they may be invoked.


| Command or Subroutine | Context |
|---|---|
| tp_cancel | master process |
| tp_change_deadline | master process |
| tp_cvsct | outside TP |
| tp_display_command_table | outside TP |
| tp_display_current_xcns | master process |
| tp_display_input_queue | outside TP |
| tp_display_master_table | outside TP |
| tp_display_output_queue | outside TP |
| tp_get_xcn_status | master process |
| tp_io_cancel | I/O process |
| tp_io_enter_test_mode | I/O process |
| tp_io_exit_test_mode | I/O process |
| tp_io_get_xcn_status | I/O process |
| tp_io_list_pending_requests | I/O process |
| tp_io_signoff | I/O process |
| tp_io_start | I/O process |
| tp_io_who | I/O process |
| tp_list_pending_requests | master process |
| tp_meters | outside TP |
| tp_pre_create.ec | outside TP |
| tp_reset_xcn_num | outside TP |
| tp_rollback_transaction_ | worker process |
| tp_shrink_q | outside TP |
| tp_start | master process |
| tp_stop | master process |
| tp_user | outside TP |
| tp_verify_transaction_ | worker process |

```
tp_who                master process
tp_worker_init_tcf    worker process
tp_worker_start       worker process
```

# MASTER PROCESS COMMANDS

This subsection contains descriptions of commands that may only be invoked in the master process. The master process commands are presented in alphabetical order within this subsection and include the following:

```
tp_cancel
tp_change_deadline
tp_display_current_xcns
tp_get_xcn_status
tp_list_pending_requests
tp_start
tp_stop
tp_who
```

Name:   tp_cancel


  The tp_cancel command removes one or more transactions from the input queue
to prevent them from being  processed.  The  transaction must  not have  started
executing.


Usage


  tp_cancel transaction_nums


where transaction_nums are the numbers of the transactions to be canceled.

Name:   tp_change_deadline


        The  tp_change_deadline  command  changes  the  deadline of  the  specified
transaction.  The transaction must not have started executing.


Usage


        tp_change_deadline transaction_num DT


where:

1.      transaction_num
                is the transaction number of the transaction whose deadline is to be
                changed.

2.      DT
                is  the new  deadline  time of   the  transaction.   It is  a  string
                acceptable to  the  convert_date_to_binary_  subroutine described in
                the MPM Subroutines.

Name:   tp_display_current_xcns, tpdcx


        The   tp_display_current_xcns   command    prints    information   about    the
transactions currently executing.  The  worker  process  name,  transaction  number,
TP command name and TP_user_id are printed.


Usage


        tp_display_current_xcns

Name:   tp_get_xcn_status, tpgxs


     The  tp_get_xcn_status   command  prints  information  about one  or  more
transactions.  If the transaction has  not yet been processed, the TP_user_id of
the TP user who submitted it, the TP command name, its deadline, its position in
the queue and the time it was  submitted will be printed.  If the transaction is
currently being processed, the above information and the time it was started are
printed.  If the  transaction has finished, except  possibly for output, all the
above information, the time it finished, whether there were any errors, real and
cpu time used, page faults, and total real time, not including output processing
are printed.


Usage


     tp_get_xcn_status transaction_nums {-control_arg}


where:

1.     transaction_nums
            are the transaction numbers of the desired transactions.

2.     control_arg
            may be -brief or -bf  to cause only the  state of the transaction to
            be printed.

Name:  tp_list_pending_requests, tplpr

     The  tp_list_pending_requests command lists the  transactions that have not
been completed.


Usage


     tp_list_pending_requests {-control_args}


where control_args may be chosen from the following list:

     -total, -tt
          prints  only the  total number  of pending  transactions for  the TP
          subsystem and for each TP user.

     -long, -lg
          prints the transaction number, deadline, time submitted, position in
          the queue, and  the user's  TP_user_id of each  pending transaction.
          The default is to  print the transaction  number and the position in
          the queue.

     -before DT, -be DT
          prints  information  about  transactions  with  deadlines before the
          specified    time.    DT    is    a    string   acceptable    to   the
          convert_date_to_binary_  subroutine described in the MPM Subroutines.

     -after DT, -af DT
          prints  information  about  transactions  with  deadlines  after  the
          specified    time.    DT    is    a    string   acceptable    to   the
          convert_date_to_binary_  subroutine described in the MPM Subroutines.

     -user STR
          prints  information only  about  transactions for  the TP user whose
          TP_user_id is specified by STR.

Name:  tp_start


The tp_start command  starts transaction  processing by initializing the TP master table.  The process it is used in becomes the master process.


## Usage


    tp_start  path {-control_arg}


where:

1.  path
            is the pathname of the directory  that contains the control segments
            for the TP subsystem.

2.  control_arg
            may be the following:

    -new_master
            to make  the  current  process the  master  process  of a running TP
            subsystem.   This should be used  if the original  master process is
            destroyed while the TP subsystem is running.


## Notes


    The subsystem databases, transaction  control file and input queue must all
be  consistent, both  internally and with  respect  to each  other, before  this
command is used.


    This command does not  start up the other TP  processes.  This command must
be  issued before  other  processes   issue the  tp_worker_start or  tp_io_start
commands.

Name:  tp_stop


        The  tp_stop  command shuts  down  transaction  processing.  First  the  I/O
processes are told to stop accepting  input.   Then the worker processes are told
to log out either  when they have  finished their current  transactions, or when
they have run out of work.  Finally,  the I/O processes log out after all output
has been printed.


        When the I/O  processes stop  accepting input, a  message is printed on all
terminals that the TP subsystem is shutting down.  All further input is ignored.
The tp_stop command does not wait for the worker or I/O processes to finish, but
returns to command level.  After the  last I/O process has logged out, a message
is  printed on the  master  process's  terminal  indicating that  TP shutdown is
complete.


Usage


        tp_stop {-control_arg}


where control_arg may be -immediate or  -im to specify that the worker processes
should log out after  finishing their current  transactions.  The default is for
the worker processes to log out when the input queue is empty.

Name:  tp_who

The tp_who command prints the names of the current users of a TP subsystem.


Usage


tp_who {TP_user_ids} {-control_args}


where:

1.   TP_user_ids
            are the TP_user_ids of TP users.

2.   control_args
            may be chosen from the following list:

    -long, -lg
            prints the channel  name and information  about a user's terminal as
            well as the TP_user_id and I/O process name.  The default is just to
            print the TP_user_id and I/O process name.

    -io_process STR
            prints  information only about  users connected  to I/O process STR.
            STR is the I/O process name as used within the TP subsystem.

WORKER PROCESS COMMANDS AND SUBROUTINES


     This subsection contains  descriptions of commands and subroutines that may
be invoked in  a worker  process.  The  commands are used in  the worker process
absentee control  segment and the  subroutines may be used  by TPRs.  The worker
process commands and subroutines are presented in alphabetical order within this
subsection and include the following:


     tp_rollback_transaction_
     tp_verify_transaction_
     tp_worker_init_tcf
     tp_worker_start

Name:   tp_rollback_transaction_


    The  tp_rollback_transaction_  subroutine is  called by  a TPR to  abort  a
transaction and roll back all changes made during the transaction.  It should be
called if a  commitment would  fail.  This can  be determined  by calling either
transaction_call_$status    with    the    verify_refs    switch    on    or
tp_verify_transaction_.   The TPR is automatically  reinvoked if the TPR's retry
limit  has not  been  exceeded. This  subroutine returns  only if  there is  no
transaction in progress.  See the  appropriate TPR language section in Section 6
for  information about closing  files and  other tasks that  may have to be done
before this subroutine is called.


Usage


    declare tp_rollback_transaction_ entry ();

    call tp_rollback_transaction_;

Name:   tp_verify_transaction_


      The tp_verify_transaction_ subroutine may be called by a TPR to check if a
commitment would succeed.  A  commitment may  fail because of  an asynchronous
change made by another transaction.   If an asynchronous change is detected, the
TPR should call  tp_rollback_transaction_.  Calling this subroutine has the same
effect as calling  transaction_call_$status with the  verify_refs switch on.  It
is not  necessary for TPRs to  call this  subroutine. It may  be called between
phases of a TPR to see if further processing would be worthwhile.  A zero status
code from  tp_verify_transaction_  does not  guarantee that a  commitment  would
succeed.


Usage


      declare tp_verify_transaction_ entry (fixed bin(35));

      call tp_verify_transaction_ (code);


where:

1.    code                    (Output)
                  is a standard status code.  It may be:
                  error_table_$asynch_change
                        If an asynchronous change was detected

Name:   tp_worker_init_tcf

     The  tp_worker_init_tcf  command  attaches  and  opens the  TP  subsystem's
transaction  control file  (TCF) on the tcf_  I/O switch.   This command must be
invoked in a worker process before  tp_worker_start and before any databases are
opened  because  the TCF  I/O  switch,  tcf_ , must  be  specified in  the attach
description of any file that is to be protected by the commitment mechanism.


Usage


     tp_worker_init_tcf path


where path is the pathname of the  directory containing the control segments for
the TP subsystem.

Name:   tp_worker_start


     The  tp_worker_start  command turns a  process into a TP  worker process and
then  starts  processing  transactions.   This  command must  be  invoked  after
tp_worker_init_tcf.


Usage


     tp_worker_start worker_process_name path


where:

1.    worker_process_name
               is the name of the worker process as used within the TP subsystem.

2.    path
               is the pathname of the directory containing the control segments for
               the TP subsystem.

This subsection contains descriptions of commands that may be invoked in an I/O process. All of these commands except tp_io_start are TP user commands and must be entered in the command table as immediate commands. Links to them must be placed in the commands directory. They should be invoked with the tp_call_strings_af_ call convention. The names used by TP users may be chosen by the TP administrator.

The I/O process commands are presented in alphabetical order within this subsection and include the following:

    tp_io_cancel
    tp_io_enter_test_mode
    tp_io_exit_test_mode
    tp_io_get_xcn_status
    tp_io_list_pending_requests
    tp_io_signoff
    tp_io_start
    tp_io_who

Access to the TP Subsystem

This section describes how a TP user starts a transaction processing session. The user must first connect a terminal to the I/O process. This is done by using a terminal connected to a slave channel, or by using the dial access request. The signon TP access request is then used to start a transaction processing session.

Name: signon, s

The signon TP access request is used to gain access to the TP subsystem. It is a request to the I/O process to start the user identification and transaction processing procedures.

The signon request asks for a password from the user, and attempts to ensure that the password does not appear at all on the user's terminal or that it is thoroughly hidden in a string of cover-up characters. The password is a string of one to eight letters and/or digits associated with a TP_user_id.

After the user responds with a password, the I/O process looks up the TP_user_id and the password in the TP person-name table and verifies that the given password matches the password registered for the user.

If the TP user is permitted to sign on, a transaction processing session is started, and the user may enter transactions.

<u>Usage</u>

```
signon TP_user_id
```

where TP_user_id is the TP user's registered personal identifier.

Name:  tp_io_cancel


    The tp_io_cancel  command removes  one or more  transactions from the input
queue to  prevent  them from  being  processed.  The  transaction must not  have
started executing.  Only the user's own transactions may be canceled.


Usage


    tp_io_cancel transaction_nums

where  transaction_nums are  the transaction  numbers of the  transactions to be
canceled.

Name:  tp_io_enter_test_mode

     The  tp_io_enter_test_mode command  causes the TP  subsystem to execute and
then  automatically roll  back all  transactions  entered by  the user until the
tp_io_exit_test_mode   command is  given.   This   does not  affect  the  output
generated by the transaction.


Usage


     tp_io_enter_test_mode

Name: tp_io_exit_test_mode

     The tp_io_exit_test_mode command removes a user from test mode. All
subsequent transactions are processed normally (committed instead of being
rolled back).

Usage

     tp_io_exit_test_mode

Name:  tp_io_get_xcn_status


     The  tp_io_get_xcn_status command  returns  information  about one or  more
transactions  submitted  by the  user.   If a  transaction  has  not yet  been
processed, the TP command name, its deadline, its position in the queue, and the
time it was  submitted  will be  printed.  If a  transaction is  currently being
processed, the above information and  the time it was started are printed.  If a
transaction has finished, except possibly for output, all the above information,
the time it  finished, whether  there were any  errors, real  and cpu time used,
page faults, and total real time, not including output processing, are printed.


Usage


     tp_io_get_xcn_status transaction_nums {-control_arg}


where:

1.     transaction_nums
              are the transaction numbers of the desired transactions.

2.     control_arg
              may be -brief or -bf to print only the state of the transaction.

Name:  tp_io_list_pending_requests


     The tp_io_list_pending_requests command lists the transactions submitted by
the user that have not been completed.


Usage


     tp_io_list_pending_requests {-control_args}


where control_args may be chosen from the following list:

     -total, tt
          prints only the  total number  of pending  transactions for  the TP
          subsystem and for the user.

     -long, -lg
          prints the transaction number,  deadline, and time submitted and the
          position in the queue  of each pending  transaction.  The default is
          to print the transaction number and the position in the queue.

     -before DT, -be DT
          prints  information  about  transactions  with  deadlines before the
          specified time.   The DT  argument is a  string  acceptable  to  the
          convert_date_to_binary_ subroutine described in the MPM Subroutines.

     -after DT, -af DT
          prints  information  about  transactions  with  deadlines  after  the
          specified time.   The DT  argument is a  string  acceptable  to  the
          convert_date_to_binary_ subroutine described in the MPM Subroutines.

Name:  tp_io_signoff

     The  tp_io_signoff command  terminates a TP  user's  transaction processing
session.


Usage

     tp_io_signoff {-control_arg}


where  control_arg  may be  -hold or  -hd to  retain  communication  with the  TP
subsystem.  Another TP user may then immediately sign on.  The default is to end
communication with the TP subsystem.

Name:   tp_io_start


     The tp_io_start command turns a process into a TP I/O process.  It attaches
any specified channels and sets itself up as a dial server.


Usage


     tp_io_start io_process_name path {channels} {-control_args}


where:

1.   io_process_name
          is the name of the I/O process as used within the TP subsystem.

2.   path
          is the pathname of the directory containing the control segments for
          the TP subsystem.

3.   channels
          are the names of  slave channels.  The  channel names must be of the
          slave service type in the system channel definition table.

4.   control_args
          may be chosen from the following list:

     -dial dial_id
          establishes a  dial line for  dial_id.  This  allows terminals to be
          connected to the I/O process via the dial access request.

     -registered_dial dial_id
          similar to -dial but allows  users to omit specifying the User_id of
          the I/O process when invoking the dial access request.

     -switch_terminal_to_tp
          switches the handling of  terminal I/O from Multics command level to
          the TP subsystem.   This may be used for  debugging.  In an absentee
          process, the tp_input I/O switch  is opened for stream_input.  Lines
          read from this I/O switch are  interpreted by the TP subsystem.  The
          tp_input I/O switch must already  be attached.  This allows prepared
          scripts of a TP session to be run.


Notes


     The -dial and  -registered_dial  control arguments are  incompatible.  Only
one -dial or -registered_dial control argument may be given.


     If a slave channel or  -registered_dial is being used, the I/O process must
have      access      to      the      appropriate      ACS.      See      the
Multics Administrator's Manual -- System Administrator, Order No. AK50, for more
information about access control segments.

<u>Name</u>:   tp_io_who


     The   tp_io_who  I/O   process   command   has the   same   function  as  the  tp_who
master process command.  See the description of the tp_who command in the master
process command subsection.

COMMANDS USED OUTSIDE THE TP SUBSYSTEM


     This subsection contains descriptions of commands that are used outside the
TP subsystem or in the master process.  The TP administrator is the typical user
of the commands in this subsection.   The commands are presented in alphabetical
order within this subsection and include the following:


        tp_cvsct
        tp_display_command_table
        tp_display_input_queue
        tp_display_master_table
        tp_display_output_queue
        tp_meters
        tp_pre_create.ec
        tp_reset_xcn_num
        tp_shrink_q
        tp_user

Name:  tp_cvsct


     The tp_cvsct command is the TP  source command table compiler.  It converts
a source language command table to the binary form usable by the TP subsystem.


Usage


     tp_cvsct path


where:

1.   path
               is the  pathname of the  source  command table.   The source command
               table must  have the  tpsct suffix.   This suffix  is assumed if not
               given.


Notes


     The  binary  command  table is  created  in  the  working  directory.   Its
entryname is that  of the  source  segment with  the  tpbct suffix.   It can  be
installed as tp_command_table_  in the TP directory only when the TP subsystem is
not running.


Description of the Command Table Source Language


     The source language for the command table consists of a list of statements.
There may be a  global section  specifying  default values  that differ from the
system-provided defaults, and then a section for each command.

     The syntax of a statement is:

     <keyword>:   <parameter>;

Comments are started with "/*" and ended with "*/".


GLOBAL SECTION


     The global section  consists of statements whose  values are to be applied
to all commands as default values.  It   appears at the beginning of the command
table source segment and is terminated by the first name statement.


     The statements in the global section may be any of the statements described
in the  command  section with  the  exception of the  name and  entry_point_name
statements.   The  values specified in  the global section  may be overridden by
statements in the command section.

The global   section may specify a  default call  convention for only queued
commands or   immediate  commands.  If an   immediate statement  is present in the
global section,  this specifies to  which type of command  a global section call
convention statement  applies.  If an immediate  statement is not present in the
global  section, a global  section call  convention statement  applies to queued
commands.

## COMMAND SECTION

There   must be a  section  in the  source  command  table for  each command
available to the  TP users.  Each  command section begins  with a name statement
and must also contain an  entry_point_name  statement.  The other statements are
optional.

The statements are as follows:

name:   name1, name2, ... namen;
        The  parameter  namei is a  name by  which a TP  user can invoke the
        command.

entry_point_name:  STR;
        STR is the entry  point name of the command.   It may be of the form
        entryname or  entryname$entry_point_name.   All commands or links to
        them must be in a  directory named  "commands" in the TP subsystem's
        directory.

call_convention: STR;
        STR is the entry  point name of a subroutine  that is used to invoke
        the command.  It converts the input line into arguments for the TPR.
        It     may     be     of     the     form     reference_name     or
        reference_name$entry_point_name.   The search rules are used to find
        reference_name.  The default is  the tp_call_strings  subroutine for
        queued commands and the tp_call_strings_af  subroutine for immediate
        commands.  See the "Standard Call Conventions" section below.

cpu_time_limit:  N;
        The parameter is a decimal  integer specifying the maximum amount of
        cpu  time,  in  seconds,  that  the  command is  to  use on a single
        transaction.  If the time limit expires, the transaction is aborted.
        The default is zero (no time limit).

deadline_interval:  N;
        The parameter is a decimal  integer specifying the offset in seconds
        from the  time a  transaction  was queued  to its  deadline.  Of two
        transactions in the queue at the same time, the one with the earlier
        deadline  will be  started first.  No  attempt is  made to execute a
        transaction by the time  indicated as its deadline.  The deadline is
        only used to  order the  execution of  transactions.  The default is
        60.  It may be negative.

immediate:  STR;
        If STR is "yes", the command is executed in the I/O process.  If STR
        is "no", the  command  is queued as  a transaction.  This is the
        default.

real_time_limit:  N;
        The parameter is a decimal  integer specifying the maximum amount of

real time, in seconds, that the command is to use on a single
invocation. If the time limit expires, the transaction is aborted.
The default is zero (no time limit).

retry_limit:   N;
The parameter is a decimal integer specifying the maximum number of
times a transaction is to be re-executed if commitment fails. After
N+1 tries, the transaction is aborted. The default is 3.


## Standard Call Conventions


The following call convention subroutines are supplied with the TP
subsystem software:

tp_call_strings_          processes the input line and invokes the command like
                          the Multics command processor. Each argument, as
                          separated by white space, is passed as a character
                          string. No active function, quoting or iteration set
                          processing is done.

tp_call_string_af_        similar to tp_call_strings_ except the command is
                          called as an active function.


## Modifying the Source Command Table Compiler


This section describes the steps necessary to add new items to the command
table.

1.    Add the new items to tp_command_table.incl.pl1.

2.    Change the source command table compiler. See the description of
      reduction_compiler in Multics System Programming Tools, Order
      No. AZ03.

3.    Recompile all TP subsystem programs that use
      tp_command_table.incl.pl1, making appropriate changes to use the new
      items.

Name:   tp_display_command_table, tpdct


     The tp_display_command_table command  prints the internal representation of
a TP binary command table in put data  format.  The user should be familiar with
the internal representation of a TP binary command table.


Usage


     tp_display_command_table {path}

where path is  the  pathname of  a  binary  command  table.   The  default  is
tp_command_table_ in the working directory.

Name:   tp_display_input_queue, tpdiq


    The tp_display_input_queue command  prints the internal representation of a
TP input queue in put data format. The user should be familiar with the
internal structure of a TP input queue.


Usage


    tp_display_input_queue {path}

where path is the pathname of a TP input queue.  The tpinq suffix is assumed if
not given.  The default is tp.tpinq in the working directory.

Name:  tp_display_master_table, tpdmt


     The  tp_display_master_table  command prints  the  contents of a TP  master
table  in put  data  format.  The  user  should be  familiar  with the  internal
representation of a TP master table.


Usage


     tp_display_master_table {path}

where  path  is  the  pathname  of  a  TP  master  table.   The  default  is
tp_master_table_ in the working directory.

Name:  tp_display_output_queue, tpdoq


     The tp_display_output_queue command prints the internal representation of a
TP  output queue  in put  data format.   The  user should  be  familiar with the
internal structure of a TP output queue.


Usage


     tp_display_output_queue {path}

where path is the  pathname of a TP output queue.   The tpoutq suffix is assumed
if not given.  The default is tp.tpoutq in the working directory.

Name:  tp_meters


The tp_meters command displays metering information derived from a TP input queue.  This includes:

- the total cpu time and page faults for TPR execution

- the number of successful completions

- the number of errors

- the number of commitment failures

- the minimum, maximum, average and  standard deviation of the number of submissions  per  hour  both  total and  per  I/O  process  (hours  are measured from beginning of time period)

- the  minimum,  maximum,  average and  standard  deviation  of the time between submission and processing

- the minimum, maximum, average and standard deviation of the time spent in execution, both total and per worker


Usage


        tp_meters path {-control_args}


where:

1.   path
            is the pathname of a TP input queue to be metered.  The suffix tpinq
            is assumed if not given.

2.   control_args
            may be chosen from the following:

        -from DT, -fm DT
            specifies  the beginning  of the time  period.  DT  must be a string
            that  is   acceptable to   the   convert_date_to_binary_   subroutine
            described  in  the  MPM Subroutines.  The  default  is the  earliest
            submission  time of a transaction in the queue.

        -to DT
            specifies the end of  the time period.  DT  must be a string that is
            acceptable to  the  convert_date_to_binary_  subroutine described in
            the MPM Subroutines.  The default is the current time.

Name:  tp_pre_create.ec


     The tp_pre_create exec_com creates  an empty indexed multisegment file.  It
should be used before the first  reference to an indexed  file if the file does
not exist or has been  truncated.  In particular,  this should be used to create
the TCF, the TP input queue and the TP output queue.


Usage


     ec >unb>tp_pre_create path


where path is the pathname of the file to be created.

Name:  tp_reset_xcn_num


      The tp_reset_xcn_num command changes the TP subsystem's current transaction number to a specified value.  The next  transaction's transaction number will be the current transaction number plus  one.  The current transaction number can be decreased only  when the input queue  is completely empty.   See the tp_shrink_q command.  This command should not be used when the TP subsystem is running.


Usage


      tp_reset_xcn_num path {transaction_num}


where:

1.    path
            is the pathname of the directory containing the control segments for the TP subsystem.

2.    transaction_num
            is the new  current  transaction  number for the  TP subsystem.  The default is zero.

Name:  tp_shrink_q


     The tp_shrink_q  command removes  from a TP input queue   records concerning
transactions that  were processed  before a specified  time.   The records may be
copied before they are deleted.


Usage


     tp_shrink_q path {-control_args}


where:

1.   path
               is the  pathname of a TP  input queue  from which  records are to be
               removed.  The suffix tpinq is assumed if not given.

2.   control_args
               may be chosen from the following list:

     -before DT, -be DT
               removes  records of only  those  transactions that  completed before
               time DT.   DT  must  be  a   string  that  is  acceptable  to  the
               convert_date_to_binary_  subroutine described in the MPM Subroutines.
               The default is the current time.

     -delete, -dl
               deletes records without copying them.

     -output_description STR, -ods STR
               specifies that the records are to be copied before they are deleted.
               STR is the  attach description  of the file to  copy the records to.
               It must be enclosed in quotes.

     -all, -a
               removes all eligible records.   This is the default.

     -successful
               removes the records of transactions that completed successfully.

     -errors
               removes the   records  of  transactions  that   aborted.   These  are
               transactions that failed for any reason except exceeding their retry
               limit.

     -commitment_failure
               removes the   records of  transactions  that could   not be committed.
               These transactions exceeded their retry limits.

Notes

The -delete and -output_description control arguments are incompatible.

The -output_description control argument may be used to specify a file in the storage system or a tape by using appropriate attach descriptions.

This command can be executed while the TP subsystem is running. It can also be used in several processes simultaneously to do such things as put records of successful transactions on tape and records of commitment failure transactions into a file.

Name:  tp_user


The tp_user command edits the TP person-name table.  It registers TP users, deregisters TP users and changes their passwords.  This command should not be used while the TP subsystem is running.  A TP user should not be deregistered if he is signed on, has pending transactions in the input queue or has pending output in the output queue.


Usage


        tp_user {path} {-control_arg}


where:

1.    path
                is the pathname of the TP subsystem's person-name table.  The
                default is tp_person_name_table_ in the working directory.

2.    control_arg
                may be the following:

        -input_file path, -if path
                specifies that the input is to come from the segment whose pathname
                is path.  The segment contains exactly the same input as the user
                would type if the command were being used interactively.  If there
                are any errors, write requests will not be performed and the user
                will be questioned at the end.


        The tp_user command keeps prompting the user for requests until the user is finished.  Requests that change the information in the TP person-name table are called action requests.  The action requests are the following:

        add_user, au             register a new TP user
        delete_user dlu          deregister a TP user
        change_password, cpw     change a TP user's password
        verify_password, vpw     verify what a TP user's password is


        Typing a TP_user_id after an action request performs that action for a TP user.  As many TP_user_ids may be typed after an action request as desired.  The last action will be performed on succeeding TP_user_ids until another action request is typed.  The prompt indicates what the last action request was.  Each action request is explained in more detail below.


        TP_user_ids may contain uppercase characters, lowercase characters, numbers or underscores.  They must begin with an uppercase character.  TP_user_ids are from one to 32 characters long.  TP_user_ids are not related to Multics User_ids.


        Passwords may contain letters or digits.  They are from one to eight characters long.  If a password is not typed ahead, the user will be prompted for it.  The tp_user command attempts to ensure that the password does not appear at all on the user's terminal or that it is thoroughly hidden in a string of cover-up characters.

As much input may  be typed ahead on  a single line as  desired.  To type a
password ahead,  enclose it in braces,  e.g. {password}   If a password is typed
ahead, it will not be hidden.


The action requests do the following:

add_user, au
        registers a TP user.  The TP user's password is entered next.

delete_user, dlu
        deregisters a TP user.

change_password, cpw
        changes a TP  user's password.   The new password  of the TP user is
        entered next.

verify_password, vpw
        verifies a TP user's  password.  A password  is entered next and the
        user is  informed if the  typed  password is the  password of the TP
        user.

        In addition to the above action requests, the following requests are
        also available:

previous, p
        whenever  a  TP_user_id is  typed, it is  remembered.   The previous
        request uses the last TP_user_id with the current action request.

write, w
        writes all changes  into the TP person-name  table.  Action requests
        change a  copy of the TP  person-name  table.  Changes are not made
        permanent until a write request is done.

quit, q
        exits the  tp_user  command.  If the  TP  person-name table has been
        changed since the last write, the user is questioned.

help
        prints information about what may be typed next.

?
        prints a list of requests


Example


        The following example  registers WSmith, CKent,  and RDavis as TP users and
changes the password of MJones.   Everything following an  exclamation point on a
line is typed by the user.

                ! tp_user
                  Type "help" for instructions.

                  Request:  !  au WSmith CKent

                  Password for WSmith:
                  Password for WSmith again:

```
Password for CKent:
Password for CKent again:

Add user:  !  RDavis

Password for RDavis:
Password for RDavis again:

Add user:  !  cpw MJones

New password for MJones:
New password for MJones again:

Change password:  !  write quit
r 1402 0.600 2.2330 102
```

SECTION 4


vfile_ TRANSACTION INTERFACES



     This section contains descriptions  of the transaction_call command and the
transaction_call_  subroutine.  They  are  included in this ¯manual because their
interfaces ¯are s̄till  preliminary.  The  transaction_call_  subroutine is used by
the TP  monitor and  should not  generally  be used ¯by T̄PRs.   The commands and
subroutines  described in  this  section are  independent of  the  TP  subsystem
described in the rest of this manual.  The commands and subroutines described in
this section may be used without the TP subsystem.


     Note that "transaction" has a different meaning in this section than in the
rest of the manual.  A  vfile_ transaction is a unit  of processing that has the
appearance of taking place as¯an indivisible, atomic operation.  The transaction
numbers used by vfile_  bear no relation to the  transaction numbers seen by the
users of a TP subsystēm.


Transactions


     A transaction  is a unit of  processing  that has the  appearance of taking
place as an indivisible,  atomic operation.   Arbitrary procedures involving any
collection of  vfile_  indexed  files may  be invoked  as  transactions via  this
subroutine.


APPEARANCE


     An incomplete  transaction terminates either by  a successful commitment or
by a rollback.  That is to say, until a commitment is made, the database appears
unchanged, except  within the current  transaction.  Any  database modifications
that a transaction  makes appear  simultaneously outside  the transaction when a
commitment is made.


PURPOSE


     There are two major reasons for encapsulating a procedure as a transaction.
The first is to simplify the  programmer's task of handling inconsistencies that
can arise from  operations  that are  interrupted and not  resumed.  This may be
because of a system crash or an application program error.  Second, in the event
that a  database is  shared  among  several  processes, the  entire  burden  of
synchronizing file  access is  removed  from the programmer  and  automatically
managed by transaction_call_ .

TCF I/O SWITCH


        Each  process  that uses   transaction_call_  requires an  I/O switch  that
associates transactions with a particular database.  This I/O switch is attached
by the user  to a  permanent  transaction control  file (TCF).   This is used in
conjunction with the files that compose a single logical database.


TRANSACTION NUMBERS


        Each   transaction   associated  with a   TCF  has a   transaction  number.
Associated with each TCF I/O switch is  a current transaction number.  Initially
and after  a  commitment or  rollback, the  current  transaction  number is zero
indicating that no  current  transaction  is defined  for a TCF  I/O  switch.  A
transaction number is assigned  automatically when an indexed file attached with
the  -transaction option  of vfile_  is referenced  and the  current transaction
number is zero.


REFERENCE LISTS


        A per-process reference list is  automatically maintained with each TCF I/O
switch.  It is  implemented as an  indexed file without  records.  The reference
list  keeps  track  of  passive  references  made  during  the  course of  each
transaction so  asynchronous changes that might  invalidate a transaction can be
detected.  The  reference list also  identifies  all items  modified during the
transaction in order to  make the database  consistent at commitment or rollback
time.


Files


DATABASE


        Any  collection of  vfile_  indexed  files  may be a  database upon  which
transactions are applied.   All that is required is  that a common TCF always be
used in  conjunction with references  to any file in the  database, and that the
individual  database files be  attached with  the -transaction  option of vfile_
specifying a TCF I/O switch attached to the TCF of the database.


TRANSACTION CONTROL FILE


        The  TCF is a  permanent  indexed  file  containing  index  entries  but no
records.  The user  is responsible for  its creation, but  the TCF is implicitly
manipulated by vfile_ and transaction_call_, so that no explicit user operations
on the TCF are required.


TCF ENTRIES


        Keys are added to the  TCF when a transaction  number is assigned for a new
transaction.  Each key's descriptor is  a flag indicating the logical completion
state of a single  transaction.  Thus the atomicity  of a transaction is reduced
to changing the descriptor of its TCF entry.

OPENING CONSTRAINTS

In order to use transaction_call_ , the user must first attach and open the database's TCF. Then all database files to be referenced must be attached and opened before starting any transactions. None of these files should be closed within a transaction. If concurrent transactions are performed on a common database, the -share option of vfile_ must be given in the TCF attach description as well as in the attach descriptions of the shared database files.


ASYNCHRONOUS CHANGES

When a commitment is attempted or upon referencing a database item previously read in the same transaction, it is possible that an error resulting from an asynchronous change by another transaction may be detected. An asynchronous change occurs when the current transaction reads an item, and then another transaction makes a commitment which changes the item before the current transaction commits or rolls back. This situation makes it impossible to correctly complete the current transaction, and the transaction must be rolled back. To determine whether an unexpected error was caused by an asynchronous database change, use the transaction_call_$status entry with the verify_refs switch on.


See the description of the vfile_ I/O module in the MPM Subroutines. See the description of the transaction_call command for a description of the command level interfaces corresponding to the transaction_call_ entries.

Name: transaction_call, trc


    The transaction_call command performs, controls, or obtains status
information about atomic database operations.


Usage


    transaction_call opname switchname {args}


where:

1.   opname
            designates the operation to be performed.

2.   switchname
            is the name of the transaction control file (TCF) I/O switch.

3.   args
            are variable, depending on the particular operation to be performed.

     The opnames permitted, followed by their alternate forms, are:

            assign, a
            commit, c
            number, n
            rollback, r
            status, s
            transact, t


     Usage is explained below under a separate heading for each designated
operation.   The explanations are arranged alphabetically rather than
functionally.


Operation: assign, a

    transaction_call assign switchname


    This command reserves a unique transaction number for the current
transaction by creating a new entry in the TCF.


Operation: commit, c

    transaction_call commit switchname


    This command attempts to complete the current transaction.  If successful,
the current transaction number is reset to zero.

Operation:  number, n

    transaction_call number switchname

    This command prints the current transaction number.

Operation:  rollback, r

    transaction_call rollback switchname

    This  command  undoes  all  modifications  made on  behalf of  the  current transaction and resets the current transaction number to zero.

Operation:  status, s

    transaction_call status switchname {transaction_ no} {-control_args}

where:

1.   transaction_no
         is the number  of the  transaction whose status  is to be found.  If omitted or zero, the current transaction number is used.

2.   control_args
         may be chosen from the following:

   -brief, -bf
         suppresses the  counting and  printing of the  number of passive and nonpassive references made by the transaction.

   -verify, -vf
         specifies that all  passive references are  checked for asynchronous changes.

    This command prints the status of any transaction associated with a TCF.

Operation:  transact, t

        transaction_call transact switchname {-control_args} command_line


where:

1.    control_args
              may be chosen from the following:

       -retry N
              specifies the  maximum  number of  times  the  transaction  is to be
              retried if commitment fails.  The default is zero.

       -signal
              specifies  that if  commitment  fails and  the retry  count has been
              exceeded, the  transaction_failure condition is  signaled.  This  is
              the default.

       -no_signal
              specifies that  the  transaction_failure  condition should  not  be
              signaled if commitment fails and the retry count has been exceeded.

2.    command_line
              is a Multics command line that need not be enclosed in quotes.


        This command executes a given command line as a transaction.


        If  the  -signal  control  argument  is  specified, a  handler  for  the
program_interrupt  condition is  established.  This  handler  re-executes  the
command line.  The default action for the transaction_failure condition prints a
message and returns to command level.  The start command does not re-execute the
command line.   The  transaction is rolled  back before the  transaction_failure
condition is signaled.


Notes


        If no  transaction  number has  been obtained  via the  assign operation, a
transaction number  is  automatically  assigned upon  the first  reference  to a
database item  within a new  transaction.  The  TCF I/O switch  must be open for
update.


        See the description of  the transaction_call_  subroutine for more detailed
information.

Name:   transaction_call_


     The  transaction_call_  subroutine  manages the  transaction  mechanism  of
vfile_, including committing and rolling back.

---

Entry:   transaction_call_$assign


     This entry reserves and returns a unique transaction number for the current
transaction.  The  current  transaction number  must be zero,  i.e.,  undefined,
before this  entry is used.   The current  transaction number  is changed to the
transaction number  of the  transaction.  The TCF I/O  switch must be opened for
update so that a new entry can be created.


Usage


     declare transaction_call_$assign entry (ptr, fixed bin(35), fixed bin(35));

     call transaction_call_$assign (tcf_iocb_ptr, transaction_no, code);


where:

1.   tcf_iocb_ptr          (Input)
               is a pointer to the iocb for the TCF I/O switch.

2.   transaction_no        (Output)
               is the new transaction number.

3.   code                  (Output)
               is a standard status code.


Notes


     The user is not  required to assign  a transaction  number at all,  in which
case one is automatically assigned upon making the first reference to a database
item of the new transaction.

---

Entry:   transaction_call_$commit


     This  entry  attempts to  complete the  current  transaction on a  database
associated with a TCF I/O  switch.  The current  transaction number becomes zero
if the commitment is successful.

Usage

```
        declare transaction_call_$commit entry (ptr, fixed bin(35), fixed bin(35));

        call transaction_call_$commit (tcf_iocb_ptr, transaction_no, code);
```

where:

1.   tcf_iocb_ptr          (Input)
            is a pointer to the iocb for the TCF I/O switch.

2.   transaction_no        (Output)
            is the  transaction number of  the transaction  whose completion was
            attempted.

3.   code                  (Output)
            is a standard status code.  It may be:
            error_table_$asynch_change
                  if an asynchronous change was detected

_____

Entry:   transaction_call_$number

    This entry returns the current transaction number.

Usage

```
        declare transaction_call_$number entry (ptr, fixed binary (35), fixed
            binary (35));

        call transaction_call_$number (tcf_iocb_ptr, transaction_no, code);
```

where:

1.   tcf_iocb_ptr          (Input)
            is a pointer to the iocb for the TCF I/O switch.

2.   transaction_no        (Output)
            is the current transaction number.

3.   code                  (Output)
            is a standard status code.

_____

Entry:   transaction_call_$rollback

    This entry  undoes all  modifications that  have been  made by the  current
transaction in a database.

Usage

     declare    transaction_call_$rollback    entry    (ptr,    fixed    bin(35),    fixed
          bin(35));

     call transaction_call_$rollback (tcf_iocb_ptr, transaction_no, code);

where:

1.    tcf_iocb_ptr          (Input)
          is a pointer  to the iocb for the TCF I/O switch.

2.    transaction_no        (Output)
          is set to the transaction number of the aborted transaction.

3.    code                  (Output)
          is a standard status code.


Notes

     The  effect of  a rollback  is  logically  invisible  outside  the  current
transaction.  The    transaction  number for a    rolled-back  transaction is  not
reused.  After issuing a rollback, the current transaction number of the TCF I/O
switch becomes zero, i.e., undefined,  and the database is restored to the state
following the last commitment.

----

Entry:    transaction_call_$status


     This entry returns the status of any transaction associated with a TCF.  If
the verify_refs switch is  on, this entry checks  items in the reference list of
the transaction for asynchronous changes caused by another transaction.


Usage

     declare transaction_call_$status entry (ptr, fixed binary (35), bit (36)
          aligned, ptr, fixed binary, fixed binary (35));

     call transaction_call_$status (tcf_iocb_ptr, transaction_no, trc_flags,
          trc_status_ptr, transaction_status, code);

where:

1.    tcf_iocb_ptr          (Input)
          is a pointer to the iocb for the TCF I/O switch.

2.    transaction_no        (Input)
          is the  transaction  whose status is  desired.  Zero  indicates the
          current transaction.

3.    trc_flags            (Input)
          indicates what operations to perform.  See "Notes" below.

4.    trc_status_ptr        (Input)
          is a  pointer to  a  trc_status  structure  in which  information is
          returned.  If this pointer is null, the information contained in the
          structure is not returned.  See "Notes" below.

5.    transaction_status  (Output)
          is the  status of  the  transaction.   See  the  description  of
          transaction_status in the trc_status structure below.

6.    code                  (Output)
          is a standard status code.  It may be:
          error_table_$asynch_change
                  If an asynchronous change was detected

Notes

       If an  asynchronous change  is detected,  the reference  counts will not be
correct.

       The trc_flags argument is a bit string of the following structure:

       declare 1 trc_flags_s        aligned based (addr (trc_flags)),
              2 verify_refs         bit(1) unaligned,
              2 pad                 bit(35) unaligned;

where:

1.    verify_refs           (Input)
          indicates whether or not to check for asynchronous changes.
          "0"b don't check for asynchronous changes
          "1"b check for asynchronous changes

2.    pad                   (Input)
          is reserved for future use and must be zero.

          This structure is declared in transaction_call.incl.pl1.

       The trc_status_ptr pointer points to the following structure:

       declare 1 trc_status         aligned based (trc_status_ptr),
              2 version             fixed binary,
              2 transaction_no      fixed binary (35),
              2 transaction_status  fixed binary,
              2 passive_refs        fixed binary (34),
              2 non_passive_refs    fixed binary (34);

where:

1.    version               (Input)
          is the version of the trc_status structure.  It must be 1.

2.    transaction_no        (Output)
          is the transaction number of the  transaction the status information
          describes.

3.    status              (Output)
          is the status of the transaction.  It may be:
          trc_INCOMPLETE        (0) The transaction is in progress but has not
                                been committed or rolled back yet.
          trc_COMMITTED         (1) The transaction has been committed.
          trc_ROLLED_BACk       (2) The transaction has been rolled back.
          trc_UNDEFINED         (3) No TCF entry for this transaction exists.

4.    passive_refs        (Output)
          Is the  number  of  items  referenced  but  not  modified  in  the
          transaction so far.

5.    non_passive_refs    (Output)
          is the number of items modified in the transaction so far.

          The structure  and the named  constants are  declared in the include
          file transaction_call.incl.pl1.

___

Entry:   transaction_call_$transact


    This entry executes a command line  as an atomic transaction on a specified
database.  If the commitment is unsuccessful, the transaction is rolled back.


Usage


      declare transaction_call_$transact entry (ptr, char(*), fixed bin (35),
          fixed bin (35));

      call transaction_call_$transact (tcf_iocb_ptr, command_line,
          transaction_no, code);


where:

1.    tcf_iocb_ptr        (Input)
          is a pointer to the iocb for the TCF I/O switch.

2.    command_line        (Input)
          Is a  Multics  command  line   that is  to be  executed as a  single
          transaction.

3.    transaction_no      (Output)
          is the transaction number of  the transaction.

4.    code                (Output)
          is a standard status code.  It may be:
          error_table_$asynch_change
              if the transaction was rolled back

# SECTION 5

## ERROR HANDLING AND RECOVERY

### CRASH RECOVERY

A system crash usually does not result in the loss of data. However, processes are destroyed and so transactions are interrupted. When transaction processing starts again, new processes are created. All work associated with incomplete transactions is discarded. The aborted transactions are automatically rescheduled. When there is loss of data, most of the input queue should be intact on disk. Records modified after the time corresponding to a consistent database should be made consistent with the rest of the database. Those transactions should then be rerun.

No facilities are provided with this release for journalizing the input queue or for adjusting databases or queues in case of loss of data.

### TRANSACTION ERROR HANDLING

The TP subsystem recovers from unexpected errors by TPRs. When a TPR raises an unrecoverable condition, the worker process aborts the transaction and sends the user a message to that effect. A message with the actual cause of the error is written into the absentee output segment. Then the worker process rolls back any changes made and goes on to the next transaction. Recoverable conditions include command_error and endpage. A TP administrator can insert his own condition handler to detect what he considers to be recoverable conditions.

Sometimes a transaction cannot be committed because of asynchronous changes caused by another transaction. When this happens, the changes made by the transaction are rolled back and the transaction is retried. The maximum number of retry attempts for a particular TPR is specified in the command table.

SECTION 6

GUIDELINES FOR WRITING TPRS

## DESCRIPTION OF OPERATING ENVIRONMENT

All TPRs run in the worker process, except for immediate commands which run in the I/O process. Therefore, Multics access control is performed with respect to the worker or I/O process's access identifier, not the TP user who submitted the transaction. A TPR should not alter the environment by changing such things as the working directory or search rules. TPRs must be serially reusable. Therefore, a TPR must explicitly initialize certain types of data rather than use language initialization facilities. See the appropriate language section below. TPRs may call other programs; however in the worker process, the other programs should be explicitly initiated in the worker absentee control segment. All TPRs in the worker process are run within a run unit. There is not necessarily a new run unit for each transaction. TPRs may not start new run units because run units are not recursive. TPRs may be installed whenever the TP subsystem is not running.

## CALLING CONVENTIONS

TPRs are called using a call convention specified in the command table. A TP administrator may define TP subsystem specific call conventions. A call convention specifies the relation between the command line a TP user enters and how the arguments are passed to the TPR. The command table specifies an external procedure which implements the call convention.

## IMMEDIATE COMMANDS

Immediate commands are called as active functions. The active function return string is printed on the TP user's terminal. Immediate commands should not make database references or do any input/output with the terminal.

## INPUT/OUTPUT

Input/Output in the TP subsystem may be of various kinds:

● Terminal Input

● Terminal Output

● File Input/Output

● Peripheral Input/Output

Their relation to the TP subsystem is explained in the following paragraphs, as well as use of the Multics Relational Data Store (MRDS).


## Terminal Input

In this release, TPRs may not ask for input from the TP user who submitted the transaction. All input from the user must be specified on the command line. TPRs should not try to read from the user_input I/O switch.


## Terminal Output

In the worker process, the user_output I/O switch is attached to an I/O module which puts output into the output queue. Therefore, anything written on the user_output I/O switch is ultimately printed on the terminal of the TP user who submitted the transaction. A TPR may not send output to any other TP user. A TPR should not make any assumptions about the physical characteristics of a TP user's terminal. The terminal a TP user enters a transaction from may not be the same one that he is using when the output is printed.

The error_output I/O switch is attached to the I/O or worker process's absentee output segment. Therefore, error_output may be used to log errors or messages.


## File Input/Output

All I/O switches used by any TPR should be attached and opened via io_call in the worker process absentee control segment to reduce TPR execution overhead. Refer to the specific language section to determine when a file opened with language I/O facilities needs to be closed with language I/O facilities. In the Multics I/O system, there is a distinction between directly invoking the I/O system to open or attach an I/O switch and using language I/O facilities to open a file. Language I/O facilities will leave the I/O switch in the state in which they found it. See the MPM Reference Guide, "Programming Language Input/Output Facilities", for more information. Only vfile_ indexed files may be protected by the commitment mechanism.

Sometimes an asynchronous change causes a language I/O error. The TPR must determine if language I/O errors are the result of an asynchronous change or something fatal. If the language I/O error was caused by an asynchronous change, the TPR should call tp_rollback_transaction_. Each section on TPR languages explains how to handle this problem.


## Peripheral Input/Output

TPRs may request that files be printed with the dprint_ subroutine described in the MPM Subsystem Writer's Guide. Tapes may be used via the tape I/O modules described in the MPM Peripheral Input/Output manual. If a TPR attaches a tape, it should also detach it and establish a cleanup handler to detach the tape if the TPR is aborted.

Using MRDS

    All databases used by any TPR should be opened in  tp_init_database.ec to
reduce TPR execution overhead.  All files of a database that will be used should
also  be  readied in   tp_init_database.ec.   When a   TPR is   invoked, all  its
databases should be open and all files that will be used should be ready.  A TPR
may call  dsl_$get_dbi to obtain the  database index.   TPRs should not finish a
file or close any databases.  When a database is  created, the -control control
argument of create_mrds_db should be used to specify the TP subsystem's TCF.


TPR LANGUAGES


PL/I


    TPRs must  initialize all  static storage  each time they  are called.  The
stop statement should not be used.   Output written on the default sysprint file
is ultimately printed on the TP user's  terminal.  The default sysin file should
not be read  from.  The  columnposition,  linenumber and  pagenumber of external
files are undefined when a TPR is started.


    Before a TPR returns and before tp_rollback_transaction_ is called, the TPR
need only close  stream files and  record files for which  locate statements are
being used.  The  TPR should also free  any based or  controlled storage that is
allocated.  It is not  necessary to have a cleanup  handler that closes files or
frees based  and controlled  storage.  A  cleanup handler is  needed to detach a
tape only if language I/O facilities were not used to attach the tape.


    Only PL/I indexed  sequential data sets may be  protected by the commitment
mechanism.  An asynchronous change   may cause  the  transmit  condition to  be
raised.  The following on  unit should be included  in each TPR to roll back the
transaction if an asynchronous change occurs.  It should be established for each
file protected by the  commitment mechanism.  The  identifier "keyed_file" below
is the file value for which on unit is established.

```
on transmit (keyed_file)
     begin;
          declare code                    fixed binary (35);
          declare continue_to_signal_ entry (fixed binary (35));
          declare transaction_error  condition;
          declare error_table_$asynch_change
                                    fixed binary (35) external static;
          declare error_table_$asynch_deletion
                                    fixed binary (35) external static;
          declare error_table_$asynch_insertion
                                    fixed binary (35) external static;
          declare error_table_$record_busy
                                    fixed binary (35) external static;
          declare pl1_io_$error_code entry (file) returns (fixed binary (35));
          declare tp_rollback_transaction_
                                    entry ();
```

6-3                                    CC96

```
                code = pl1_io_$error_code (keyed_file);
                if code = error_table_$asynch_change
                        | code = error_table_$asynch_deletion
                        | code = error_table_$asynch_insertion
                        | code = error_table_$record_busy
                then call tp_rollback_transaction_;

                call continue_to_signal_ (code);
                if code ^= 0
                then signal transaction_error;
        end;
```


COBOL


       COBOL TPRs run in  implicitly started COBOL  run-units.  Therefore the STOP
or STOP RUN statements should not be used.  The EXIT PROGRAM statement should be
used to finish a TPR.  Input from the  command line can be passed as a character
string or a structure to  the COBOL TPR by a call  convention routine written in
PL/I.  See the Multics COBOL Users' Guide, Order No. AS43, for more information.
Since a  program may  be called  many  times within a  COBOL run-unit, all data
should be  explicitly  initialized  when the TPR  starts.  The  ACCEPT statement
should not be  used.  The DISPLAY  statement with the UPON  SYSOUT phrase may be
used to output  information to the TP  user who submitted  the transaction.  The
DISPLAY statement with the UPON  CONSOLE phrase may be used to put messages into
the worker process'  absentee output segment.  The  COBOL Message Control System
should not be used.


       To reduce overhead  associated with opening and  closing files, an external
switch should be  used to indicate if  files have been  opened.  The first thing
every COBOL TPR should do is call an initialization program to open files if the
external  switch  is off.   The  called  intialization  program  should open  all
external files that are used by any TPR.  The initialization program should then
turn the external switch on to  indicate that files have been opened.  It is not
necessary to close external files.


       Only COBOL files with relative  organization or indexed organization may be
protected by the commitment mechanism.  The FILE-CONTROL paragraph for each file
protected by the  commitment mechanism should  include the following FILE STATUS
clause:

       FILE STATUS IS status-key-name-1, status-key-name-2

The status keys are described as follows:

       WORKING-STORAGE SECTION.
       01 status-keys.
        02 status-key-name-1 PICTURE XX.
        02 status-key-name-2 PICTURE 9999.
        02 status-key-3 REDEFINES status-key-name-2.
         03 w PICTURE 9.
         03 x PICTURE 9.
         03 yz PICTURE 99.

The following USE procedure should be included in each TPR to roll back the transaction if an asynchronous change occurs. It should be specified for each file protected by the commitment mechanism. The word "file-name" below is the file for which the USE procedure is specified.

```
PROCEDURE DIVISION.
DECLARATIVES.
check-status SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON file-name.
check-for-asynch-change.
    IF (status-key-name-1 = "30") AND (w = 3 OR 4 OR 5 OR 6 OR 7)
        AND (x = 4 OR 6 OR 7 OR 8) AND (yz = 30)
        CALL "tp_rollback_transaction_".
    CALL "print_cobol_error_$switch" USING "error_output".
    CALL "signal_" USING "transaction_error".
    EXIT PROGRAM.
END DECLARATIVES.
```

## FORTRAN

FORTRAN TPRs should be written as FORTRAN subroutines. The stop statement should not be used. The return statement should be used to finish a TPR. Input from the command line can be passed as arguments to the FORTRAN TPR by a call convention routine written in PL/I. Alternatively, the PL/I call convention routine can put the TPR's input into a FORTRAN common block. See the Multics Fortran Users' Guide, Order No. CC70, for more information about how PL/I and FORTRAN programs communicate. TPRs should not use the read statement with the default unit numbers (5 or 41) or the input statement. To produce output that is ultimately displayed on the terminal of the TP user who submitted the transaction, use the write statement with the default unit numbers (6 or 42) or the print statement.

It is not necessary for FORTRAN TPRs to explicitly close files. However, a TPR should explicitly close a tape file when it is finished with it.

Only FORTRAN direct access files may be protected by the commitment mechanism. An asynchronous change may result in an error message in the absentee output segment. This message can be ignored. A FORTRAN TPR need not include error processing for asynchronous changes. If a FORTRAN TPR encounters an asynchronous change, the transaction will be rolled back and retried.

## COMMITMENT AND ROLLBACK FEATURES

The system provides a commitment facility which updates a file with all changes made by the process since the last commitment. Any changes made before the commitment are invisible to other users. At any time before the commitment, all changes made since the last commitment may be rolled back.

In the TP subsystem, a commitment is automatically attempted at the end of each transaction. It is recommended that TPRs not perform intermediate commitments since this may affect the TP recovery mechanism.

However, if a TPR takes a long time to execute, there is an increased probability that some of the records read would have since been changed by another transaction. If this is likely, the TPR should be written to include a call to transaction_call_$status with the verify_refs switch on or to tp_verify_transaction_. This will indicate whether a commitment would succeed if issued at that time. If it wouldn't, there is no reason to continue further. The TPR should call the tp_rollback_transaction_ subroutine which will roll back the current work and reinvoke the TPR.

worker_start_up.absin (cont)
  see worker process

CUT ALONG LINE

| TITLE | SERIES 60 (LEVEL 68) MULTICS TRANSACTION PROCESSING REFERENCE MANUAL |
|---|---|

ORDER NO. CC96-01

DATED JUNE 1979

**ERRORS IN PUBLICATION**

**SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION**

Your comments will be promptly investigated by appropriate technical personnel and action will be taken ☐
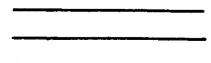as required. If you require a written reply, check here and furnish complete mailing address below.
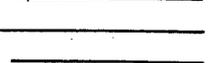
FROM: NAME _____  DATE _____

TITLE _____

COMPANY _____

ADDRESS _____

_____

PLEASE FOLD AND TAPE —
NOTE:  U. S. Postal Service will not deliver stapled forms
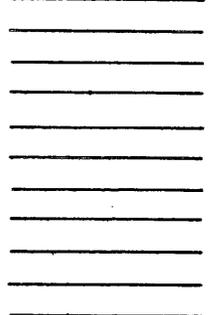
FIRST CLASS
PERMIT NO. 39531
WALTHAM, MA
02154

Business Reply Mail
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

**HONEYWELL INFORMATION SYSTEMS**
**200 SMITH STREET**
**WALTHAM, MA 02154**

**ATTENTION:  PUBLICATIONS, MS 486**

# Honeywell

CUT ALONG LINE

FOLD ALONG LINE

FOLD ALONG LINE

# Honeywell