FIND/FILE Activity Driver (ZTEST FFACT)

Overview

The ZTEST FFACT (aka ZTEST FFAC) will be used to generate FIND and FILE activity on z/TPF. The driver is a FIND/FILE continuous mode driver that will run until stopped by the user.

Each instance will be associated with a long running ECB which will repeatedly generate as many child ECBs of the driver as the user specifies. These child ECBs will operate on either the fixed file record type or pool type, for the specified number of records. The long running ECB will create an additional set of child ECBs once every 10 milliseconds.

The driver provides the user options to start and stop specific instances of the driver, and to display the status of each instance of the driver. There is minimal verification of the data that is read and written as the main purpose of this driver is to generate read/write activity.

Modes Supported By the Driver

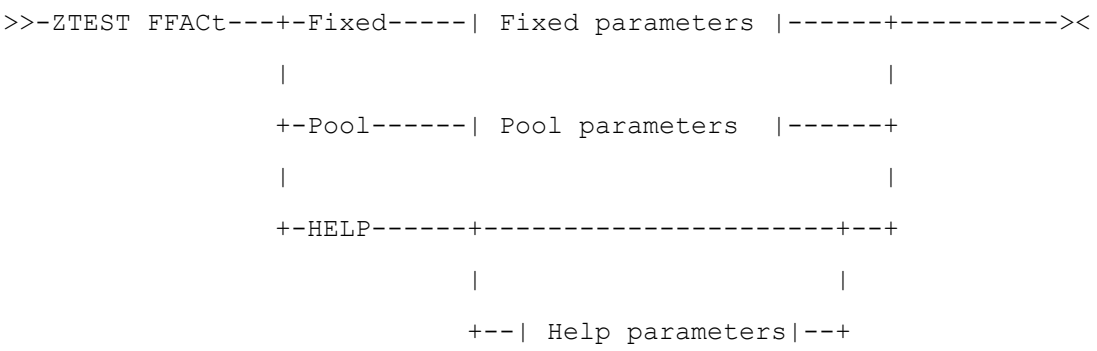
The driver will have 2 separate modes of operation:

(1) Fixed file records
(2) Short term pool records

The user is free to run multiple instances of the driver in either mode simultaneously so as to have a load of activity for both fixed and pool file records. The group numbers used for FIXED are separate from those used for POOL.

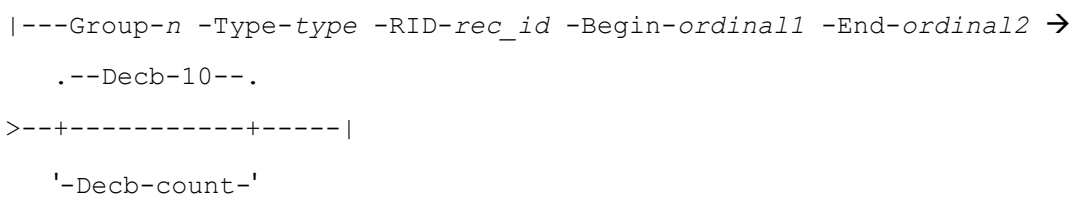
When using fixed file records, the existing contents of those records will be overwritten and replaced with validation data created by the driver. Care should be taken when selecting a record type to use.

Overall Syntax

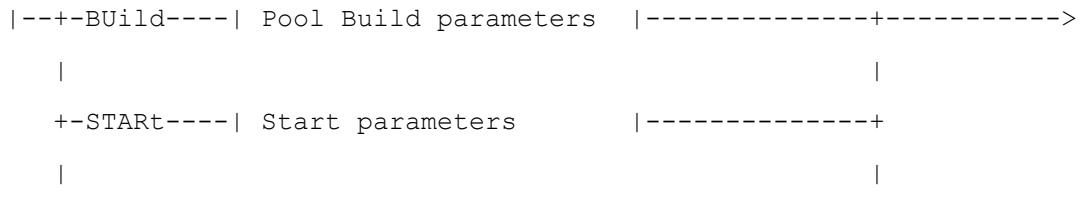


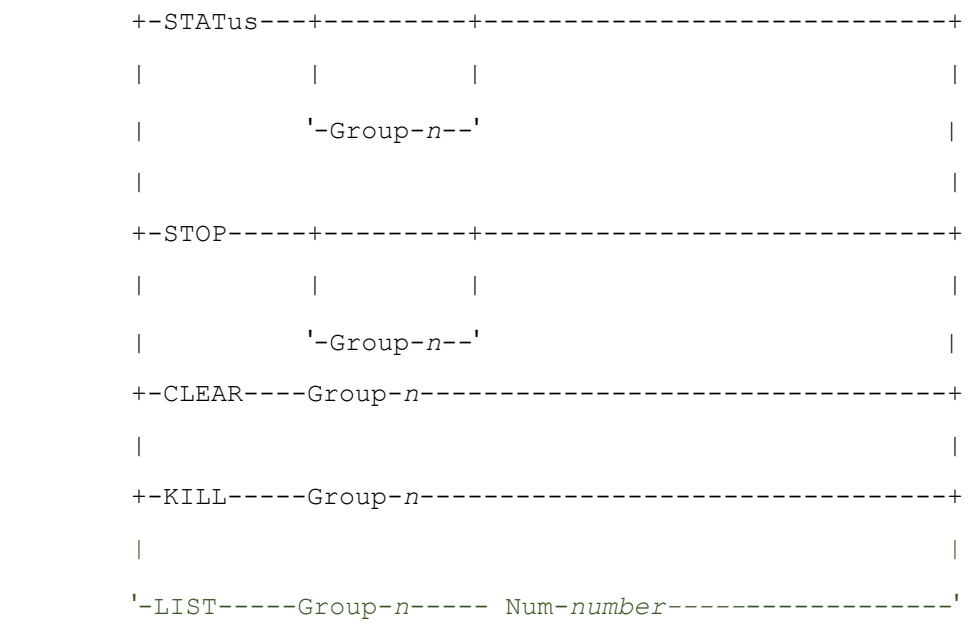
Fixed parameters

Fixed Build parameters

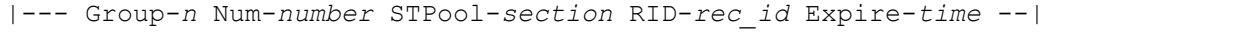


Pool parameters

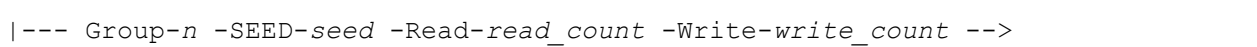


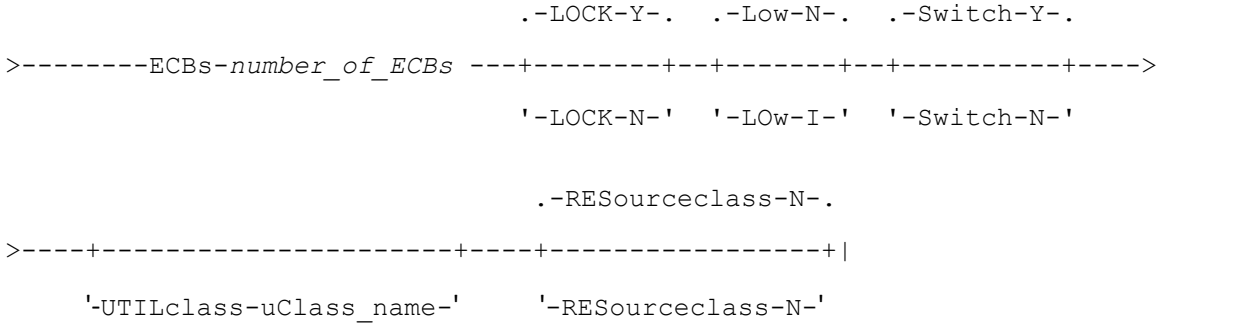


Pool Build parameters

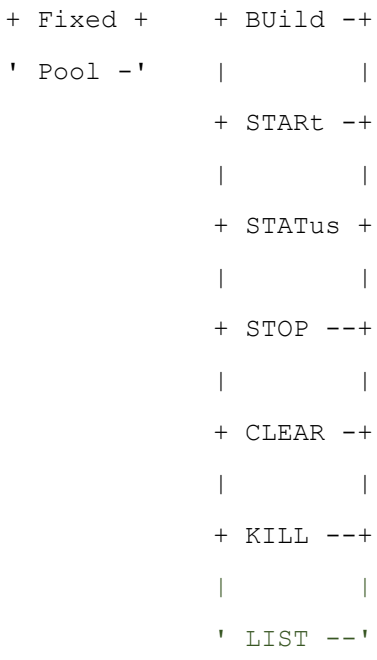


Start parameters





Help parameters



For a description of each of the fields, see the sections on **Fixed File Mode Syntax** and **Pool File Mode Syntax** that follow.

Fixed File Mode Syntax

All parameters are mandatory unless enclosed in [ ]:

(1)

ZTEST FFACT|FFAC Fixed BUild Group-*n* **Type-***type* **RID-***rec_id* **Begin-***ordinal1* **End-***ordinal2* **[Decb-***count***]**

Performs a build operation which will initialize a group=range of ordinals for a given record type so that continuous operations may later be performed on that group.

You will only need to run the BUILD command for one processor in the complex. When a BUILD command is run, besides initializing the records in the group it will store a file in the file system with the parameters that were used for the BUILD (record type, beginning and end ordinals, record type.)

Where:

Group-*n* assigns an integer value of *n* where 99999 >= *n* > 0 to the group being built. This value will be used for other commands operating on that group

There is a separate list of GROUPs maintained for each subsystem. So GROUP 1 build on BSS can have different attributes from GROUP 1 that was built on another subsystem; GROUP 12 may have only been built on BSS and not exist on any other subsystem and vice versa, etc.

Type-*type* specifies the record type that is to be built, where *type* is a string such as #WAARI, #HOTRE9C, #FFSREC1, etc. The # prefix is optional.
RID-*rec_id* specifies the record ID to be used to initialize the records on the build operation, where *rec_id* is specified as 4 hexadecimal digits (2 bytes of data).

Begin-*ordinal1* specifies the beginning ordinal for the record type whose records are to belong to that group. *ordinal1* is a 1- to 16-digit hexadecimal .

End-*ordinal2* specifies the ending ordinal for the record type whose records are to belong to that group. *Ordinal2* is is a 1- to 16-digit hexadecimal.

Decb-*count* specifies the number of DECBs to use when initializing the records. The default is 10. Additional DECBs will allow the build to complete faster but add additional overhead to the DASD subsystem in the process. *count* is a two digit decimal value between 1 and 99.

Notes:
Because ZTEST FFACT FIXED BUILD currently creates a file in the file system of the subsystem where the group is being built for the first time, the file system must be active for the subsystem on which a ZTEST FFACT FIXED BUILD command is entered. (That is, at least do ZPOOL 1052 UP if the subsystem is in 1052 state.) Once a fixed file group is built, the file system (and pools) do not have to be active to execute activity on that group. For example, you can build a group in NORM state, cycle your system down to 1052 state, and then execute ZTEST FFACT FIXED START to start FIND/FILE activity on that group.

If the system goes catastrophic in the middle of a ZTEST FFACT FIXED BUILD command, when you come back up you may want to look for /usr/QFNL* in the file system. If there are file(s) named /usr/QFNL_LOCK_GROUP* you want to delete them before attempting to build any fixed file groups again. This issue does not exist for ZTEST FFACT POOL BUILD which does not make any use of the file system.

(2)

ZTEST FFACT|FFAC Fixed STARt Group-*n* **SEED-***seed* **Read-***read_count* **Write-***write_count* **ECBs-***number_of_ECBs* **[Lock-Y|N][LOw-Y|I]**

[**SWitch-Y|I**] [**UTILclass**-*uClass_name*] [**RESourceclass-Y|N**]

Starts an instance of continuous activity which will operate on the identified group of fixed file records. There is no limit to the number of instances you can start to operate on a specific group.

Group-*n* identifies the number of the group that was previously built for which the operation will be performed, where *n* is the integer that was assigned to the group when it was built.

SEED-*seed* is the seed to be used to randomly access ordinals within the group, where *seed* is an integer >=0 <=999.

Read-*read_count* is the number of reads per iteration of continuous activity, where *read_count* is a decimal integer >=0 <=999.

Write-*write_count* is the number of writes per iteration of continuous activity, where *write_count* is a decimal integer >=0 <=999.

ECBs-*number_of_ECBs* is the number of ECBs that will be spawned to perform activity against the specified group for this instance of the continuous driver, where *number_of_ECBs* is a decimal integer >=1 <=999.

ECBs will be activated on via SWISC and will be load balanced across the active I-streams.

Lock-Y|N
Lock-Y is the default and means that prior to every Write, a find with hold will be done to lock the record before a file is done to update it.

If LOCK-Y is specified (or default taken), prior to every write, a find with hold (which is a read) must be done. Because of this, if LOCK-Y is in effect, the number of reads requested must be >= the number of writes requested, or the command will be rejected.

When LOCK-Y is in effect, for each write that is requested, a find with hold will be done for one of the reads. Any additional reads that are done will be done with a regular find (without hold).

Lock-N means skip locking. So any writes (file APIs) that are done will be done without ever having held the record.

Note:

1. If LOCK-N is specified, any reads done will translate to finds without hold. Any writes that are done will be done without having held the record.

2. Also, when LOCK-N is specified, there are no restrictions as to the number of reads you must request (e.g. You can do zero reads and only writes if you desire.)

Running instances with LOCK-N and LOCK-Y are mutually exclusive.

LOw-Y|I

LOw-Y sets the LOWPRIORITY attribute for the parent ECB

LOw-I sets the LOWPRIORITY attribute for the parent ECB and its child ECBs will INHERIT it _

SWitch-Y|I

SWitch-Y sets the SWITCHABLE attribute for the parent ECB

SWitch-I sets the SWITCHABLE attribute for the parent ECB and its child ECBs will INHERIT it

UTILclass-uClass_name causes this instance of the driver to use the specified utilization class

RESourceclass-Y|N

RESourceclass-Y this instance will use a LODIC resource class _

RESourceclass-N this instance will NOT use a LODIC resource class -- THIS IS THE DEFAULT

(3)

ZTEST FFACT|FFAC Fixed STATus [Group-*n*]

Use this command to display status of the fixed file continuous mode driver where

Group-*n* identifies the group for which status is to be displayed. The status will be a cumulative summary of status for all instances operating against that group, where *n* is the number used to identify the group when it was built. The status information will include:

- The record type of the group
- The beginning and ending ordinal number of the group
- The total number of ECBs currently performing continuous activity against the specified group
- The total number of read operations, including find with hold, executed for that group so far.
- The number of find with hold operations for that group so far, where a find with hold will be automatically done for each write. These are also included in the count of total reads

- The total number of writes for that group so far. The number of writes should equal the number of find with hold if LOCK-Y was specified.
- The total number of I/O errors that were encountered operating on this group. Each ECB which encounters an error will decrement the count of active ECBs and exit. When the count of active ECBs is zero, the status of activity for the group will be STOPPED.
- The total number of active ECBs currently active which are performing operations on this group.
- An indicator of the STATUS of activity for the group, which will be RUNNING if there are instance(s) currently started, or STOPPED if activity against that group has been stopped.

If you specify a group that was built on another processor or from a previous IPL, the information from that group will be read from the file system into memory and a detailed status will be displayed for that group on the current processor and subsystem where the command was entered.

If you omit GROUP from the ZTEST FFACT FIXED STAT command, it will display a list of groups, by number, which exist in memory for that subsystem. The status of any groups built on other processors in the complex or from previous IPL will not be included until they have been started again or their status displayed individually.

(4)

ZTEST FFACT|FFAC Fixed STOP [Group-*n*]

Use this command to stop all instances operating on a given group of fixed file records, where

Group-*n* identifies the group for which all activity is to be stopped, where *n* is the number that was associated with that group when it was built.

If you omit the GROUP on the STOP command, all fixed file groups in memory on the current processor will be stopped.

(5)

ZTEST FFACT|FFAC Fixed CLEAR Group-*n*

This will reset the counters on the processor that the command was issued on for a specific group, provided that there are no active ECBs associated with that group.

If a CLEAR is not done before starting additional instances for that group then the counters will be cumulative totals including activity from previous instances.

In order to reset the counters on all processors for a group a CLEAR command must be issued for each processor.
Where

Group-*n* identifies the group whose counts are to be cleared, where *n* is the number that was associated with that group when it was built.

(6)

ZTEST FFACT|FFAC Fixed KILL Group-*n*

Use this command to delete the file definitions for a group. In addition to removing the files for this group it will free the slot in the in-core table for that group.
You can only use this command if all activity for a given group of fixed file records has been stopped on that processor (either explicitly via the ZTEST FFACT FIXED STOP command or implicitly if all ECBs operating on that group have exited due to an error).

Any ECBs on other processors in the complex that are still operating on this group will continue to be able to until a KILL is entered for that group on that processor or the processor IPLs.

Where:

Group-*n* identifies the group which is to be "killed", where *n* is the number that was associated with that group when it was built.

Fixed File Mode Operation

The "parent ECB" (the one that services the START command) will loop and create the specified number of ECBs, passing it the start parameters that contain the number of reads/writes. For each child ECB it spawns, it will generate a random number (from the input seed from the start command), passing a different random number to each child ECB as an additional parameter.

Before spawning any ECBs for an instance, the driver ECB will:

- Read the first record in the group with record ID X'0000'.
    - If the record had not been initialized, the driver command will exit with an error

    - If the record is initialized, obtain the record ID and pass this as a parameter for all children ECBs to use this record ID when finding and filing updates to the record.
    - RCC will be zero in all cases so that RCC is not used for validation
    - NOTE: The BUILD command will also have stored the record ID in the named system heap to keep status of the group.
- Check if there is named system heap storage for that GROUP:
    - If yes, ECBs will be spawned against that GROUP
    - If no, check a file in the file system for that GROUP.
        - If found, allocate system heap for that GROUP and set up the parameters for the group that was probably built from another processor (or from a previous IPL), and then spawn the appropriate number of ECBs using the input parameters on the START command plus any information on that GROUP (record type and ordinal numbers to operate on).
        - If not found, issue an error message saying that a BUILD is required and exit.

After spawning the specified number of ECBs, the “parent ECB” will wait 10 milliseconds and then spawn another batch of ECBs

Each child ECB will:

- Check the named system heap for that GROUP to see the GROUP’s START/STOP indicator.
    - If set to STOP, the ECB will decrement the count of active ECBs for that GROUP and exit; otherwise it will continue
- Use the input random number as another seed to randomly access a different ordinal each time it does a read or write
- Perform the number of reads/writes requested and then exit.
    - As noted above, each if Lock-Y was specified, each write will be preceded by a find with hold so there must always be at least one read for every write;
    - Any additional reads will use find without hold;
    - For WRITEs, the driver will toggle back and forth between using the following APIs:
        - FILUC
        - FILNC / WAITC / UNFRC

- If any I/O error is encountered, counters for errors on read or write will be incremented, and the count of total ECBs for that group will be decremented and the child ECB will exit with an error message indicating:
    - The API that failed
    - file address that took the error
    - the SUD value
    - the expected record ID
- For each read, verify that the file address is stored in the record, as well as that there is a count field replicated throughout the record which steadily increments. Also include timestamps in the record and make sure they are all the same.

Fixed File Mode Examples

The following are examples of commands that you can enter for Fixed File Mode

```
ZTEST FFAC fixed Build GROUP-1 TYPE-#WAARI RID-C1C2 BEGIN-0 END-513
   >>>> Builds a group assigned number 1, of fixed records of type
#WAARI with starting ordinal 0, ending ordinal 1,299 (hex 513).  All
records in the group are assigned record ID hex C1C2.

ZTEST FFACT FIXED BUILD GROUP-2 TYPE-WAARI RID-C1C9 BEGIN-514 END-A27
   >>>> Builds a group assigned number 2, of fixed records of type
#WAARI (because record type WAARI does not exist), with starting
ordinal 1,300 (hex 514) and ending ordinal 2,599 (hex A27). All
records in the group are assigned record ID hex C1C9.

ZTEST FFACT FIXED BUILD G-3 TYPE-#HOTREC RID-C8D9 BEGIN-0 END-C8
   >>>> Builds a group assigned number 3, of fixed records of type
#HOTREC, with starting ordinal 0 and ending ordinal 200.  All records
in the group are assigned record ID hex C8D9.


ztest FFACT FIXED START G-1 SEED-33 R-3 W-2 ECBS-10
   >>>> Starts an instance of continuous activity against fixed record
group 1, spawning 10 ECBs, using SEED=33 to randomly access ordinals,
performing 3 reads for every 2 writes.  2 of these reads will be a
find with hold done before each write; the other read will be find
without hold.

Ztest FFACT FIXED START G-1 SEED-55 R-1 W-1 ECBS-5
   >>>> Starts another instance of continuous activity against fixed
record group 1, spawning 5 ECBs, using SEED-55 to randomly access
ordinals, performing 1 read for every 1 write.  Because the number of
reads = number of writes, all reads will be find with hold.

ZTEST FFACT FIXED START G-3 SEED-33 R-1 W-1 ECB-20
  >>>> Starts an instance of continuous activity against fixed record
group 3, spawning 20 ECBs, using SEED=33 to randomly access ordinals,
```

performing 1 read for every 1 writes. . Because the number of reads = number of writes, all reads will be find with hold.

ZTEST FFACT FIXED START G-3 SEED-33 R-1 W-0 ECB-15
>>>> Starts an instance of continuous activity against fixed record group 3, spawning 15 ECBs, using SEED=33 to randomly access ordinals, performing 1 read for no writes. Because there are no writes, all reads will be done as find without hold.

ZTEST FFACT FIXED STAT G-3
>>>> Displays the status of continuous activity against fixed record group 3

ZTEST FFACT FIXED STAT
>>>> Just displays a list of all groups for fixed that are in memory for the current processor. It is up to you to enter ZTEST FFACT FIXED STAT Group-n to see the details on that specific instance.

ZTEST FFACT FIXED STOP G-3
>>>> Stops all continuous activity on this processor for fixed record group 3.

ZTEST FFACT FIXED CLEAR G-3
>>>> Clears counters on this processor for fixed record group 3.
ZTEST FFACT FIXED START G-3 SEED-99 R-1 W-10 ECB-5 LOCK-N
>>>> Starts an instance of continuous activity against group 3, without locking, spawning 5 ECBs, using SEED=99 to randomly access ordinals, performing 1 read for every 10 writes. We allow more reads than writes because LOCK-N is specified. Also, because there is no locking, all reads will be find without hold.

ZTEST FFACT FIXED STAT G-1
>>>> Displays detailed status for Fixed group 1.

ZTEST FFACT FIXED STOP G-1
>>>> Stops all continuous activity on this processor for fixed record group 1.

ZTEST FFACT FIXED KILL G-1
>>>>> the driver “forgets” about the group and frees up all resource that was being used to keep track of it. NOTE: If other processors are in the complex they may continue to operate on that group until they are stopped. (It is suggested after stopping activity for the group on other processors, you kill activity on those processors for that group as well.)

Pool File Mode Syntax

All parameters are mandatory unless enclosed in [ ]:

(1)

ZTEST FFACT|FFAC Pool Build Group-*n* **Num-***number* **STPool-***section* **RID-***rec_id* **Expire-***time*

ZTEST FFACT POOL needs a build done on each processor in the L/C and repeated following any IPL if you intend to run continuous activity on that processor.

Get and initialize a group of pool records for the requested record ID, so that continuous operations may be performed on those pools. The pools that are gotten (by GETFC) will be stored in a table in named system heap. If not enough room exists in a given table for a GROUP that is being built, additional tables(s) will be allocated and chained to the first table for that GROUP.

The parameters on the build command are as follows:

Group-*n* assigns an integer value of *n* where 99999 >= *n* > 0 to the group being built. This value will be used for other commands operating on that group.

There will be a separate list of GROUPs maintained for each subsystem. So GROUP 1 build on BSS can have different attributes from GROUP 1 that was built on another subsystem; GROUP 12 may have only been built on BSS and not exist on any other subsystem and vice versa, etc.

STPool-*section* specifies the short-term pool section, where *section* is 4 characters (the 3-character section name plus a 1-character device type of A, B, C, or D). Valid pool section names are:

SST

Small, short-term.

LST

Large, short-term.

4ST

4 KB, short-term.

4S6

4 KB, short-term FARF6.

The GETFC calls to obtain pool records will use the record IDs defined for each short term section in tpfdf/rt/ub20.asm. They are coded in the header qfnl.h.

RID-*rec_id* specifies the record ID to be used to initialize the records on the build operation, where *rec_id* is specified as 4 hexadecimal digits (2 bytes of data).

The record ID (RID) is only used to initialize the data and does not have to correspond in the RIAT to a valid record ID used for the pool section specified by STPool-section.

Num-*number* specifies the number of pools for that record ID to be obtained and initialized as part of this group. *number* is a decimal integer > 0 but <= 999,999.

Expire-*time* specifies how often the driver should set a flag that says pools that have already been gotten should no longer be considered valid (i.e., it is "expired"). If a continuous driver ECB encounters a pool that is flagged as invalid, it will remove the file address for the pool from the table, and issue a GETFC to get a new pool file, replace address to be used in its place. *time* is a value in minutes >= 0 but <= 1440 (number of minutes in 24 hours). A value of zero means never flag the pools as invalid.

(2)

ZTEST FFACT|FFAC Pool STARt Group-*n* **SEED-***seed* **Read-***read_count* **Write-***write_count* **ECBs-***number_of_ECBs* **[Lock-Y|N]** **[LOw-Y|I]** [**SWitch-Y|I**] [**UTILclass**-*uClass_name*] [**RESourceclass-Y|N**]

Starts an instance of continuous activity which will operate on the identified group of pool records; where:

Note: There is no limit to the number of instances you can start to operate on a specific group.

Group-*n* identifies the number of the group that was previously built for which the operation will be performed, where *n* is the integer that was assigned to the group when it was built.

SEED-*seed* is the seed to be used to randomly access "ordinals" within the group, where *seed* is an integer >=0 <=999.

An "ordinal" for pools translates to an index in the table in the named system heap where the pools that were gotten for a given group have been stored.

Read-*read_count* is the number of reads per iteration of continuous activity, where *read_count* is a decimal integer >=0 <=999.

Write-*write_count* is the number of writes per iteration of continuous activity, where *write_count* is a decimal integer >=0 <=999.

Lock-Y|N
Lock-Y is the default and means that prior to every Write, a find with hold will be done to lock the record before a file is done to update it.

If LOCK-Y is specified (or default taken), prior to every write, a find with hold will be done. Because of this, if LOCK-Y is in effect, the number of reads requested must be >= the number of writes requested, or the command will be rejected.

When LOCK-Y is in effect, if the number of reads exceeds the number of writes, any additional reads that are done will be done with a regular find (without hold).

Lock-N means skip locking. Any writes (file APIs) that are done will be done without ever having held the record.

Notes:

1. *If LOCK-N is specified, any reads done will translate to finds without hold. Any writes that are done will be done without having held the record.*
2. *When LOCK-N is specified, there are no restrictions as to the number of reads you must request. You can do zero reads and only writes if you desire.*
3. *Running instances with LOCK-N and LOCK-Y are mutually exclusive.*

ECBs-*number_of_ECBs* is the number of ECBs that will be spawned to perform activity against the specified group for this instance of the continuous driver, where *number_of_ECBs* is a decimal integer >=0 <=999.

LOw-Y|I
LOw-Y sets the LOWPRIORITY attribute for the parent ECB

LOw-I sets the LOWPRIORITY attribute for the parent ECB and its child ECBs will INHERIT it _

SWitch-Y|I
SWitch-Y sets the SWITCHABLE attribute for the parent ECB
SWitch-I sets the SWITCHABLE attribute for the parent ECB and its child ECBs will INHERIT it

UTILclass-uClass_name causes this instance of the driver to use the specified utilization class

RESourceclass-Y|N
RESourceclass-Y this instance will use a LODIC resource class _

RESourceclass-N this instance will NOT use a LODIC resource class – This is the default.

(3)

ZTEST FFACT|FFAC Pool STATus [Group-*n*]

Use this command to display status of the pool file continuous mode driver where

Group-*n* identifies the group for which status is to be displayed. The status will be a cumulative summary of status for all instances operating against that group, where *n* is the number used to identify the pool group when it was built. If omitted, the command will display a list of groups, by number, which have been built for the current subsystem.

The status information will include:

- The record ID of the group
- The pool section type (e.g. SSTA, 4STB, 4S6A).
- The number of pool records for that record ID
- The total number of ECBs currently performing continuous activity against the specified group
- The total number of read operations excecuted for that group so far including find with hold operations that will be performed for each write.
- The number of find with hold operations for that group so far, where a find with hold will be automatically done for each write. These are also included in the count of total reads
- The total number of writes for that group so far.
- The total number of I/O errors that were encountered operating on this group. NOTE: Each ECB which encounters an error will decrement the count of active ECBs and exit.
- The total number of active ECBs currently active which are performing operations on this group.
- An indicator of the STATUS of activity for the group, which will be RUNNING if there are instance(s) currently started, or STOPPED if activity against that group has been stopped.
- The address of the system heap used to store the status information, which for POOL will include a list of the pool file addresses that were gotten for use by that processor.

(4)

ZTEST FFACT|FFAC Pool STOP [Group-*n*]

Use this command to stop all instances operating on a given group of pool file records, where:

Group-*n* identifies the group for which all activity is to be stopped, where *n* is the number that was associated with that group when it was built.If GROUP is omitted on the STOP command, all pool groups in memory on the current processor will be stopped.

(5)

ZTEST FFACT|FFAC Pool CLEAR Group-*n*

This will reset the counters on the processor that the command was issued on for a specific group provided that there are no active ECBs associated with that group.

If a CLEAR is not done before starting additional instances for that group then the counters will be cumulative totals including activity from previous instances.
Where

Group-*n* identifies the group whose counts are to be cleared, where *n* is the number that was associated with that group when it was built.
You can only use this command if activity for a given group of pool file records has been stopped (either explicitly via the ZTEST FFACT POOL STOP command or implicitly if all ECBs operating on that group have exited).

(6)

ZTEST FFACT|FFAC Pool KILL Group-*n*

Use this command to "kill" a group (telling the driver to "forget" about that group) by basically freeing all storage that was associated with that group.
NOTE: There is no RELFC done to release the pools as the driver will only obtain short term pools.

Group-*n* identifies the group which is to be "killed", where *n* is the number that was associated with that group when it was built.

You can only use this command if activity for a given group of pool file records has been stopped (either explicitly via the ZTEST FFACT POOL STOP command or implicitly if all ECBs operating on that group have exited).

(7)

ZTEST FFACT|FFAC Pool LIST Group-*n* **Num-***number*

Use this command to list the file addresses of the pool records currently in memory for the specified pool group.

Group-*n* identifies the group for which file addresses are to be listed is to be displayed.

Num-*number* identifies the number of file addresses you wish to have listed. To list all file addresses currently in memory for the specified group, you must ensure that *number* is the total number of records in the group. The number of records for a group can be displayed using the **ZTEST FFACT Pool STATus Group-***n* command.

Pool Mode Operation

The "parent ECB" (the one that services the START command) will loop and create the specified number of ECBs, passing it the start parameters that contain the number of reads/writes. For each child ECB it spawns, it will generate a random number (from the input seed from the start command), passing a different random number to each child ECB as an additional parameter.

Before spawning any ECBs for an instance, the driver ECB will:

- Read the first record in the group with record ID X'0000'.
  - If the record had not been initialized, the driver command will exit with an error
  - If the record is initialized, obtain the record ID and pass this as a parameter for all children ECBs to use this record ID when finding and filing updates to the record.
  - RCC will be zero in all cases so that RCC is not used for validation
  - NOTE: The BUILD command will also have stored the record ID in the named system heap to keep status of the group.
- Check if there is named system heap storage for that GROUP:
  - If yes, ECBs will be spawned against that GROUP
  - If no, check a file in the file system for that GROUP.
    - If found, allocate system heap for that GROUP and set up the parameters for the group that was probably built from another processor (or from a previous IPL), and then spawn the appropriate number of ECBs using the input parameters on the START command plus any information on that GROUP (record type and ordinal numbers to operate on).
    - If not found, issue an error message saying that a BUILD is required and exit.

After spawning the specified number of ECBs, the "parent ECB" will wait 10 milliseconds and then spawn another batch of ECBs

Each child ECB will:

- Check the named system heap for that GROUP to see the GROUP's START/STOP indicator.
    - If set to STOP, the ECB will decrement the count of active ECBs for that GROUP and exit; otherwise it will continue
- Use the input random number as another seed to randomly access a different ordinal each time it does a read or write
- Perform the number of reads/writes requested and then exit.
    - As noted above, each if Lock-Y was specified, each write will be preceded by a find with hold so there must always be at least one read for every write;
    - Any additional reads will use find without hold;
    - For WRITEs, the driver will toggle back and forth between using the following APIs:
        - FILUC
        - FILNC / WAITC / UNFRC

- If any I/O error is encountered, counters for errors on read or write will be incremented, and the count of total ECBs for that group will be decremented and the child ECB will exit with an error message indicating:
    - The API that failed
    - file address that took the error
    - the SUD value
    - the expected record ID
- For each read, verify that the file address is stored in the record, as well as that there is a count field replicated throughout the record which steadily increments. Also include timestamps in the record and make sure they are all the same.

Pool File Mode Highlights

- There is an EXPIRE time on the build command. On a time-initiated basis as specified by the EXPIRE time, the parent ECB will go through the table of pool file addresses for the group that was built and flag the item as having expired.

- All child ECBs that were spawned to perform continuous activity against that group will check the "expired" indicator before reading or writing to the pool file record to see if it has expired or not:

- If the pool file record is not expired, the ECB will issue a read or write to the pool file address depending on the Read and Write counts specified by the parent;

- If the pool file record is expired, the ECB will remove it from the table. It will then do a GETFC to get another pool file record for the same record ID, and store that address into the table. The Read and Write operations will then be performed on that new address.

  As only short term pools are used, there is no reason to RELFC the pools that expire as only short term pools will be used.

- When randomly accessing "ordinals" for pool file records within a given GROUP, the Pool File mode of the ZTEST FFACT driver will randomly access indexes within the table of pool file addresses that has been created for that GROUP.

Pool File Mode Examples

The following are examples of commands that you can enter for Pool File Mode:

```
Ztest FFACT pool Build Group=1 NUM-10000 STPOOL-4STA RID-D8F1 EXpire-5

    >>>> Builds a group of pool file records, assigning it
         Group number 1, with 10,000 records with record ID hex D8F1
         (character Q1).  Regardless of what attributes that record ID
         has in the RIAT, the pools for this group will be 4K Short Term
         pools from Device Type A.

         The records will expire every 5 minutes.

ztest FFACT pool Build Group=2 NUM-80000 STPOOL-SSTB RID-E5E5 Expire-10

    >>>> Builds a group of small, short term pool file records on device
         type B, assigning it Group number 2, with 80,000 records with
         record ID Hex E5E5 (character VV).  The records will expire
         every 10 minutes.


ztest FFACT pool start Gr-1 r-1 w-1 SEED-10 ECB-10 LOCK-Y

     >>>> Starts an instance of continuous activity against
          Pool Group number 1, spawning 10 ECBs which will
          use a SEED=10 to do continuous reads/writes of
          those records.  There will be one read for every
          write operation.  Because the number of reads =
          the number of writes, all reads will be find
          with hold.

ztest FFACT pool start Gr-1 r-4 w-1 SEED-11 ECB-20 LOCK-Y
```

>>>> Starts an instance of continuous activity against
Pool Group number 1, spawning 20 ECBs which will
use a SEED=10 to do continuous reads/writes of
those records. There will be 4 reads for every
write operation. Because the number of reads >
the number of writes, the reads that correspond to
the number of writes (1) will be find with hold; the
reads that exceed the number of writes (4 - 1 = 3)
will be find without hold.

ztest FFACT pool start Gr-2 r-100 w-1 SEED-99 ECB-15

>>>> Starts an instance of continuous activity against
Pool Group number 2, spawning 15 ECBs which will
use a SEED=99 to do continuous reads/writes of
those records. There will be 100 reads for every
write operation. One of these reads, which corresponds
to the number of writes, will use find with hold because the
default of LOCK-Y was taken.
The remaining 99 reads (100 - 1) will use find without hold.

ZTEST FFACT POOL STAT G-1
>>>> Displays the status of continuous activity against pool record group 1

ZTEST FFACT POOL STAT
>>>> Displays a list of all groups for pool records which the driver knows about. It is up to you to enter
ZTEST FFACT POOL STAT G-n
to see the details on that specific group.

ZTEST FFACT POOL STOP G-1
>>>> Stops all continuous activity on this processor for pool group 1.

ZTEST FFACT POOL CLEAR G-1
>>>> Now that the driver has been stopped for pool group 1, this command clears all counters for pool group 1.

ztest FFACT pool start Gr-1 r-10 w-1 SEED-22 ECB-5 LOCK-N
>>>> Starts a new instance of activity against pool group 1, spawning 5 ECBs which will use a SEED=22 to do continuous reads/writes of those records. There will be 10 reads per every 1 write. Because LOCK-N is specified, all reads will be find without hold.

ZTEST FFACT POOL STAT G-1
>>>> Displays the status of continuous activity against pool record group 1. Because a CLEAR had been done, the counts displayed will not include counts from any instance that had been running before the STOPO and CLEAR had been done.

ZTEST FFACT POOL STAT

>>>> Displays a quick summary status of all pool groups currently defined (in memory) for this processor, with the counts displayed in hex.

ZTEST FFACT POOL LIST G-1 NUM-500

>>>> Displays the file addresses of the first 500 records in Pool Group 1.

ZTEST FFACT POOL STOP G-1

>>>> Stops all continuous activity on this processor for pool group 1.

ZTEST FFACT POOL KILL G-1

>>>>> the driver "forgets" about the group and frees up all resource that was being used to keep track of it. NOTE: If other processors are in the complex they may continue to operate on that group. Because for ZTEST FFACT POOL BUILD each processor keeps its own storage to keep track of the pool group, the group will continue to exist on any other processors in the complex where it was built unless you stop the group ***and*** "kill" it there as well.