Using DataMove with InterSystems IRIS

Version 2023.1
2024-07-11

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using DataMove with InterSystems IRIS
InterSystems IRIS Data Platform Version 2023.1 2024-07-11

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

Using DataMove with InterSystems IRIS

This document describes how to use the DataMove process to move existing data associated with an InterSystems IRIS® data platform namespace to a new location.

Important: InterSystems highly recommends that you test DataMove with your specific namespaces and databases in a test environment before using it on a live system.

Note: Some single lines of code in this document wrap across lines, due to the page and font sizes.

1 Introduction to DataMove

The DataMove process allows you to move existing data associated with an InterSystems IRIS namespace to a different database, by:

- Creating new mappings for the namespace.
- Analyzing the mapping changes to calculate which globals and global subscripts need to be moved.
- Copying the data to the new database or databases.
- Activating the mapping changes.

For example, DataMove allows you to move a global, or portion of a global, from a namespace's default globals database to a different database. You can use the process to move a namespace's globals to a separate database from its routines, to split data across multiple databases, or otherwise move data to new locations based upon updated design decisions over the evolution of an application. DataMove allows you to move data and change mappings while actively using the same data in your applications.

For more information on mappings, see Add Global, Routine, and Package Mapping to a Namespace.

See Deprecated Calls and States below for APIs that have been deprecated and their replacements.

2 DataMove Restrictions

The DataMove process is subject to the following restrictions:

- DataMove is designed to move data between databases on the same instance. Extra operational care is required for mirrored systems and systems which have ECP clients attached.
- DataMove should not be used on applications that use extended global references. Data integrity cannot be ensured if the application mixes the use of mappings and extended references.
- Journaling must be turned on for the system.
- The freeze on journal error switch must be set.
- Data can only be moved from a local database to another local database, or from a mirrored database to a mirrored database.

- In a DataMove which contains mirrored databases, you can only run the DataMove on the Primary node.
- If a DataMove is running on the primary mirror, and the failover or DR node becomes the primary, the DataMove is stopped. If the node on which the DataMove was running becomes the primary again, the DataMove will resume where it left off.

For more information on extended global references, see Namespaces.

3 The DataMove Workflow

The DataMove workflow comprises the following phases:

1. Changes to the namespace mappings are saved in a temporary storage area.
2. A set of data moves is generated from the mappings.
3. The data moves are validated against the specified globals and databases, and sufficient free disk space in the destination databases is confirmed. If any issues are found, the user can correct them and resume the workflow.
4. When the DataMove starts, several background jobs copy the existing data to the new location. As the data is copied, other background jobs process the journal files and apply changes to the copied data.
5. When all of the data has been copied and journals applied, the new mappings are activated in the namespace.
6. After the namespace changes have been successfully activated, you can delete the old source data that has been copied to the new locations.

DataMove maintains a log file DataMoveName.log of all operations in the /mgr directory. There is also a record of all previous operations is in the DataMove.log file in the /mgr directory.

4 DataMove in a Mirror or ECP Environment

When using DataMove in an ECP or mirror environment, you must make sure that the new namespace mappings are updated on the ECP clients and mirrors. Note that if you are in a mirror environment and are going to move data into newly created mirrored databases, you must also create these mirrored databases on the backup and Async mirrors.

1. Create your DataMove and run it until it is ready to activate the new mappings.
2. If you have a failover mirror, you must demote the backup to a DR mirror before you activate.
3. If you have ECP clients, you must stop the application on the clients, or bring the ECP systems down before you activate.
4. If you are using Async mirrors, you must stop any applications accessing the data there, or any ECP clients connected directly to them before you activate.
5. At this point you can activate the new DataMove mappings.
6. Update the mappings on the Async mirrors to match the new DataMove mappings you just applied, including the former backup failover mirror.
7. Promote the former backup mirror from a DR Mirror back to a failover member.
8. Update any ECP clients with the new DataMove mappings you just applied.
9. Allow applications to restart on the Async mirrors and ECP clients.

5 The DataMove APIs

This section details the API calls required for each phase of the DataMove workflow.

Make note of the following guidelines:

- You must provide any needed scripting or user interface, according to your specific requirements.
- To make use of the needed macros, your code should include the %syDataMove file.
- Your code should check the status returned by each method before proceeding to the next API call.
- If any destination databases do not exist, you must create them prior to calling the DataMove APIs.
- The workflow must be executed in the %SYS namespace.

Important: InterSystems recommends that you have a current full backup of your system before running DataMove.

5.1 Create and Edit DataMove Namespace Mappings

5.1.1 DataMove.API.MapInitialize(Name as %String, Namespaces As %String) As %Status

Initializes the temporary storage area for a new set of mapping edits.

Argument:

- *Name* is the name of the DataMove.
- *Namespaces* is a comma-separated list of the namespaces on which you want to perform the DataMove.

This method must be called before any edits are made, and is only valid for the specified namespaces. You may move data in multiple namespaces as part of the same DataMove.

CAUTION: Calling this method deletes any existing edits in the temporary storage area, which is used for all mapping changes in all namespaces, including those made by the DataMove API and those made using the Global Mappings page of the Management Portal. (See Global Mappings.) To prevent concurrent mapping changes from causing failures, you must ensure that there are no mapping changes for any namespace in progress when you call MapInitialize(), and that no other mapping changes are initiated until the DataMove process is complete.

Example:

```
Set Namespace = "SALES"
Set Status = ##Class(DataMove.API).MapInitialize("DMName",Namespace)
If '$$$ISOK(Status) Write !,$SYSTEM.Status.GetErrorText(Status)
```

Example:

```
Set Namespaces = "SALES,PROSPECTS"
Set Status=##Class(DataMove.API).MapInitialize("DMName",Namespaces)
If $$$ISOK(Status) Write !,$SYSTEM.Status.GetErrorText(Status)
```

5.1.2 DataMove.API.MapGlobalsCreate(Name as %String, Namespace As %String, GblName As %String, ByRef Properties As %String) As %Status

Creates a new global mapping for this namespace in the temporary storage area. You can call this method one or more times, depending on the number of mappings you plan to include in this DataMove.

Arguments:

- *Name* is the name of the DataMove.
- *Namespace* is the namespace on which you want to perform the DataMove.
- *GblName* is the name of a global to be mapped to a specific database.
- *Properties* is an array of properties needed for this mapping, in particular, the name of the database used for this mapping.

Setting the *GblName* argument to "A" specifies that this mapping affects the entire global ^A. Setting this argument to "A(5):A(10)" specifies that this mapping affects the range of the global with subscripts ^A(5) up to, but not including, ^A(10).

Setting the *Properties* argument to an array `Properties` where `Properties("Database")` is set to "`USER2`" specifies that the global (or range of a global) is to be mapped to the database `USER2`.

The MapGlobalsModify() and API.MapGlobalsDelete() methods can be used to modify or delete mappings. See the Class Reference for more information.

Examples:

```
Set Properties("Database")="DSTDB"

;Move ^X(100) up to but not including ^X(200) to database DSTDB
Set Status = ##Class(DataMove.API).MapGlobalsCreate("DMName","SALES","X(100):(200)",.Properties)

;Move the entire ^INFO global to database DSTDB
Set Status = ##Class(DataMove.API).MapGlobalsCreate("DMName","SALES","INFO",.Properties)

;Move all Data from the first subscript in ^BILLING
;(including the node ^BILLING itself) up to but not including ^BILLING(100)
;to database DSTDB
Set Status = ##Class(DataMove.API).MapGlobalsCreate("DMName","SALES","BILLING(BEGIN):(100)",.Properties)

;Move the entire global ^PROSPECT
;from its currently mapped database to DSTDB
Set Status = ##Class(DataMove.API).MapGlobalsModify("DMName","SALES","PROSPECT",.Properties)

;Move the entire global ^ORDERS from its currently mapped database
;back to the default database for the namespace SALES
Set Status=##Class(DataMove.API).MapGlobalsDelete("DMName","SALES","ORDERS")
```

For more information on defining global ranges, see Global Mappings.

5.2 Generate the DataMove

5.2.1 DataMove.API.Generate(Name As %String, ByRef Properties As %String, ByRef Warnings As %String, ByRef Errors As %String) As Status

Creates a new DataMove based on map edits in the temporary storage area.

Arguments:

- *Name* is the name of the DataMove to be created.
- *Properties* is an array of optional properties to be used to create the DataMove.
- *Warnings* is an array returned with conflicts that do not prevent the DataMove from being performed.
- *Errors* is an array returned with conflicts that do prevent the DataMove from being performed.

`Properties` is an array passed in as the *Properties* argument:

- `Properties("MaxMBPerMin")` optionally specifies the maximum number of MB per minute the DataMove Operation is allowed to move to the destination database. Setting this to `0` allows the DataMove to run as fast as it can. If not passed, it uses the system default.
- `Properties("Description")` optionally provides a description of the DataMove to be performed.
- `Properties("Flags")` optionally provides any flags describing the DataMove operation, such as:
  - `$$$BitNoSrcJournal`, allow copying of non-journaled databases.
  - `$$$BitNoBatchMode`, run data copy without batch mode.
  - `$$$BitCheckActivate`, call the user-supplied routine **$$CheckActivate^ZDATAMOVE()** to check the application status before activating the mapping changes.
- `Properties("LogFile")` optionally specifies a log file name, if other than the default.

The `Warnings` array contains a list of mappings where the data being moved is also mapped from another namespace. This array is subscripted by the name of the global or the global range.

The `Errors` array contains a list of mappings where a conflict prevents the data from being moved. This array is subscripted by the name of the global or the global range.

Example:

```
Set Properties("Flags")= $$$BitCheckActivate
Set Status=##Class(DataMove.API).Generate("TEST",.Properties,.Warnings,.Errors)
```

For more information on database journaling, see Journaling.

For more information on running a process in batch mode, see Process Management.

5.3 Start the DataMove

5.3.1 DataMove.API.StartCopy(Name As %String) As %Status

Starts the DataMove copy, which handles the actual moving of data.

Argument:

- *Name* is the name of the DataMove.

This method starts the initial DataMove copy as a set of background jobs which copy the individual ranges from the source databases to the destination databases while at the same time journal transactions are applied to the destination databases for data which changes as it is being copied.

Before the copy actually starts, StartCopy() calls Validate() and ValidateSizes() to validate the DataMove. If the validation returns an error, the StartCopy() will return the validation error and will not start the DataMove. The user can then correct the error, and call StartCopy() again.

5.4 Monitor the DataMove

5.4.1 DataMove.API.GetProperties(Name As %String, ByRef Properties as %String) As %Status

Returns the current properties of the DataMove.

Arguments:

- *Name* is the name of the DataMove.
- *Properties* is an array of the DataMove properties as follows:

- `Properties("ExpandedState")` – Expanded external format of the state.
- `Properties("JRNMBToApply")` – Current number of MB of Journal file to process=0.
- `Properties("MaxMBPerMin")` – Maximum number of MB per minute the DataMove is allowed to move.
- `Properties("MBCopied")` – Amount of MB copied. At the end of a DataMove when all the data has been copied, and before Activate() is called, the amount of MB copied may not equal the amount of MBToCopy on a system where data in the source database is being updated while the DataMove is running.
- `Properties("MBToCopy")` – Approximate amount of MB to copy.
- `Properties("State")` – Current state of the move; see the table below.
- `Properties("StateExternal")` – External format of the State property.
- `Properties("Status")` – %Status value of any errors which occur.
- `Properties("Stop")` – Stop state of the DataMove:
  - `$$$DMStopNone` – Stop not signaled.
  - `$$$DMStopNormal` – Stop signaled when called by the Activate() method.
  - `$$$DMStopShutdown` – Stop signaled by normal system shutdown. DataMove will resume on system restart if the parameter to restart on system startup is set.
  - `$$$DMStopUser` – Stop signaled by user calling the method StopCopy(). Copy can be restarted by StartCopy().
  - `$$$DMStopForce` – Stop signaled by Force(). Copy can be restarted by StartCopy().
  - `$$$DMStopErrorRecoverable` – Stop signaled by error. Copy can be restarted by StartCopy() once the error is corrected.
  - `$$$DMStopErrorUnRecoverable` – Stop signaled by unrecoverable error. This may be due to several different reasons including journal errors. The only option here is to call Rollback() and then StartCopy() to restart the DataMove, or Delete() to delete the DataMove.

Example:

```
%SYS>Set x=##Class(DataMove.API).GetProperties("ABC1",.Properties)

%SYS>Zwrite Properties
Properties("JRNMBToApply")=0
Properties("MaxMBPerMin")=1200
Properties("MBCopied")=17030.67
Properties("MBToCopy")=20034.44
Properties("State")=6
Properties("StateExternal")="Copy"
Properties("Status")=1
Properties("Stop")=0
```

5.5 ^DATAMOVE Utility

The ^DATAMOVE utility displays the current state of all DataMoves defined on the system and allows users to set DataMove default parameters. Only one DataMove can be run at a time.

5.5.1 Setting DataMove Default Parameters

DataMove default system parameters allow the user to specify what happens when the system restarts, and can also limit the amount of data which can be moved per minute. By default, a running DataMove will copy data as fast as it can. However, this can affect other user processes on the system, as well as affect the latency of a mirrored system.

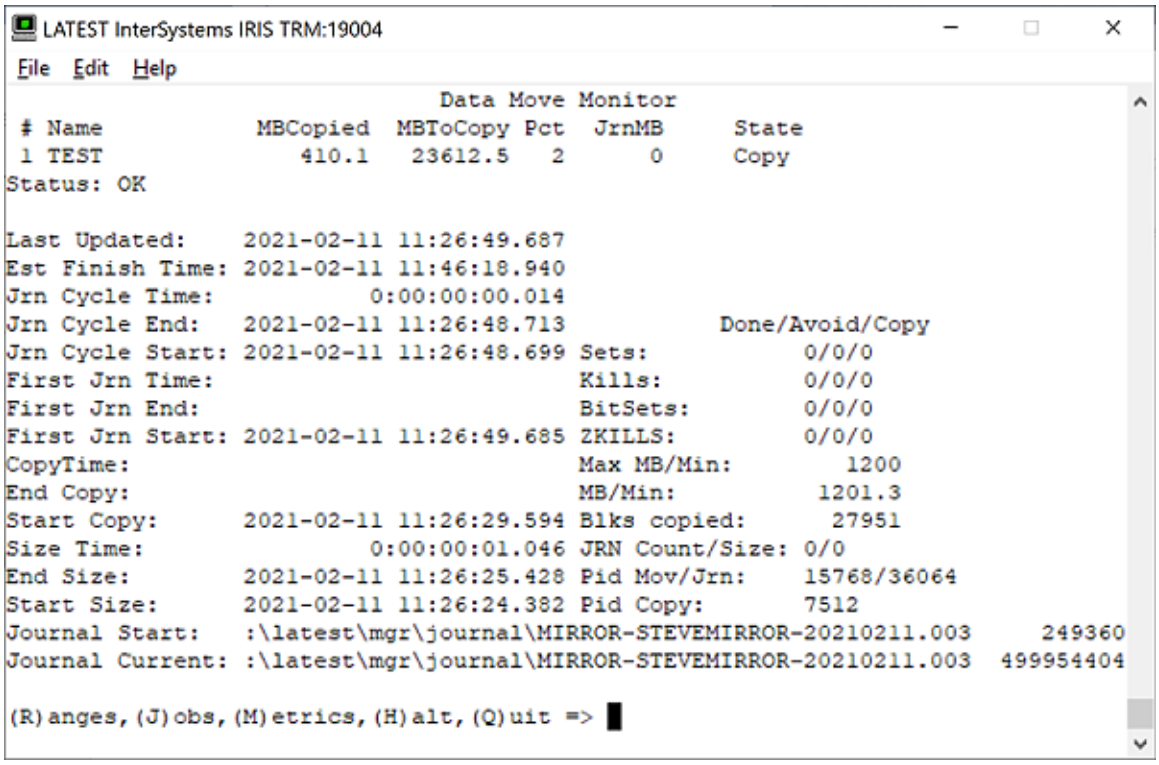
```
%SYS>d ^DATAMOVE
1) Monitor DataMove
2) Edit DataMove settings
3) Exit
Option? 2

Restart DataMove after normal system shutdown? Yes => Yes
Restart DataMove after system crash? Yes => Yes

Setting the maximum number of MB DataMove is allowed to copy per minute
to 0 means that DataMove will copy the data as fast as it can. Changing
this setting will apply the new value to future DataMoves, and optionally
any currently running DataMoves. You can also change this setting through
the 'Metrics' display screen for a running DataMove without changing the
system default.
Maximum number of MB to copy per minute? 0 => 1200
Do you want to apply this change to existing Data Moves? Yes => Yes
Confirm changes? Yes => Yes
DataMove settings updated

1) Monitor DataMove
2) Edit DataMove settings
3) Exit
```

5.5.2 Monitoring DataMove

Once you have called the Generate() method, you can use ^DATAMOVE to examine the state of the move, and which ranges are set to move.

```
%SYS>d ^DATAMOVE
1) Monitor DataMove
2) Edit DataMove settings
3) Exit
Option? 1
```

This is an example of what is displayed after Generate() and StartCopy() have been called. It shows the accumulated statistics for all the ranges in the DataMove.

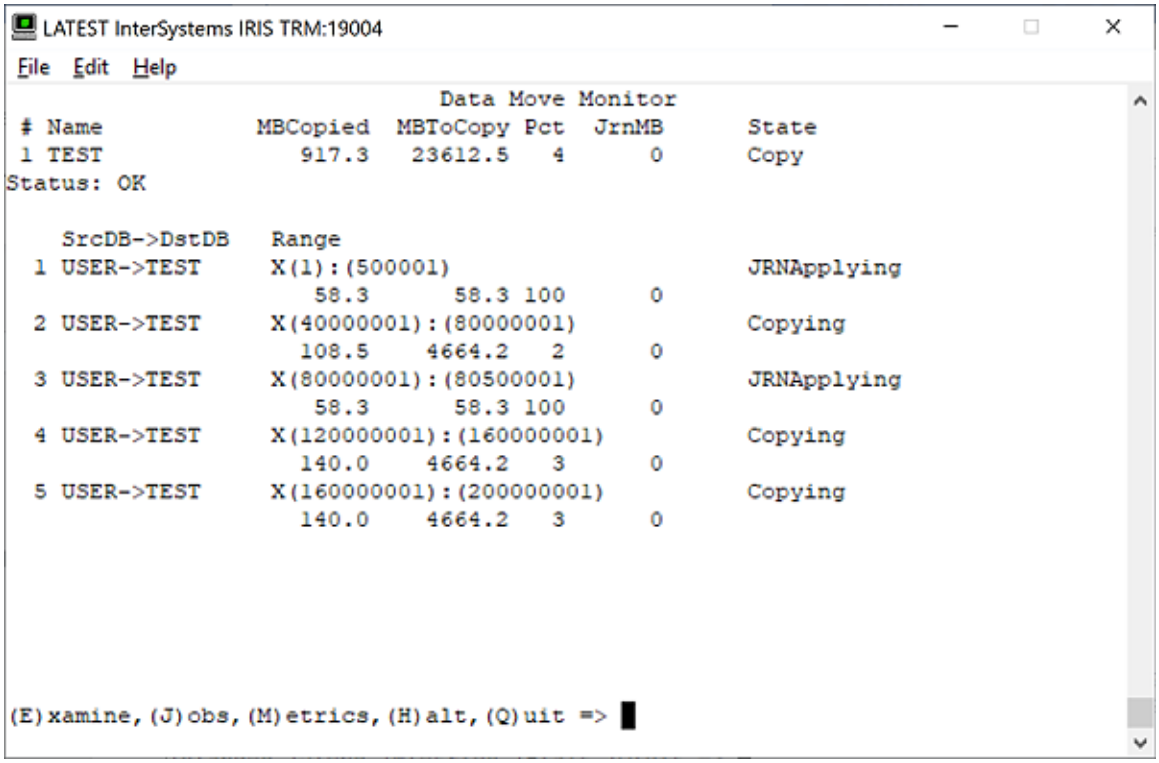


These fields show the following information:

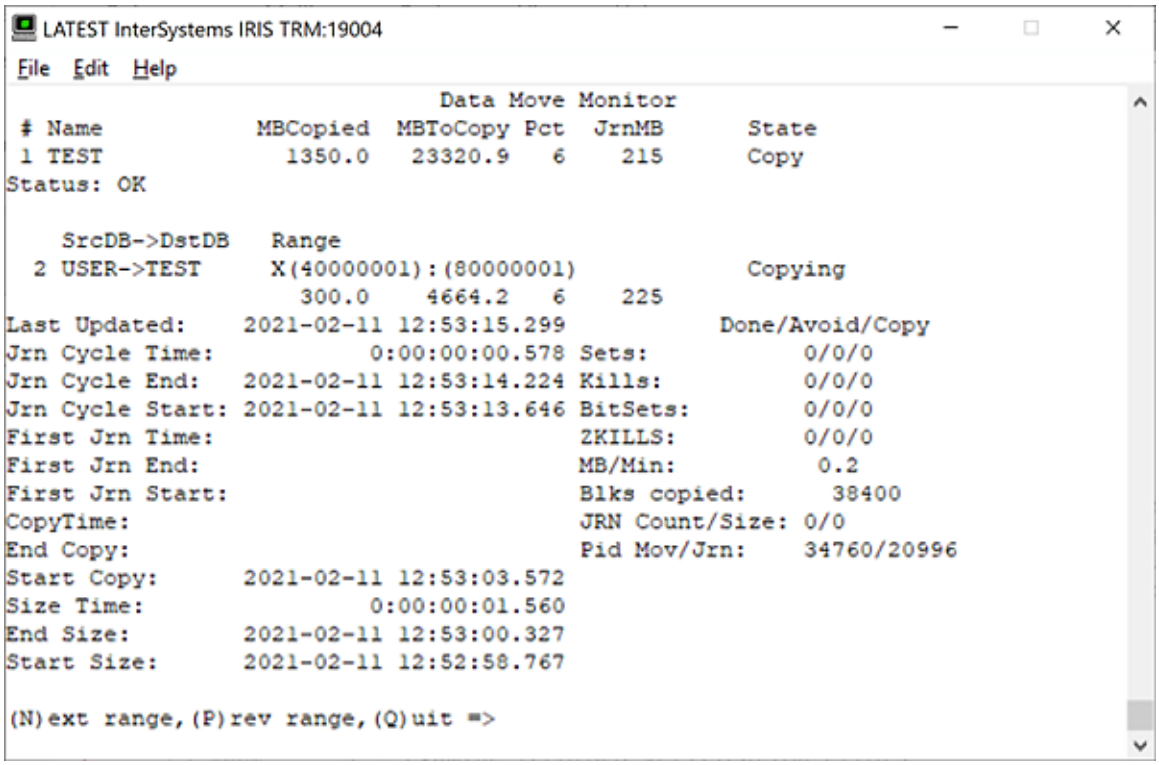
- `Jrn Cycle Time` – Amount of time it took for the journal scanner/apply to reach the end of the current journal file for the last time it reached the end of the journal.

- `First Jrn Time` – Amount of time it took for the journal scanner/apply to reach the end of the current journal file for the first time.
- `CopyTime` – Amount of time it took to copy the global.
- `Size Time`– Amount of time it took for the global size calculation.
- `Done` – Amount of each operation found in the journal file that it applied to the destination database.
- `Avoid` – Amount of each operation found in the journal file which could be skipped. Operations can be skipped if the Copy has not yet started, or the operation occurs in a piece of the global which has not yet been copied to the destination.
- `Copy` – Amount of each operation which were applied to the destination database while the Copy was running.
- `Journal Start` – Journal file and offset where the DataMove started.
- `Journal Current` – Current journal file and offset.
- `Max MB/Min` – Maximum MB per minute the DataMove is allowed to copy. This is set for the DataMove in one of three ways:
  - DataMove default from ^DATAMOVE.
  - The Metrics screen shown below.
  - The Modify() API.
- `MB/Min` – MB per minute the DataMove has copied over the lifetime of the DataMove.
- `Blks Copied` – Actual number of database blocks copied and the % which were prefetched
- `JRN Count/Size` – Number of transactions and total size for this DataMove which were found in the current journal file and the previous journal file before the StartCopy() method was called.
- `Pid Move/Jrn` – Process ID of the two processes handling the copy and dejournal.
- `Pid Copy` – Process ID of the Master DataMove process.

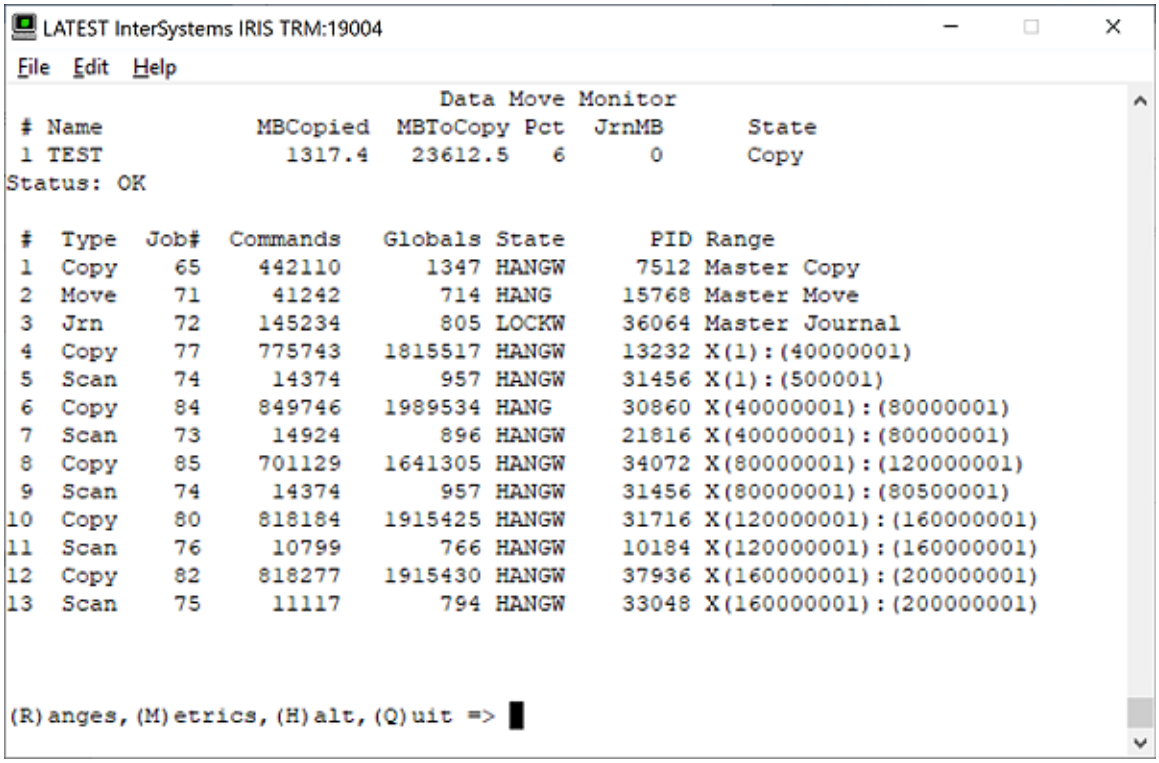
When you select the range option, you can see that there are 5 ranges of globals which are being moved.



You can examine the detail of each of the ranges in the Range Detail screen. These are described above, but apply only to the selected range.

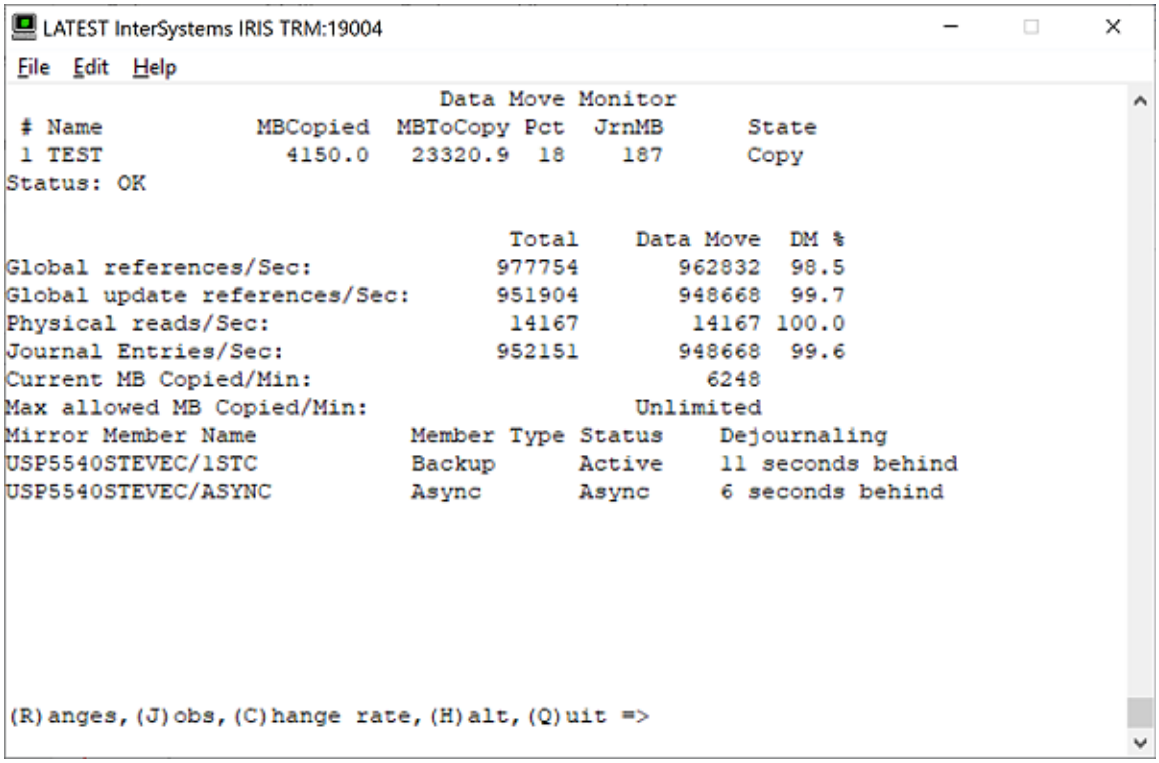


The Jobs display show which jobs are operating on which pieces of data.



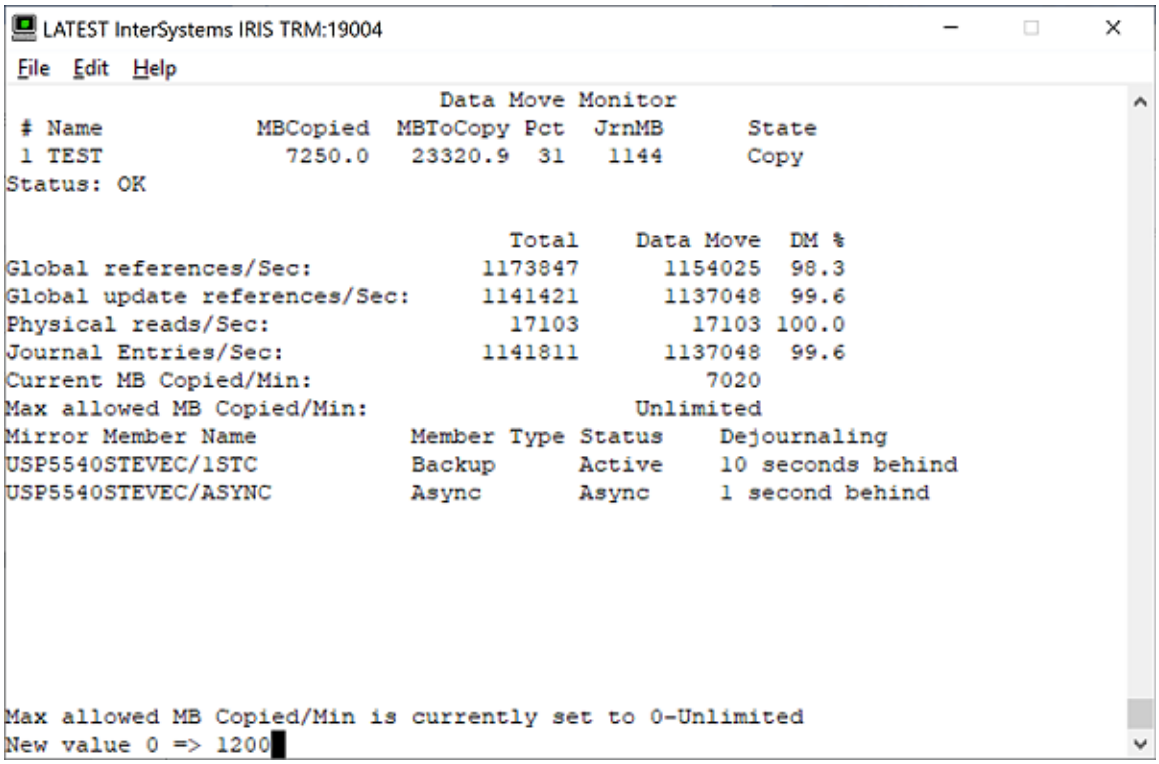
Here you can see that job 84 is currently copying the data in Range X(40000000), job 73 is scanning the journal for the same range and applying transactions it finds in this range. The "Master Move" job 71 controls all processes which are still copying data. The "Master Journal" job 72 controls all the processes which are scanning or applying journals. And the "Master Copy" job accumulates statistics and determines when the Activate() method can be called. The job number, Commands, Globals, State, and PID here correspond to the same information found in JOBEXAM or the Management Portal process display.

The Metrics display shows the activity on the system, and how much of it is related to the running DataMove.

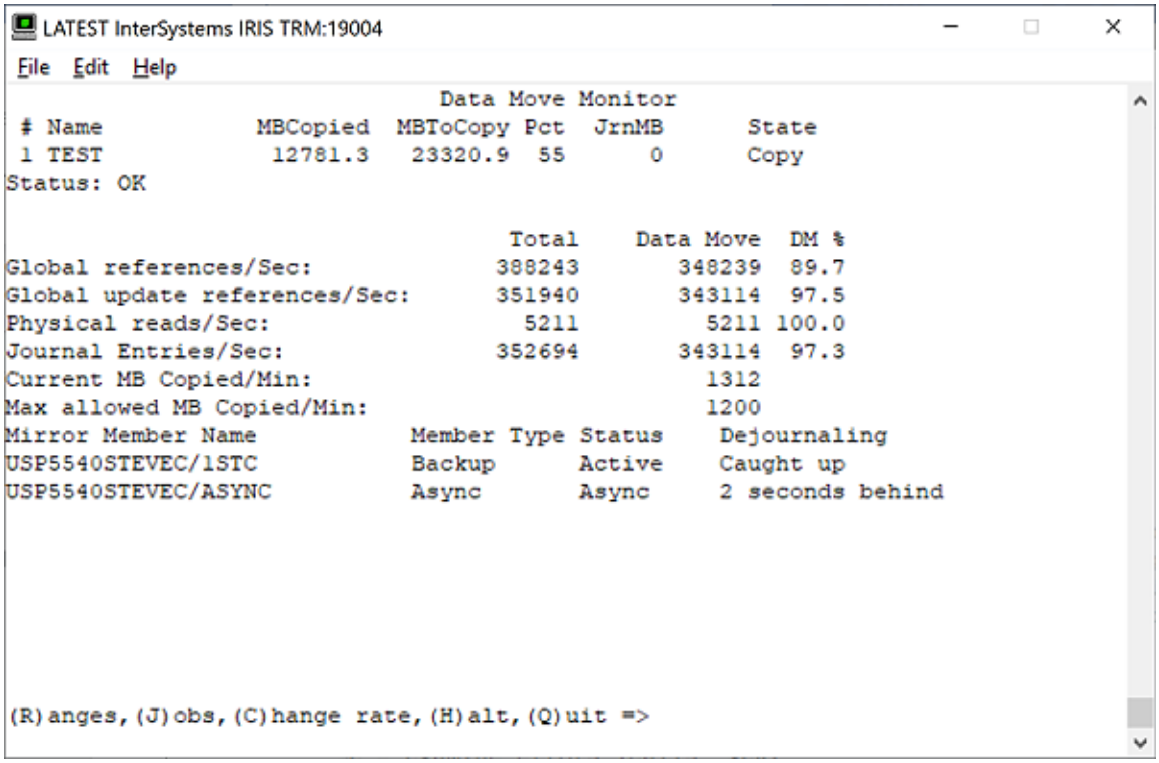


In this example we can see the DataMove is running with the Max allowed MB Copied/Min set to 0 (Unlimited). We also see that it is taking up the majority of global references and journal entries, as well as causing the mirror latency to increase.

We can restrict the DataMove to a slower rate as follows:



And now we see the new rate and much lower metrics.



5.6 Activate Mapping Changes and Finish the DataMove

5.6.1 DataMove.API.Activate(Name As %String, Display as %Boolean, Timeout As %Integer = 120) As %Status

Finishes the DataMove and activates the namespace mapping changes.

Arguments:

- *Name* is the name of the DataMove.
- *Display* should be set to 1 if you want to see progress messages.
- *Timeout* is number of seconds to wait for the DataMove to finish running and apply journals operation before proceeding.

This method stops the DataMove background jobs, finishes processing any journal files, writes the mapping changes to the CPF, and activates the mapping changes. It momentarily sets switch 10 to prevent other processes from interfering with the execution.

If `$$$BitCheckActivate` is set, the method will call a user-supplied routine **$$CheckActivate^ZDATAMOVE()**, if it exists, to execute before continuing. If **$$CheckActivate^ZDATAMOVE()** does not return `1`, the method will quit without activating the new mappings, leaving the DataMove running.

Note: The user-supplied routine is called immediately before switch 10 is set to activate the new mappings. This routine can do anything the user wants before activating the new mappings.

For more information on switch 10, see Using Switches.

Note: Activate() checks the State property of the DataMove to make sure that the initial copy is complete and the journal apply is caught up before proceeding.

5.7 Delete Source Globals and Finish DataMove

After the copy has completed and the updated namespaces activated, you should verify that that your application data has been successfully copied and the mappings activated. Once you have done this, you can proceed with deleting the source globals which have been moved, and finish the DataMove.

5.7.1 DataMove.API.DeleteSourceGlobals(Name As %String) As %Status

Deletes the globals from the source directory that have been copied in the DataMove.

Argument:

- *Name* is the name of the DataMove.

This method deletes all globals in the source database that have been copied to the destination database. Several processes may be created to delete the source globals.

5.7.2 DataMove.API.Finish(Name As %String) As %Status

Completes the DataMove process.

Argument:

- *Name* is the name of the DataMove.

This method writes a success or failure message to the log file, closes the log file, and sets the State property of the DataMove to `$$$DMStateDone`. It also copies the log file into the file DataMove.log, which is a record of all the DataMoves performed on the system.

5.7.3 DataMove.API.Delete(Name As %String) As %Status

Cleans up the DataMove process.

Argument:

- *Name* is the name of the DataMove.

This method deletes the DataMove and cleans up any temporary storage.

If the DataMove operation you want to delete has been started and then stopped, you should first call the Rollback() method to rollback any of the data which had been moved.

5.8 Other API Calls

5.8.1 DataMove.API.Validate(Name As %String) As %Status

Validates the DataMove.

Argument:

- *Name* is the name of the DataMove.

Validating the DataMove involves looking at all of the specified mappings, checking the source and destination databases, and making sure the destination globals do not already exist. Any errors are reported in the status.

Validate() is called as part of StartCopy(). You can use this method to validate the DataMove before you actually start the copy with StartCopy()

5.8.2 DataMove.API.Modify(Name as %String, byref Properties as %String) as %Status

Modifies an existing or running DataMove.

Arguments:

- *Name* is the name of the DataMove.
- *Properties* is an array of properties to modify, which may include:
  - `Properties("MaxMBPerMin")` - Maximum number of MB per minute the DataMove operation is allowed to move to the destination database. Setting this to `0` allows the DataMove to run as fast as it can. This can be called on a running DataMove to change the rate.

5.8.3 DataMove.API.ValidateSizes(Name As %String) As %Status

Makes sure sufficient space exists for the data specified by the DataMove to be copied.

Argument:

- *Name* is the name of the DataMove .

Validating sizes for a DataMove involves determining the amount of data to be copied and ensuring enough space exists in the destination database. Any errors are reported in the status.

ValidateSizes() is called as part of StartCopy(). You can use this method to validate the DataMove size requirements before you actually start the copy with StartCopy()

5.8.4 DataMove.API.StopCopy(Name As %String) As %Status

Stops the DataMove copy job.

Argument:

- *Name* is the name of the DataMove.

This method stops the DataMove copy background jobs, allowing you to gracefully stop the copy after it is in process. You can restart the DataMove with StartCopy().

5.8.5 DataMove.API.Rollback(Name As %String) As %Status

Rolls back the DataMove.

Argument:

- *Name* is the name of the DataMove.

This method deletes any globals that have been copied to destination databases by the DataMove copy job. This method can be used to abort the DataMove or recover from an error in the copy process.

StopCopy() must be run before calling this method.

After doing the rollback, you can start over with StartCopy() or delete the DataMove with Delete().

5.8.6 DataMove.API.RollbackMappings(Name As %String)

Rolls back the DataMove mappings.

Argument:

- *Name* is the name of the DataMove.

This will restore the systems mappings to the state before the DataMove *Name* was run. The DataMove *Name* must be in the State $$$DMStateNSPActivateDone. An example of using this would be if you started to test the application after the DataMove activates its mappings, and detected there was something wrong. After the mappings are restored to their previous value, the State will be set to $$$DMStateJrnApplyDone. From here you can call Rollback() (recommended), or StartCopy() and retry the activation.

5.8.7 DataMove.API.RollbackCopy(Name As %String, Warnings as %String, Errors as %String)

Rolls back a copy of the DataMove.

Arguments:

- *Name* is the name of the DataMove.
- *Warnings* is an array returned with conflicts that do not prevent the DataMove from being performed.
- *Errors* is an array returned with conflicts that do prevent the DataMove from being performed.

This will create a DataMove called Name-ROLLBACK which when run will move the data copied in the DataMove *Name* back to its original source directories. The DataMove *Name* must be in the State $$$DMStateDeleteSrcGlobalsDone (all the source globals have been deleted by the DeleteSourceGlobals() method. After the method finishes, you can then run StartCopy() and Activate() to move the data back to their original directories. The DataMove *Name* will have its State set to $$$DMStateRollbackCopy.

5.8.8 Query DataMove.API.ListDMs(Names As %String, Flags as %Integer=0) As %Status

Query which returns a list of all DataMoves and their properties.

Arguments:

- *Names* is a commas separated list of DataMoves.
- Flags –
  - 0 = Return current information in the DataMove record
  - 1 = Return information summed across all the ranges

5.8.9 Query DataMove.API.ListRanges(Name As %String, SrcDBs As %String = "*", DstDBs As %String = "*", Ranges As %String = "*", Flags = 0) As %Status

Query which returns a list of all the DataMove ranges and their properties.

Arguments:

- *Names* is the name of the DataMove.
- SrcDBs – comma separated list of source databases
- DstDBs – Comma separated list of destination databases
- Ranges – Comma separated list of ranges
- Flags –
  - 0 = Return formatted times
  - 1 = Return times as $h UTC times

5.8.10 Query DataMove.API.ListProcesses(Name As %String) As %Status

Query which returns a list of all the DataMove processes.

Argument:

- *Name* is the name of the DataMove.

6 DataMove States

The DataMove keeps track of its progress through the workflow by means of the State property. You can inspect this property to monitor its progress using GetProperties(), or to troubleshoot any issues that arise, using the query ListDMs().

The integer values for the each state are defined in the **%syDataMove.inc** include file.

Table 1: List of DataMove States

State	Description
$$$DMStateNotStarted	Generate() has been called to create the DataMove.
$$$DMStateStarted	StartCopy() has been called to start the copy.
$$$DMStateSize	The size of the data to be moved is being calculated.
$$$DMStateSizeDone	The size of the data to be moved has been calculated.
$$$DMStateCopy	The source data is being copied to the destination databases.
$$$DMStateCopyDone	The source data has finished being copied to the destination database.
$$$DMStateJrnApply	All the source data has been copied to the destination databases, and the journals are now being applied to the destination databases. It will remain in this state until Activate() or StopCopy() is called, or the system shuts down cleanly.
$$$DMStateJrnApplyDone	DataMove processes have halted, either during a call to Activate() or StopCopy().
$$$DMStateNSPActivate	Activate() has been called. Switch 10 has been set, and the new namespace mappings are being activated.
$$$DMStateNSPActivateDone	The new namespace mappings have been activated.
$$$DMStateDeleteSrcGlobals	DeleteSourceGlobals() has been called, and the source globals are being deleted.
$$$DMStateDeleteSrcGlobalsDone	All of the source globals have been deleted.
$$$DMStateDone	Finish() has been called. All data has been moved to the destination databases, namespaces activated and the final log file updated.

State	Description
$$$DMStateRollback	The Rollback() method has been called, and all the destination globals which have been copied are being deleted. When complete the state will be set to $$$DMStateNotStarted.
$$$DMStateRollbackCopy	The RollbackCopy() method has been called. This DataMove was completed, and a new DataMove was created from this which will move the data back to its previous location.

7 DataMove Example

This ObjectScript routine moves globals ABC and DEF in namespace LIVE to database LIVE-CT using a DataMove object named DataMover.

For demonstration purposes, this example does not include comprehensive error checking. Check the returned status after each API call.

```
#include %occStatus
#include %syDataMove
#include %syConfig

#; Sample routine to illustrate using DataMove to move globals.
Example() {
    #; New Mapping definition
    Set NewMap = 1
    Set NewMap(1, "name") = "ABC"
    Set NewMap(1, "db") = "LIVE-CT"
    Set NewMap = 2
    Set NewMap(2, "name") = "DEF"
    Set NewMap(2, "db") = "LIVE-CT"

    #; Let's start initializing the edits space before moving the globals
    Write "Initialize edits........."
    Set Namespace = "LIVE"
    Set Status = ##class(DataMove.API).MapInitialize("DBNAME",Namespace)

    #; Now we need to create the new mappings
    Write "Create/Update maps........."

    #; Main loop to define mappings and operations to be done.
    For i = 1:1:NewMap {
        Kill Obj, Name, Prop, Status
        Set Prop("Database")=NewMap(i, "db")
        Set Name=NewMap(i, "name")

        #; Write name after a new line
        Write !, Name

        #; Checking if the global exists
        Do ##Class(Config.MapGlobals).Exists(Namespace, Name, .Obj)
        If (Obj = "") {
            #; Creating the global
            Write !, "  - Create........."
            Set Status = ##class(DataMove.API).MapGlobalsCreate("DBNAME",Namespace,Name,.Prop)
        } Else {
            #; Modifying the global
            Write !, "  - Modify........."
            Set Status = ##class(DataMove.API).MapGlobalsModify("DBNAME",Namespace,Name,.Prop)
        }
    }

    #; We have everything set up, now we need to create the DM object
    Write "Create DM........."
    Set Status = ##class(DataMove.API).Generate("DBNAME",.Properties, .Warnings, .Errors)
    If $data(Warnings) Write ! Zwrite Warnings
    If $data(Errors) Write ! Zwrite Errors
```

```
    #; Creating the background job that needs to be run
    Write "Creating the JOB........."
    Set Status = ##class(DataMove.API).StartCopy("DBNAME")

    #; Monitor the DataMove until we can activate it.
    For {
        #; Monitor state every 5 seconds
        Hang 5
        Set Status=##Class(DataMove.API).GetProperties("DBNAME",.Properties)
       #; Activate will only run if we are applying journals and have 5 MB or less of journal to apply.

        If (Properties("State")=$$$DMStateJrnApply) {
            If (Properties("JRNMBToApply")<=5) {
                Quit
            }
        }
        Write !,"MB to copy: "_Properties("MBToCopy")
        Write !,"MB copied: "_Properties("MBCopied")
        Write !,"Journal MB still to apply: "_Properties("JRNMBToApply")
    }
    Write !, "Activating the new Mappings..."
    Set Display=1
    Set Status=##Class(DataMove.API).Activate("DBNAME",Display)

    #; Delete the source globals that have been copied in the DataMove
    Write "Deleting the old Globals......."
    Set Status = ##class(DataMove.API).DeleteSourceGlobals("DBNAME")

    #; Stopping the DataMover
    Write "Closing the DataMover......."
    Set Status = ##class(DataMove.API).Finish("DBNAME")

    #; Delete the DataMove and clean up any temporary storage
    Write "Deleting the DataMover......."
    Set Status = ##class(DataMove.API).Delete("DBNAME")
}
```

8 Deprecated Calls and States

There have been changes to both calls and states. The following two tables list the old, deprecated calls or states and their associated replacements.

Table 2: List of Deprecated and New DataMove Calls

Old Name	New Name
Config.Map.InitializeEdits	DataMove.API.MapInitialize
Config.MapGlobals.Create	DataMove.API.MapGlobalsCreate
Config.MapGlobals.Delete	DataMove.API.MapGlobalsDelete
Config.MapGlobals.Modify	DataMove.API.MapGlobalsModify
Config.Map.Revert(0)	DataMove.API.RollbackCopy
Config.Map.Revert(1)	DataMove.API.RollbackMappings
DataMove.Data.Activate	DataMove.API.Activate
DataMove.Data.CreateFromMapEdits	DataMove.API.Generate
DataMove.Data.Dispatch(Name, "Delete")	DataMove.API.Delete
DataMove.Data.Dispatch(Name, "DeleteSrcGlobals")	DataMove.API.DeleteSourceGlobals
DataMove.Data.Dispatch(Name, "Finish")	DataMove.API.Finish
DataMove.Data.Dispatch(Name, "Rollback")	DataMove.API.RollBack
DataMove.Data.Dispatch(Name, "Validate")	DataMove.API.Validate

Old Name	New Name
DataMove.Data.Dispatch(Name, "ValidateSizes")	DataMove.API.ValidateSizes
DataMove.Data.DispatchJob(Name, "Copy")	DataMove.API.StartCopy
DataMove.Data.Display()	^DATAMOVE
DataMove.Data.Exists	No replacement
DataMove.Data.StopCopy	DataMove.API.StopCopy

Table 3: List of Deprecated and New DataMove States

Old State	New State
$$$Copy	$$$DMStateCopy
$$$CopyDone	$$$DMStateCopyDone
$$$CPFUpdate	No corresponding state
$$$CPFUpdateDone	No corresponding state
$$$Delete	$$$DMStateDeleteSrcGlobals
$$$Done	$$$DMStateDone
$$$JrnApply	$$$DMStateJrnApply
$$$JrnApplyDone	$$$DMStateJrnApplyDone
$$$NotStarted	$$$DMStateNotStarted
$$$NSPActivate	$$$DMStateNSPActivate
$$$NSPActivateDone	$$$DMStateNSPActivateDone
$$$Size	$$$DMStateSize
$$$SizeDone	$$$DMStateSizeDone
$$$Started	$$$DMStateStarted
No corresponding state	$$$DMStateRollback
No corresponding state	$$$DMStateRollbackCopy