



Using DataMove with InterSystems IRIS

Version 2025.1
2025-06-03

Using DataMove with InterSystems IRIS
PDF generated on 2025-06-03
InterSystems IRIS® Version 2025.1
Copyright © 2025 InterSystems Corporation
All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble®, InterSystems HealthShare®, InterSystems IRIS®, and TrakCare are registered trademarks of InterSystems Corporation. HealthShare® CMS Solution Pack™, HealthShare® Health Connect Cloud™, InterSystems® Data Fabric Studio™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, and InterSystems TotalView™ For Asset Management are trademarks of InterSystems Corporation. TrakCare is a registered trademark in Australia and the European Union.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Congress Street, Boston, MA 02114, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

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

Using DataMove with InterSystems IRIS	1
1 Introduction to DataMove	1
2 DataMove Restrictions	1
3 Preparing to Use DataMove	2
4 The DataMove Workflow	3
5 DataMove in a Mirror or ECP Environment	3
6 Tips for Creating DataMove Mappings	4
7 The DataMove APIs	5
7.1 Create, Modify, and Delete DataMove Namespace Mappings	5
7.2 Generate the DataMove	9
7.3 Start the DataMove	10
7.4 Monitor the DataMove	10
7.5 ^DATAMOVE Utility	11
7.6 Activate Mapping Changes and Finish the DataMove	17
7.7 Delete Source Globals and Finish DataMove	18
7.8 Other API Calls	19
8 DataMove States	21
9 DataMove Example	22

List of Tables

Table 1: List of DataMove States	21
--	----

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](#).

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 will automatically continue running on the new primary node.

For more information on extended global references, see [Namespaces](#).

3 Preparing to Use DataMove

DataMove creates and uses several globals to track its progress and state. These globals are stored in two databases you must create. These databases should be created with the same attributes as the IRISYS database. This includes file and directory permissions and resource names. The **MaxSize** parameter should be 0 (unlimited) and the database needs to be journaled. These databases can grow to be several GB, so ensure there is plenty of disk space allocated for them. The two databases are:

IRISDATAMOVE — Required for running any type of DataMove on mirrored or non-mirrored systems.

IRISDATAMOVEMIRROR — Required only for running a DataMove on a mirrored system. This database needs to be created as a mirrored database and must exist on all members of the mirror including Failover, Async, DR, and reporting members.

These databases need to be created and defined in the CPF file. Here is an example of what to include in your CPF file on a mirrored system:

```
[Databases]
IRISDATAMOVE=C:\irisdatamove\
IRISDATAMOVEMIRROR=C:\irisdatamovemirror\
```

Here is an example of what the attributes of the IRISDATAMOVEMIRROR database should be:

```
%SYS>d ^DATABASE
1) Create a database
2) Edit a database
3) List databases
4) Delete a database
5) Mount a database
6) Dismount a database
7) Compact globals in a database
8) Show free space for a database
9) Show details for a databse
10) Recreate a database
11) Manage database encryptions
12) Return unused space for a database
13) Compact free space in a database
14) Defragment a database
15) Show background database tasks

Option? 9
Database directories? IRISDATAMOVEMIRROR
Device:
Right margin: 80 =>

-----
Directory:                c:\irisdatamovemirror\
MirrorDBName:             IRISDATAMOVEMIRROR
MaxSize:                  0
Size:                    31
Status:                  Mounted/RW
BlockSize:               8192
ClusterMountMode:       0
ClusterMounted:         0
ExpansionSize:          0
LastExpansionTime:      03/03/2023 15:25:25
Mounted:                 1
NewGlobalCollation:     IRIS standard
NewGlobalGrowthBlock:   50
NewGlobalIsKeep:        0
GlobalJournalState:     Yes
NewGlobalPointerBlock:  16
ReadOnly:                0
ResourceName:           %DB_IRISYS
MountedReadOnly:        0
```

```
EncryptedDB: 0
EncryptionKeyID:
Configured DB Name: IRISDATAMOVEMIRROR
Mount Required At Startup: No
```

4 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 the data has been copied, and the journal files have been applied, DataMove will run a DataCheck on the source and destination globals to verify that the data is the same in both databases.
6. After the DataCheck has completed, the new mappings are activated in the namespace.
7. 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 in the `DataMove.log` file in the `/mgr` directory.

5 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. Additionally, update any ECP clients which are connected to the IRIS server. You can use the [iris merge command](#) to help you with this.
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.

6 Tips for Creating DataMove Mappings

Use NamespaceModify() to move routines to their own databases

If you have a namespace with a database where routines and globals are combined and you want to split the routines into their own routine database, use the [NamespaceModify\(\)](#) method.

For example, use **NamespaceModify()** method to go from:

```
[Namespaces]
USER=USER
```

To:

```
[Namespaces}
USER=USER,USERRTN
```

Select static subscript ranges

When selecting a subscript range of data to move in a global, it is best to select a range in the global which is very static.

For example, if you have a global ^SALES, with a structure similar to:

```
^SALES=n
^SALES(1)=record
^SALES(2)=record
...
^SALES(n)=last_record
```

And, you are constantly adding data to the end of the global, ^SALES(n+1), ^SALES(n+2),... In this case, it is best to move a range from ^SALES(1):SALES(x) rather than from SALES(x):SALES(END). This can reduce the amount of journal data generated during the duration of the DataMove and significantly minimize the time the system takes to run the Activate() method.

Move large globals and global ranges one at a time

If you are moving a large global or global range to a new database, it is best to only move the single global or global range into that database. Do not move other globals into the same database during the same DataMove. This will effectively defragment that global and will improve the speed of integrity checks run against that database.

Manage system resources

DataMove can consume significant system resources. You can use the DataMove.API.Modify() method to increase or decrease the amount of data being moved or checked per minute. For example, during peak system hours, you can set the amount of data to move to 500mb per minute and change it to a higher level during off-peak hours. If you are moving Terabytes of data, the DataMove operation may take a week or more to run. If the databases containing the DataMove globals being moved are mirrored, you should monitor the latency between your mirror nodes.

Monitor journal activity

You will need to monitor your journal activity. DataMove generates significant amounts of journal data during its operation. For every global node which is moved, a journal record will be created. You will need to plan for this and may need to adjust your journal backup and purge schedule to avoid running out of disk space. You should also evaluate using the option to [compress journal files](#) to reduce journal size.

7 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: DataMove operations may move and delete large amounts of data and can take a significant amount of time to complete. Because of this, InterSystems recommends you take a full backup of your system at the following points in the DataMove workflow:

1. Immediately before you call **DataMove.API.StartCopy()**
2. Immediately before you call **DataMove.API.Activate()**
3. Immediately before you call **DataMove.API.DeleteSourceGlobals()**

A full backup includes the /mgr directory and all your journal data. These backups should be made *in addition* to your regularly scheduled backups.

Since DataMove can generate a large amount of journal data, it is possible to set your journal archive and purge schedule to a shorter timeframe. DataMove will only allow journal files to be purged which are no longer needed by the DataMove operation. When purging journal files while using DataMove, use only the system provided journal purge methods.

7.1 Create, Modify, and Delete DataMove Namespace Mappings

Important: After creating DataMove mappings, you must not modify any mappings in the CPF file which refer to either the source or destination of these mappings. Other mappings may be added or modified at any time.

7.1.1 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.

Example:

ObjectScript

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

Example:

ObjectScript

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

7.1.2 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.

To modify and delete global mappings or to create mappings for routines and packages, see [Additional DataMove API Methods](#).

Examples:

ObjectScript

```
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](#).

7.1.3 DataMove.API.NamespaceModify(Name as %String, Namespace As %String, ByRef Properties As %String) As %Status

Modifies the default global and routine mapping database for this namespace in the temporary storage area.

Arguments:

- *Name* is the name of the DataMove.
- *Namespace* is the namespace on which you want to perform the DataMove.
- *Properties* is an array of properties needed for this mapping, in particular, the name of the databases used for this mapping.
 - *Properties*("Globals") is the database for globals.
 - *Properties*("Routines") is the database for routines.

Example 1:

The following example shows the basic usage of the API.

ObjectScript

```
Set Properties("Routines")="SALESRTNDB"
;Set the default routine data base for namespace SALES to SALESRTNDB
Set Status = ##Class(DataMove.API).NamespaceModify("ABC1", "SALES", .Properties)

Set Properties("Globals")="SALESGBLDB"
;Set the default global data base for namespace SALES to SALESGBLDB
Set Status = ##Class(DataMove.API).NamespaceModify("ABC1", "SALES", .Properties)
```

Example 2:

This is an example of a routine which will create a new empty routine database, add it to the CPF file, and generate a DataMove which will move all routines and classes from the default database of a namespace into the new routine database.

In this example, the namespace USER has its default database as USER. We are going to move all routine and classes existing in USER into the new database NEWRTNDB.

The CPF file initially has the following entries:

```
[Databases]
USER=c:\latest\mgr\user\

[Namespace]
USER=USER
```

After the DataMove is complete, we will have the following entries in the CPF file:

```
[Databases]
USER=c:\latest\mgr\user\
NEWRTNDB=c:\latest\mgr\newrtndb\

[Namespace]
USER=USER, NEWRTNDB
```

This example uses the following variables:

- *DMName* — name of the DataMove
- *Namespace* — namespace where the routines are to be split into a new routine database
- *NewRtnDB* — name of the new routine database
- *NewRtnDir* — directory where the new routine database exists

ObjectScript

```
Main
s DMName="RTNSPLIT"
s Namespace="USER"
s NewRtnDB="NEWRTNDB"
```

```

s NewRtnDir="c:\latest\mgr\NEWRTNDB\"

/*
You do not need to create the database and add it to the CPF here. The new
routine database can be created ahead of time and added to all the mirror members.
*/

w !,"Creating database "_NewRtnDB
i '##Class(%File).DirectoryExists(NewRtnDir) {
  s Ok=##Class(%File).CreateDirectoryChain(NewRtnDir)
  i 'Ok w !,"Error creating Directory "_NewRtnDir
}
i '##Class(%File).Exists(NewRtnDir_"IRIS.DAT") {
  s Status=##Class(SYS.Database).CreateDatabase(NewRtnDir)
  i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) g ExitError
}
i '##Class(Config.Databases).Exists(NewRtnDB) {
  s Properties("Directory")=NewRtnDir
  s Status=##Class(Config.Databases).Create(NewRtnDB,.Properties)
  i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) g ExitError
}
w !,"Creating DataMove "_DMName
s Status=##Class(DataMove.API).MapInitialize(DMName,Namespace)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) g ExitError
#;Now create the mapping for the DataMove
k Properties
s Properties("Routines")=NewRtnDB
s Status=##Class(DataMove.API).NamespaceModify(DMName,Namespace,.Properties)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) w !,$$SYSTEM.Status.GetErrorText(Status)
g ExitError
s Status=##Class(DataMove.API).Generate(DMName,.Properties,.Warnings,.Errors)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) g ExitError
w !,"DataMove creation succeeded"
q
ExitError
/* On failure, delete the DataMove */
w !,"DataMove creation failed"
s Status=##Class(DataMove.API).Delete(DMName)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status)
q
Error
s $zt=""
w !,$$SYSTEM.Status.GetErrorText($$$ERROR($$$ObjectScriptError,$ze))
g ExitError

```

7.1.4 Additional DataMove API Methods

Additional DataMove APIs are available that allow you to modify and delete global mappings and to operate on routines and packages. For more information on global, routine, and package mappings, see [Adding Mappings to a Namespace](#).

MapGlobalsModify

Modifies a global mapping for this namespace in the temporary storage area.

MapGlobalsDelete

Deletes an existing global mapping for this namespace in the temporary storage area.

MapRoutinesCreate

Creates a new routine mapping for this namespace in the temporary storage area.

MapRoutinesModify

Modifies an existing routine mapping for this namespace in the temporary storage area.

MapRoutinesDelete

Deletes an existing routine mapping for this namespace in the temporary storage area.

MapPackagesCreate

Creates a new package mapping for this namespace in the temporary storage area.

MapPackagesModify

Modifies an existing package mapping for this namespace in the temporary storage area.

MapPackagesDelete

Deletes an existing package mapping for this namespace in the temporary storage area.

7.2 Generate the DataMove

7.2.1 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("MaxMBCheckPerMin")` optionally specifies the maximum number of MB per minute the DataMove DataCheck operation is allowed to check.
Setting this to 0 allows the DataMove DataCheck to run as fast as it can. If not passed, it used 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:
 - `$$$BitNoSrcJournal`, allow copying of non-journaled databases.
 - `$$$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.

Example:

ObjectScript

```

Set Properties("Flags")= $$$BitCheckActivate
Set Status=##Class(DataMove.API).Generate("TEST",.Properties,.Warnings,.Errors)
If $System.Status.GetErrorCodes(Status)[$$$ERRORCODE($$MGBLHasWarnings)
{ f i=1:1:Warnings w !,Warnings(i) }

```

For more information on database journaling, see [Journaling](#).

For more information on running a process in batch mode, see [Process Management](#).

The `Warnings` array contains a list of mappings where the data being moved is also mapped from another namespace. When warnings are returned, the DataMove is still generated and the status of the `Generate()` method returns

\$\$\$MGBLHasWarnings. A warning will include the other namespace which has mappings to the global range. To preserve access to the global range from the other namespace, you should manually update the mappings.

The `Errors` array contains a list of errors where there are conflicts in the generated mappings. A conflict is typically generated when you create a mapping to a destination database and, at the same time, create a second mapping from the destination database which overlaps the first mapping.

For example, assume the following values in the CPF file:

```
[Map.USER]
Global_A=USER
Global_A("SLMA")=USER1
```

If you are trying to move `A("SLMA")` back to `USER` and `A("SLMB")` to `USER1`, then the following code will generate a conflict error:

ObjectScript

```
Set Status = ##class(DataMove.API).MapGlobalDelete(DMName, "USER", "A("SLMA")")
Set Properties("Database") = "USER1"
Set Status = ##class(DataMove.API).MapGlobalsCreate(DMName, "USER", "A("SLMB")", .Properties)
```

7.3 Start the DataMove

7.3.1 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.

After the copy is complete, `StartCopy()` will call `DataCheck` to verify that the source and destination data are the same. Once `DataCheck` successfully completes, `Activate()` can be called.

7.4 Monitor the DataMove

7.4.1 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("MaxMBCheckPerMin") – Maximum number of MB per minute the DataMove DataCheck operation is allowed to process.
- 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("MBChecked") – Amount of MB that have been DataChecked. After the DataCheck completes, this value will be more than the size of the data copied.
- 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( "ExpandedState" )="ActivateReady/Run"
Properties( "JRNMBToApply" )=0
Properties( "MBChecked" )=25622
Properties( "MBCopied" )=24722
Properties( "MBToCopy" )=24722
Properties( "MaxMBCheckPerMin" )=0
Properties( "MaxMBPerMin" )=0
Properties( "State" )=10
Properties( "StateExternal" )="ActivateReady"
Properties( "Status" )=1
Properties( "Stop" )=0
```

7.5 ^DATAMOVE Utility

The ^DATAMOVE utility displays the current state of all DataMoves defined on the system and allows users to set DataMove default parameters. Multiple DataMoves can be running at the same time.

7.5.1 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
```

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 => 5000
Do you want to apply this change to existing Data Moves? Yes => Yes
```

Setting the maximum number of MB DataMove DataCheck is allowed to check per minute to 0 means that DataMove DataCheck will check 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 DataCheck without changing the system default.

```
Maximum number of MB to check per minute? 0 => 8000
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
```

7.5.2 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.

```

DATAMOVE InterSystems IRIS TRM:1148
File Edit Help
DataMove Monitor
# Name          MBToCopy  Copied Pct  Checked Pct  JrnMB  State
1 ABC2          24722     24722 100    25622 100    0 ActivateReady/Run
Status: OK

                                Done/Avoid/Copy
Last Updated:    2023-04-14 14:39:26.330 Sets:          0/0/0
Est Finish Time: 2023-04-11 15:40:27.520 Kills:           0/0/0
Jrn Cycle Time: 00:00:00.004 BitSets:          0/0/0
Jrn Cycle End:   2023-04-14 14:39:26.330 ZKILLS:         0/0/0
Jrn Cycle Start: 2023-04-14 14:39:26.325 Max MB/Min:      Unlimited
DataCheck Time: 00:01:19.326 MB/Min:           9844
DataCheck End:   2023-04-11 15:50:44.483 Blks copied:    3164349
DataCheck Start: 2023-04-11 15:49:25.156 Nodes Checked:  5368310
CopyTime:        00:02:30.682 JRN Count/Size: 0/0
End Copy:        2023-04-11 15:40:17.520 Pid Mov/Jrn:   12800/21992
Start Copy:      2023-04-11 15:37:46.837 Pid Copy/Chk:  4700/37204
Size Time:       00:00:00.579
End Size:        2023-04-11 15:37:41.441
Start Size:      2023-04-11 15:37:40.861
Journal Start:   c:\datamove\mgr\journal\20230411.103 210540
Journal Current: c:\datamove\mgr\journal\20230414.002 76482548

(R)anges, (J)obs, (M)etrics, (D)ataCheck, (H)alt, (A)ctivate, (Q)uit => █

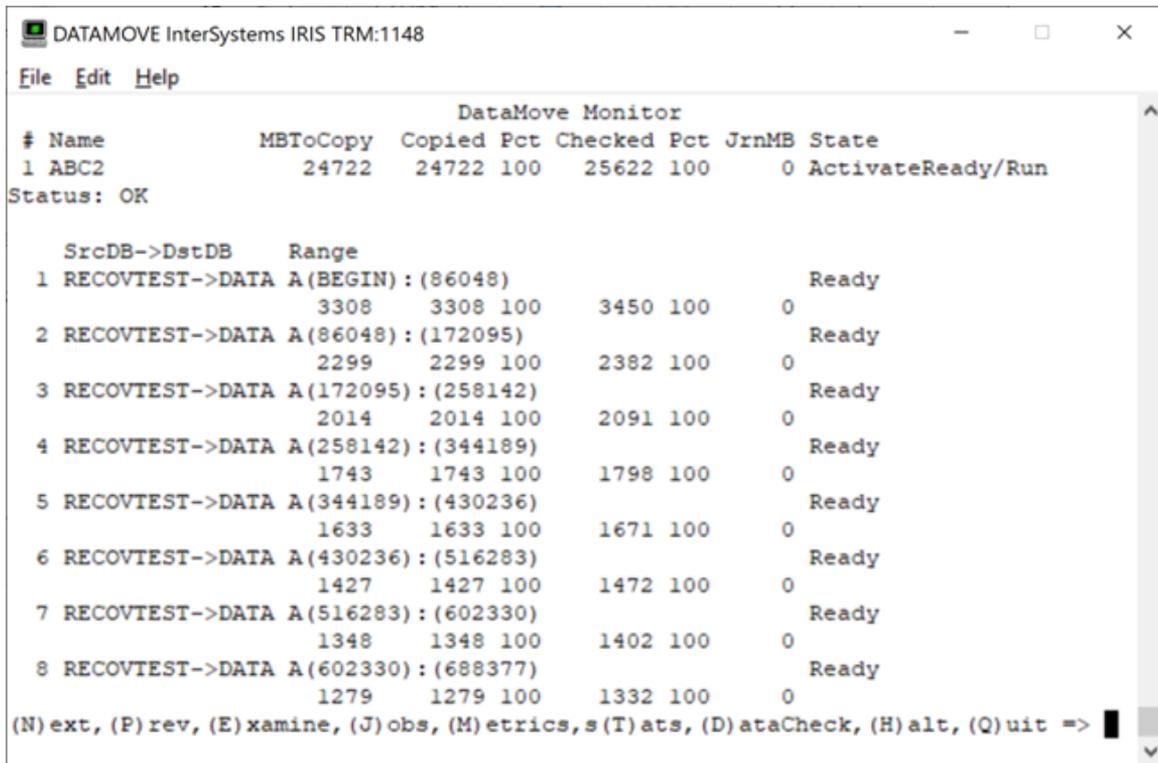
```

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.
- `DataCheck Time` – Amount of time it took for the DataCheck to complete.
- `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.
- `Nodes Checked` – Number of global nodes which were DataChecked.

- 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/Chk – Process ID of the Master DataMove process and Master DataCheck Process.

When you select the range option, you can see that there are 8 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.

```

DATAMOVE InterSystems IRIS TRM:1148
File Edit Help

DataMove Monitor
# Name          MBToCopy  Copied Pct  Checked Pct  JrnMB  State
1 ABC2          24722    24722 100    25622 100    0 ActivateReady/Run
Status: OK

SrcDB->DstDB    Range
2 RECOVTEST->DATA A(86048) : (172095)
                2299    2299 100    2382 100    0 Ready
Last Updated:  2023-04-14 14:46:49.547 Done/Avoid/Copy
Jrn Cycle Time: 00:00:00.009 Sets: 0/0/0
Jrn Cycle End:  2023-04-14 14:46:49.547 Kills: 0/0/0
Jrn Cycle Start: 2023-04-14 14:46:49.538 BitSets: 0/0/0
DataCheck Time: 00:01:04.836 ZKILLS: 0/0/0
DataCheck End:  2023-04-11 15:50:36.901 MB/Min: 1028
DataCheck Start: 2023-04-11 15:49:32.064 Blks copied: 294260
CopyTime:       00:02:14.246 JRN Count/Size: 0/0
End Copy:       2023-04-11 15:40:06.467 Pid Mov/Jrn: 8756/34072
Start Copy:     2023-04-11 15:37:52.221
Size Time:      00:00:00.138
End Size:       2023-04-11 15:37:41.384
Start Size:     2023-04-11 15:37:41.245

(N)ext range, (P)rev range, (Q)uit =>

```

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

```

DATAMOVE InterSystems IRIS TRM:1148
File Edit Help

DataMove Monitor
# Name          MBToCopy  Copied Pct  Checked Pct  JrnMB  State
1 ABC2          24722    24722 100    25622 100    0 ActivateReady/Run
Status: OK

# Type Job#  Commands  Globals State  PID Range
1 Copy  44  3721531  227423  HANGW  4700 Copy Daemon
2 Jrn   46  17311739 2122577  LOCKW  21992 Journal Daemon
3 Chk   16  33370481 5808662  HANGW  37204 DataCheck Daemon
4 Jrn   47  20847518 923346  HANGW  25524 A(BEGIN) : (86048)
5 Jrn   49  20827088 922855  HANGW  34072 A(86048) : (172095)
6 Jrn   48  20824897 922796  HANGW  29108 A(172095) : (258142)
7 Jrn   47  20847518 923346  HANGW  25524 A(258142) : (344189)
8 Jrn   47  20847518 923346  HANGW  25524 A(344189) : (430236)
9 Jrn   48  20824897 922796  HANGW  29108 A(430236) : (516283)
10 Jrn  49  20827088 922855  HANGW  34072 A(516283) : (602330)
11 Jrn  50  20827969 922878  HANGW  35500 A(602330) : (688377)
12 Jrn  51  20827131 922867  HANGW  21656 A(688377) : (774424)
13 Jrn  51  20827131 922867  HANGW  21656 A(774424) : (860471)
14 Jrn  50  20827969 922878  HANGW  35500 A(860471) : (946518)
15 Jrn  48  20824897 922796  HANGW  29108 A(946518) : (1032565)
16 Jrn  47  20847518 923346  HANGW  25524 A(1032565) : (1118612)

(N)ext, (P)rev, (R)anges, (M)etrics, (D)ataCheck, (H)alt, (Q)uit =>

```

Jobs with a type of “Copy” are processes which are copying data from the source to the destination database. Jobs with a type of “Jrn” are applying journal transactions to the destination database. Jobs with a type of “Chk” are running DataChecks between the source and destination databases. 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.

```

MIR2 InterSystems IRIS TRM:18800
File Edit Help

DataMove Monitor
# Name          MBToCopy Copied Pct Checked Pct JrnMB State
1 ABC2          20034   9684 48      0  0  1074 Copy/Run
Status: OK

Total          DataMove  DM %
Global references/Sec:      19905      19724  99
Global update references/Sec: 19185      19192  100
Physical reads/Sec:        8830       8869  100
Journal Entries/Sec:      19175      19192  100
MB Copied/Min Allowed/Current Unlimited  0
MB Checked/Min Allowed/Current Unlimited  0

Mirror Member Name      Member Type      Status      Dejournaling
/MIR1                   Disaster Recovery Connected 30 seconds behind

(R)anges, (J)obs, (D)ataCheck, (C)hange rate, (H)alt, (Q)uit =>
    
```

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 most of the 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:

```

MIR2 InterSystems IRIS TRM:18800
File Edit Help

DataMove Monitor
# Name          MBToCopy Copied Pct Checked Pct JrnMB State
1 ABC2          20034   17665 88      0  0  0 Copy/Run
Status: OK

Total          DataMove  DM %
Global references/Sec:      4746       4402  93
Global update references/Sec: 4349       4236  97
Physical reads/Sec:        3446       3287  95
Journal Entries/Sec:      4339       4236  98
MB Copied/Min Allowed/Current Unlimited  2394
MB Checked/Min Allowed/Current Unlimited  0

Mirror Member Name      Member Type      Status      Dejournaling
/MIR1                   Disaster Recovery Connected 12 seconds behind

Max allowed MB Copied/Min is currently set to: 0-Unlimited
New value 0 (0=Unlimited) => 1200
    
```

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

```

MIR2 InterSystems IRIS TRM:18800
File Edit Help

DataMove Monitor
# Name          MBToCopy  Copied Pct  Checked Pct  JrnMB  State
1 ABC2          20034    19385 97         0 0      0 Copy/Run
Status: OK

                Total      DataMove  DM %
Global references/Sec:      192         1      1
Global update references/Sec:  15         0      0
Physical reads/Sec:         0         0      0
Journal Entries/Sec:        7         0      0
MB Copied/Min Allowed/Current  1200      1153
MB Checked/Min Allowed/Current Unlimited  0

Mirror Member Name      Member Type      Status      Dejournaling
-----/MIR1           Disaster Recovery Connected Caught up

(R)anges, (J)obs, (D)ataCheck, (C)hange rate, (H)alt, (Q)uit =>

```

7.6 Activate Mapping Changes and Finish the DataMove

7.6.1 DataMove.API.Activate(Name As %String, Display as %Boolean = 1, Timeout As %Integer = 120, Force as %Integer=0) As %Status

Finishes the DataMove and activates the namespace mapping changes. Activate() will not run until the DataCheck() completes.

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.
- *Force* will start the namespace mapping activation even if the journals have not caught up. Setting the *Force* flag will cause the activate process to spend extra time applying the last of the journal files while switch 10 is set.

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. 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, `DataCheck()` has completed, and the journal apply is caught up before proceeding.

7.6.2 Update Mappings on Mirror Members and ECP Clients

After you have activated the new namespace mappings on the system, you can update the mappings on all other mirror members and ECP clients. `DataMove` generates a file in the `MGR` directory with the name: `DataMove_DataMoveName_Actions.cpf`. This file contains a set of actions which can be merged into the mirror and ECP CPF files. Below is an example of the file's format:

```
[Actions]
CreateMapGlobal:Namespace=SALES,Name=A(86048):(172095),Database=DATA1
CreateMapGlobal:Namespace=SALES,Name=A(172095):(258142),Database=DATA2
CreateMapGlobal:Namespace=SALES,Name=A(258142):(344189),Database=DATA3
CreateMapGlobal:Namespace=SALES,Name=A(344189):(430236),Database=DATA4
```

Examine this file closely. It should contain the set of mappings which have been newly applied to your system's CPF file. You can then copy this file to your other mirror members and ECP clients and use the [iris merge command](#) to insert and activate the mappings into the CPF file on those systems.

Important: Make sure that the mirror members are caught up with all the journaled `DataMove` data before applying the mappings to them.

7.7 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`.

7.7.1 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.

7.7.2 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.

7.7.3 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.

7.8 Other API Calls

7.8.1 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()

7.8.2 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.
 - `Properties("MaxMBCheckPerMin")` – Maximum number of MB per minute the DataMove DataCheck operation is allowed to check. Setting this to 0 allows the DataMove DataCheck to run as fast as it can. This can be called on a running DataMove to change the rate.

7.8.3 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()

7.8.4 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().

7.8.5 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().

7.8.6 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 \$\$\$DMStateReady. From here you can call Rollback() (recommended), or StartCopy() and retry the activation.

7.8.7 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.

7.8.8 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

7.8.9 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

7.8.10 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.

8 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
\$\$\$DMStateCreate	The DataMove is in the middle of being created.
\$\$\$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.

State	Description
\$\$\$DMStateCopyDone	The source data has finished being copied to the destination database.
\$\$\$DMStateJournal	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 the DataCheck starts running.
\$\$\$DMStateDataCheck	DataMove processes are running DataCheck
\$\$\$DMStateReady	The DataMove has completed the DataCheck and is ready to activate.
\$\$\$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.
\$\$\$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.

9 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.

```
IRIS for Windows^MAC^^~Format=IRIS.S~^RAW
%RO on 22 Jun 2023 2:33 PM
DMEXAMPLE^MAC^^66647,52367.6489234^0
DMEXAMPLE
#include %occInclude
#include %syDataMove
/*
  This example creates a DataMove called "TEST" which moves pieces of the
  global ^A in namespace TESTDATA into 3 new databases.

  Assume the following entries in the CPF file, where the NEWDATA* databases
  are newly created empty databases, and the DATA database contains the
```

entire global ^A.

```
[Databases]
DATA=c:\iris\mgr\data
NEWDATA1=c:\iris\mgr\newdata1
NEWDATA2=c:\iris\mgr\newdata2
NEWDATA3=c:\iris\mgr\newdata3
```

```
[Namespaces]
TESTDATA=DATA
```

The first subscript in ^A up to ^A(5000) will move into database NEWDATA1.
 ^A(5000) up to ^A(7000) will move into database NEWDATA2.
 ^A(7000) up to ^A(10000) is not moved and remains in the database DATA.
 ^A(10000) through the last subscript will move into database NEWDATA3.

After the DataMove completes, the CPF file will have mappings added to it as follows:

```
[MAP.TESTDATA]
Global_A(BEGIN):(5000)=NEWDATA1
Global_A(5000):(7000)=NEWDATA2
Global_A(10000):(END)=NEWDATA3
```

```
*/
New
s DMName="TEST"
s Namespace="TESTDATA"
#;Initialize the namespace
s Status=##Class(DataMove.API).MapInitialize(DMName,Namespace)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) q
#;Now create the 3 mappings for the DataMove
s Properties("Database")="NEWDATA1"
s Status=##Class(DataMove.API).MapGlobalsCreate(DMName,Namespace,"A(BEGIN):(5000)",.Properties)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) q
s Properties("Database")="NEWDATA2"
s Status=##Class(DataMove.API).MapGlobalsCreate(DMName,Namespace,"A(5000):(7000)",.Properties)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) q
s Properties("Database")="NEWDATA3"
s Status=##Class(DataMove.API).MapGlobalsCreate(DMName,Namespace,"A(10000):(END)",.Properties)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) q
#;Generate the DataMove. Limit the maximum number of MB per minute the
#;DataMove operation is allowed to move to 1GB (60GB/hour)
k Properties
s Properties("MaxMBPerMin")=1000
s Status=##Class(DataMove.API).Generate(DMName,.Properties,.Warnings,.Errors)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) q
#;At this point you could use the ^DATAMOVE utility to start, halt, monitor, and
#;activate the DataMove.
#;Or you can use the API's as follows:
#;Start the copy and dejournal processes
s Status=##Class(DataMove.API).StartCopy(Name)
i '$$$ISOK(Status) w !,$$SYSTEM.Status.GetErrorText(Status) q
#;Monitor state every 10 seconds
#;Activate will only run if the state is "Ready" and there is 20 MB or less
#;of journal to apply.
For {
  Hang 10
  Set Status=##Class(DataMove.API).GetProperties(DMName,.Properties)
  If (Properties("State")=$$DMStateReady) {
    If (Properties("JRNMBToApply")<=20) {
      Quit
    }
  }
  Write !,"State:      "_Properties("ExpandedState")
  Write !,"MB to copy:  "_Properties("MBToCopy")
  Write !,"MB copied:  "_Properties("MBCopied")
  Write !,"MB checked: "_Properties("MBChecked")
  Write !,"Journal MB still to apply: "_Properties("JRNMBToApply")
}
w !,"DataMove is ready to Activate"
q
```

