Unified Model Documentation Paper Y1

Unified Model Automated Output Processing System

I. Edmond

Version No.11

Dated 22 December 1998

Corresponding to version 4.5 of Unified Model

Climate Research
Meteorological Office
London Road
BRACKNELL
Berkshire
RG12 2SY
United Kingdom

Modification Record		
Model version	**Author**	**Description....................**
4.4	L.Wiles	Major rewrite.
4.5	I Edmond	Description of Job Release/Dump deletion/archiving for Regular/Irregular dumping.

CONTENTS

1. Introduction

Climate mode integrations carried out on powerful supercomputers can generate output data at rates which makes it necessary for results to be processed and housekeeping tasks carried out as the model is running. Failure to do this would soon lead to model failure due to disk space shortages, a backlog of archiving and delays in analysis. Furthermore, the amount of staff time involved in manually building up and submitting jobs to do these tasks is prohibitive for all but very short experiments.

2. Overview of Slave Process

The model initiates output post-processing by sending output processing requests direct to the slave process, which acts as a server for the model itself (see UMDP C0 - The top-level control system, Figures 2(a) and 2(c)). In order to minimize the memory requirement for the combined model-plus-server, only unix facilities (except FIELDCOS for processing pp files) are used in the slave process since these are generally less memory-demanding than FORTRAN programs. An extra benefit is that the software for the slave process will be much more portable.

The interface between the model and the slave process is defined by an agreed protocol for output processing requests. This is set out in more detail in section 5, but consists basically of a one-line command for each file to be processed containing the name of the file on the Cray, plus a list of processing code options corresponding to standard post-processing actions (eg. archive on IBM or CRAY to cartridge, delete from Cray disk, transfer to IBM disk, etc).

No explicit synchronization between the model and the slave process is required if the model only sends requests to the slave which can be actioned independently of the model. The overall model design is such that this is achievable. The actual mechanism for transmitting requests is a standard UNIX pipeline, since this allows concurrent execution of the two processes but with a transparent link between them.
Output processing requests are embedded in the standard output from the model, and are filtered out from the rest of the output and actioned by the slave as and when they are detected.

3. The Output Processing History File

3.1 Purpose

This file holds output processing information that is used by the model slave process to control the processing of requests issued by the Unified Model. The Unified Model history file cannot be used for this information as it does not wait for output processing requests to be completed and therefore would not necessarily have an up to date record of the status of output processing.

3.2 Naming Convention

Refer to Unified Model Documentation Paper No 7

3.3 Origin/Lifetime

The Unified Model User Interface produces a PPCNTL file in a Front-End Library for a new run. Users can modify certain values during following runs. The runid.pphist file is set up by a new run and must be retained throughout the life time of an experiment.

3.4 Structure/Updating

The file consists of a series of single line records of the following format :

```
                    History variable=value
```

These values are assigned to corresponding slave process script variables by simply executing the file as a script.

During processing,the slave job modifies certain output processing script variables and, on completion of each output processing request,it updates the output processing history file, pphist. This will be achieved by overwriting the existing file using current relevant script variable values.

4. Output Processing Requests

4.1 Purpose

Output processing requests define front end processing and back-end housekeeping actions on files generated by the Unified Model as it runs.

4.2 Origin of Requests

Requests are issued by the model control routines as the model executes and automatically 'piped' as input to a slave process. The slave process runs in parallel with the model and requests are dealt with sequentially until the model completes and the pipe is closed.

Job Release

Job Release operates on distinct set of timesteps independent of dump deletion/archiving. Job release only depends on the values set in JOBREL_STEPim array variable irrespective of whether there is regular or irregular dumping set. (Negative values in JOBREL_STEPim indicate periodic release with a period determined by the modulus of its value).

Dump deletion

There are no differences in the "Dump deletion" for regular and irregular dumping intervals and depends only on the "dumptimes". However, if Climate Meaning is selected, dump deletion is dependent on both the dumping frequency and meaning period. For irregular dumping, STASH/Climate meaning is not available and dump deletion is determined by the dumptimes only.

Archiving requests

Archiving operates independently from dump deletion. The archiving frequency of dumps is expressed as a multiple of restart dump occurrences for regular dumping. Climate mean dumps have there own archiving control switches as they are almost never required to be archived.
For irregular dumping, all dumps will be archived if archiving is requested. To enable this functionality, the user must manually set ARCHDUMP_FREQim =1,1,1,1 in the job library. (The UMUI currently initialises this variable to zero).

4.3 Format of Requests

Requests have the following general format :

```
Request
Identifier     Cray Filename     Disp of file     Type of file

Note  (1)           (2)               (3)               (4)
```

Notes

(1) The string %%% (in cols 1-3) is used and is the same for all requests. This distinguishes output processing requests from other model output data.

(2) (Mandatory variable)
This is not the full pathname , but the file name in the Unified Model output data directory as specified in the DATAM environment variable in the Unified Model job script.

(3) (Mandatory variable)
The following dispositions are currently recognized

'DELETE'	: Delete named file from Cray disk
'ARCHIVE'	: Archive named file on frontend tape
'REL'	: Execute (Release) user supplied output processing script.

(4) File type
The following types are mandatory for disposition = ARCHIVE.

'DUMP'	: Model dump file
'PPNOCHART'	: Model (including mean) PP file
'BNDY'	: Model boundary file

4.4 Examples

a) Archive 'atmosphere' model restart dump cai7na.daj1904 to front end tape

```
%%% cai7na.daj1904 ARCHIVE DMP
```

b) Delete restart dump cai7na.daj1904 from Cray disk

```
%%% cai7na.daj1904 DELETE
```

c) Trigger execution of user supplied output processing script

```
%%% A_JOB_MEMBER_1 REL    NET
```

If the request is for user script release then type "NET" is written to the requests file.

This type is presently unused as the server/archiving scripts only recognise request types DUMP, PPCHART, PPNOCHART or BNDY which are used to set up " Archiving Streams" to control how the data is written to the Camelot databases and archiving request files. See http://www-hc/~hadkr/UMinfo/archstreams.html for advice on setting up Archiving streams for CR UM runs.

5. Details of the Slave Process

Purpose of Slave Process : To detect, decode and process sequentially each output processing request issued by the Unified Model

The slave process consists of a number of UNIX scripts that run on the Cray.

5.1 Main functions performed by the Slave Process

a) Disk space management on Cray disk files generated by model.
ie Delete model files from Cray disk as requested by the model.

b) Transfer model PP field, Restart and Dump files to front-end transfer disk.

c) Execute user supplied output processing scripts when required

d) Update model output processing history variables as requests are dealt with and recreate the output processing history file on completion of each request.

e) Copy contents of output processing history file to output when model has completed its run.

f) Create a list of files in a 'trigger file' that need to be copied to cartridge from the front-end and transfer this to the front end.

Upon successful data file transfer, a 'trigger file' is updated with details such as secondary system dataset name, expiry date of archived file, a code for the type of tape to be used, the users accounting code and userid.

Upon reaching a maximum file number per trigger file, which is requested by the user, the trigger file is then transferred to the secondary system.

An Archive Server Function is active on the secondary system and deals with the generation and submission of Archive tasks. Such tasks examine the trigger file contents, then carry out archiving procedures. At the Met. Office, the archive server runs on the IBM MVS system.

These scripts can be used on other systems, as well as at the Met. Office. However, on other systems the user must then develop their own archive server on the secondary system.

5.2 Description of scripts used by Slave Process

qsserver -
Top level script for server process which runs in parallel to the model run and archives data to another machine.
The server reads archive requests by a pipe connected to the model. Data files are transferred to the front-end along with trigger files, which are text files detailing archive requirements.

qscasedisp -
Process an archiving request i.e. delete or copy file to secondary system. Also releases user script if necessary and creates trigger files.

copy2dest -
Copy data from the machine that model runs on to a secondary system.

hdstransfer -
To create text required for the transfer of UM data files, to a IBM MVS system.

ppcray2hds -
Process model file using Fortran executable qxfieldcos to convert from Cray to IBM format, and from direct access to sequential format

newpphist -
To initialize variables stored in the post processing history file

archfail -
Copy requests in pipe to a failure file when an error occurs in the server.

autopp_tidyup -
Archives remaining data files to secondary system when a model run is restarted after system or archive server failure.

What Happens if things go wrong ?

5.3 Qsserver Error Messages.

Normally qsserver should not produce error messages. If error messages occur they take the form "qsserver: error message" and appear early in the model output before the message saying end of qsmain. Anyone with this type of error should report it to the owner of the Automatic post-processing scripts. Note that this type of failure could leave a lot of files on the Cray which will need to be tidied up.

5.4 Controlled Failures.

If a model run has problems transferring files to the front end or fails when executing fieldcos the server script will stop in a controlled way and cause the model run to stop. A message will appear in the model output saying the model was stopped due to a qsserver failure and give the date and time. In this situation a failure file called RUNID.failure should appear in the DATAM directory for the run.

The controlled failure works in the following way. The server script is linked to the model run by a file called ERRFLAG. Before a model run the file ERRFLAG is created containing the message

```
"F  No request to stop model run"
```

This file is read by the model each time the subroutine EXITCHECK is called. Provided the first letter in the file remains an F the model continues to run. If the server has a transfer or fieldcos failure it calls the script failure which writes out a new message to the file ERRFLAG of the form

```
"T  qsserver failure at date and time".
```

This message is then read by the model and the run is stopped. At this point the failure script copies the current archive request to a file RUNID.failure. The server continues to read the piped output from the model, reading in each request from the model and copying it to the end of the failure file until the model stops. This way all the post-processing requests since the failure are kept allowing the post-processing to be tidied up.

A script - autopp_tidyup - carries out the following operations at the start of each CRUN:

Search user’s experiment directory for current runid trigger files. Transfer remaining trigger files on Cray in $DATAM to front-end in order to archive data files on disk from previous crashed run. Input contents of .failure file (archive requests from failed run) into qsserver in order to process unarchived data files. Warning: If any old trigger files for this runid remain for any reason, then they will be copied to front-end.