HP Operations Orchestration

For Windows and Linux

Software Version: 9.06

Purging OO Run Histories from MSSQL Databases

Document Release Date: October 2012

Software Release Date: October 2012

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

About Deleting Run Histories

This document is designed to provide a method for pruning old run history data for Central administrators and DBAs involved in the management of the data stored by Central systems.

This document is divided into three main sections:

1. Descriptions of the tables involved in storing historical run data in the HP OO database. See "HP OO Database Tables" on page 7.
2. The procedure for physically deleting old run history data. See "Physically Deleting Data" on page 9.
3. Appendices that contain information such as a diagram of the tables in the **Run** schema, how to upgrade older schemas, and performance implications.

   The code examples shown in the appendices and the script that calls the pruning process are included in text form in the file **MSSQL_RunHistory_Purge.zip** (which also contains this document). The code files are:

   - For Appendix B: Upgrading older schemas:

     `sqlserver_oo_upgrade_history_schema.sql`

   - For Appendix C: Example cleanup stored procedure:

     `sqlserver_oo_prune_run_history.sql`

   - For Appendix D: Example scheduling scripts:

     `sqlserver_oo_schedule_prune_run_history.sql`

Before deciding whether to implement the procedures in this document, read the entire document including "Appendix E: Performance Implications" on page 23.

Required knowledge

Microsoft SQL Server database knowledge is required.

HP OO Database Tables

The tables involved in capturing run history information belong to the OO database. See "Appendix A: Table Diagram" on page 11 for a diagram of the tables in the schema. The tables in the **Run** schema are:

- The **run** table
- The **run_history** table
- The **runstep_history** table
- The **property_history** table
- The **log_record** table
- The **flow_metrics** table

Run Table

The run table stores information about flows that have not yet finished running. Every time a run performs a checkpoint, its current frame stack (including context variables) is placed into a binary object and written to a row in this table. The primary key of the run table is the run id. As soon as a run finishes, the entry in the run table is removed and placed in the **run_history** table.

There are no foreign keys between this table and any other table.

Run_history Table

The **run_history** table stores run information that is used in reporting. There is one row in this table stored for every execution of a flow. The table stores general information about the run, such as its start time, end time, the number of its steps, and how the run ended.

Important: Deleting data from the **run_history** table causes the loss of reporting information. However, if storage space is critical, you can delete data from this table. Just be aware that flows deleted from the **run_history** table will no longer be visible in any reports.

Runstep_historyTable

The **runstep_history** table stores reporting information for each step. There is a one-to-many relationship between the **run_history** table and the **runstep_history** table, enforced by a foreign key relationship between the **runstep_history.run_history_id** and **run.oid** fields, which uses cascading deletes.

Important Deleting data from the **runstep_history** table causes the loss of reporting information for each step of a flow, but the general flow information is still available for reporting. You will not however, be able to "drill down" into the steps which were executed by a flow that has been pruned. However, if storage space is critical, you can delete data from this table. Deleting data from the **runstep_history** table also deletes any related records from the **property_history** table.

> **Note:** Note: OO versions older than 7.20 require schema altering in order to properly support cascading deletes.See "Appendix B: Upgrading Older Schemas" on page 12.

Property_history Table

The property_history table stores a row for each input of a step. There is a foreign key relationship between the fields **property_history.runstep_hist_id** and **runstep_history.oid**, with cascading deletes.

Log_record Table

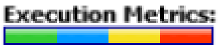
The **log_record** table stores a row for each step input that was designated to be recorded for reporting under a domain-term name. Essentially, it stores a subset of the data in the **property_history** table, but there is no foreign key relationship to the **runstep_history** table. If a run_history row is deleted, rows will also be deleted from the **runstep_history** and **property_history** tables, but the **log_record** table is left intact.

The data in the **log_record** table is used to plot dashboard charts, so deleting data from it will result in loss of dashboard information. This may or may not be a problem depending on how often you prune data. Since dashboard charts are meant to give a more "real-time" picture of what's going on with OO, deleting data from the **log_record** table for a period past where the data is useful for dashboards should be fine.

Flow_metrics Table

The flow_metrics table stores flow outcome counters. There is one entry for each flow, with counters broken down into **Resolved**, **Error**, **Diagnosed**, **No Action Taken**, and **Failed** outcomes, as well as the cumulative time taken by the flows.

This table is used to create the flow metrics bar:



Physically Deleting Data

To delete run histories, use the following approach

1. Upgrade the database schema if necessary (see "Appendix B: Upgrading Older Schemas" on page 12).
2. Establish a timestamp (date and time) when run histories older than it are deleted.
3. Determine how many run histories should be deleted.
4. Divide these run histories into batches to minimize the transaction size.
5. Starting with the oldest batch, delete the batches using one transaction per batch as follows:
   a. Begin the transaction.
   b. Delete data from the **run_history** table, if required.
   c. Update the **flow_metrics** table to reflect the deleted rows, if run histories were deleted.
   d. Delete data from the **runstep_history** table if data was not removed from the **run_history** table.
   e. Delete the rows for the deleted run steps from the **log_record** table, if necessary.
   f. Commit the transaction.

These steps, excluding the first one (upgrading), can be performed on a periodic basis from a scheduled job. An example stored procedure is provided in "Appendix C: Example Cleanup Stored Procedure" on page 13.

You can schedule the cleanup job, as explained in "Appendix D: Example Scheduling Scripts" on page 20.

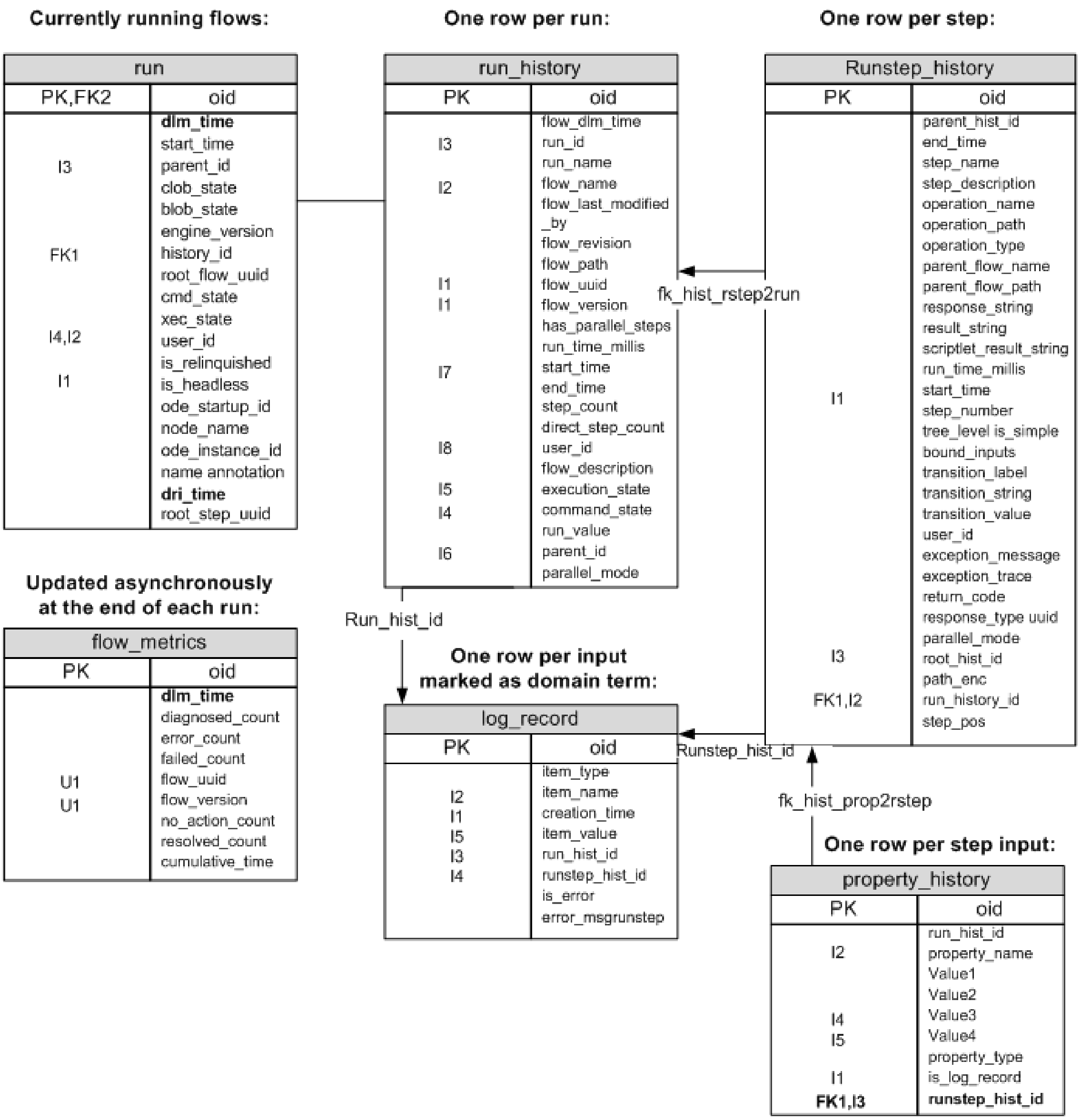
Because an orphaned flow is not considered completed, its related run history is not deleted by the purging scripts.

Appendices

The appendices in this section are meant to help you perform the necessary tasks involved in deleting run histories.

Appendix A: Table Diagram

7.50 Run Schema



Currently running flows:

run	
PK,FK2	oid
	dlm_time
	start_time
I3	parent_id
	clob_state
	blob_state
	engine_version
FK1	history_id
	root_flow_uuid
	cmd_state
	xec_state
I4,I2	user_id
	is_relinquished
I1	is_headless
	ode_startup_id
	node_name
	ode_instance_id
	name annotation
	dri_time
	root_step_uuid

One row per run:

run_history	
PK	oid
	flow_dlm_time
I3	run_id
	run_name
I2	flow_name
	flow_last_modified_by
	flow_revision
	flow_path
I1	flow_uuid
I1	flow_version
	has_parallel_steps
	run_time_millis
I7	start_time
	end_time
	step_count
	direct_step_count
I8	user_id
	flow_description
I5	execution_state
I4	command_state
	run_value
I6	parent_id
	parallel_mode

One row per step:

Runstep_history	
PK	oid
	parent_hist_id
	end_time
	step_name
	step_description
	operation_name
	operation_path
	operation_type
	parent_flow_name
	parent_flow_path
	response_string
	result_string
	scriptlet_result_string
	run_time_millis
I1	start_time
	step_number
	tree_level is_simple
	bound_inputs
	transition_label
	transition_string
	transition_value
	user_id
	exception_message
	exception_trace
	return_code
	response_type uuid
	parallel_mode
I3	root_hist_id
	path_enc
FK1,I2	run_history_id
	step_pos

Updated asynchronously at the end of each run:

flow_metrics	
PK	oid
	dlm_time
	diagnosed_count
	error_count
	failed_count
U1	flow_uuid
U1	flow_version
	no_action_count
	resolved_count
	cumulative_time

One row per input marked as domain term:

log_record	
PK	oid
	item_type
I2	item_name
I1	creation_time
I5	item_value
I3	run_hist_id
I4	runstep_hist_id
	is_error
	error_msgrunstep

One row per step input:

property_history	
PK	oid
	run_hist_id
I2	property_name
	Value1
	Value2
I4	Value3
I5	Value4
	property_type
I1	is_log_record
FK1,I3	**runstep_hist_id**

Appendix B: Upgrading Older Schemas

The following script detects older versions of the schema (HP OO versions 7.0 and earlier) and alters the appropriate tables in order to support cascading deletes. We recommend that you use the text copy of this script contained in the file **sqlserver_oo_upgrade_history_schema.sql** instead of copying the code below, which has line breaks to make reading easier.

```
USE @DBNAME@
GO

DROP PROCEDURE dbo.oo_upgrade_history_schema
GO

CREATE PROCEDURE dbo.oo_upgrade_history_schema
/* This procedure finds out if the db is at some old version that
needs some
 * DDL to implement cascaded deletes and if so, it alters the
foreign keys
 * to have the proper cascades.
 */
AS
    /* find out the build version so we know if we need to do some
schema altering */
    DECLARE @need_alters INTEGER ;
    SET @need_alters = 0;


    SELECT @need_alters=1
      FROM dbo.build_info
     WHERE dri_time IN (SELECT max(dri_time) FROM dbo.build_info)
       AND ((version LIKE '7.0%') OR (version LIKE '7.10%'));

    /* only do this if version is < 7.11 !!! */
    IF (@need_alters <> 0)
    BEGIN

          RAISERROR('Preparing old schema for pruning ...', 0,1) WITH
NOWAIT;

        /** this may fail for some versions because the constraint
has already been dropped. That's ok. */
        ALTER TABLE dbo.runstep_history DROP CONSTRAINT fk_hist_
rstep2parent;

        /** create index if not there already */
        CREATE INDEX idx_hist_prop_runhist_id ON dbo.property_
history(run_hist_id);
```

```
        /* replace some of the foreign keys generated by hibernate
        with the same foreign keys, but with DELETE CASCADE */
        ALTER TABLE dbo.runstep_history
              DROP CONSTRAINT fk_hist_rstep2run;

        ALTER TABLE dbo.runstep_history
              ADD CONSTRAINT fk_hist_rstep2run
              FOREIGN KEY(run_history_id)
              REFERENCES run_history
              ON DELETE CASCADE;

        ALTER TABLE dbo.property_history
              DROP CONSTRAINT fk_hist_prop2rstep ;

        ALTER TABLE dbo.property_history
              ADD CONSTRAINT fk_hist_prop2rstep
              FOREIGN KEY (runstep_hist_id)
              REFERENCES runstep_history(oid)
              ON DELETE CASCADE;
    END;

    RETURN NULL;

-- END PROCEDURE
GO
```

Appendix C: Example Cleanup Stored Procedure

The following stored procedure illustrates the points made in the deletion algorithm. We recommend that you use the text copy of this stored procedure contained in the file **sqlserver_oo_prune_run_history.sql** instead of copying the code below, which has line breaks to make reading easier.

```
USE @DBNAME@
GO

DROP PROCEDURE dbo.oo_prune_run_history
GO

CREATE PROCEDURE [dbo].[oo_prune_run_history]
    @keep_this_many_hours INTEGER = 2160, -- 90 days
    @prune_batch_size INTEGER = 1000,
    @prune_run_history varchar(5) = 'false',
    @prune_dashboards VARCHAR(5) = 'true',
    @recompute_flow_metrics VARCHAR(5) = 'true',
    @verbose INTEGER = 1
```

```
/*
* This procedure attempts to prune the run histories whose start_
time
* was prior to a specified date. Please configure the parameters
below
* according to your needs.
*
* The procedure will recompute the counters in the flow_metrics
table.
*
* The script will execute a series of batch deletes to minimize the
* size of the transaction. The default size of the batch is 1000
histories
* at one time (note the records deleted is a lot more than 1000 as
the
* associated run steps and their dependents are also deleted -- in
other
* words, the script deletes 1000 history trees at a time).
*
* PARAMETERS:
*
* @keep_this_many_hours: sets the number of hours of history
retained, relative
*    to the last known start time.
* Histories for runs whose start_time is less than (max(run_
history.start_time)
*
* will be deleted, unless their runs have not completed.
* Defaults to 2160 (90 days).
*       Example: @keep_this_many_hours = 24 -- keep only the last
day.
*
* @prune_batch_size (default 1000): the batch size for pruning, in
terms
* of number of run histories being deleted.
*       Example: @prune_batch_size = 1000
*
* @prune_run_history: if set to 'true', the script will delete
records from
* the run_history table, which in turn deletes from runstep_history
and
* property_history. If set to 'false', the script will leave
records in
* run_history intackt and only delete the runstep_history date (and
* property_history data by cascading deletes)
*
* @prune_dashboards: if set to 'true', the script will also delete
data
```

```
* that supports dashboards. Otherwise it will leave the data in
place,
* but because the supporting histories are deleted, links from the
dashboards
* may not always produce the proper reports.
*      Example: @prune_dashboards = 'false'
*
* @recompute_flow_metrics: if set to 'true', after each batch of
run histories
* is deleted, the flow_metrics table is update to reflect the new
counts.
*      Example: @recompute_flow_metrics = 'false'
*
* @verbose: if set to a value greater than 0, the script will
output messages
* about the progress of deletion, as follows:
*       if @verbose = 1, it outputs basic information at startup
*       if @verbose = 2, it also outputs information about each
batch being
*                        deleted
*       if @verbose >= 3, it also outputs detailed information
about flow
*                        metrics update
*
* RESULT: how many histories were pruned (an integer).
*/
AS
    SET NOCOUNT ON;


    /* set the deadlock priority for this session to low such that
in case
       we disturb real runs, we're the losing party  */
    SET DEADLOCK_PRIORITY LOW;

    -- validate input params
    IF (@prune_batch_size < 2) BEGIN
        RAISERROR('Invalid pruning batch size, must be at least 2
rows',0,1)
               WITH NOWAIT;
    END;
    IF (@keep_this_many_hours < 1) BEGIN
        RAISERROR('Invalid time window, must be at least 1hr',0,1)
WITH NOWAIT;
    END;

    DECLARE @msg VARCHAR(1000),
            @prune_start_time DATETIME,
            @batch_start_time DATETIME,
            @seconds INTEGER,
            @max_start_time DATETIME,
            @last_start_time DATETIME;

    SET @prune_start_time = GETDATE();

    /* declare local table variable to store the ids for what we
want deleted. */
    DECLARE @oo_pruning_table TABLE (
        oid integer identity not null primary   key,
        run_id numeric(19,0),
        flow_uuid varchar(255) COLLATE DATABASE_DEFAULT,
        execution_state int,
        run_time_millis numeric(19,0));

    /* populate the pruning table with increasing id's taken from
the
     * run_history table. Exclude histories whose runs are still
active
     * inside the run table
     */
    SELECT @max_start_time = max(start_time) from run_history with
(nolock);

    SET @last_start_time = DATEADD(hour, (-1 *  @keep_this_many_
hours),


    IF (@verbose > 0)
    BEGIN
        SET @msg =
               'Preparing pruning table. Will delete histories where
start_time <= '
            + CAST (@last_start_time AS VARCHAR);
        RAISERROR(@msg, 0,1) WITH NOWAIT;
    END;

    INSERT INTO @oo_pruning_table (run_id,
                                   flow_uuid,
                                   execution_state,
                                   run_time_millis)
     SELECT h.oid,
                       h.flow_uuid,
                       h.execution_state,
                       cast(h.run_time_millis as numeric(19,0))
       FROM dbo.run_history h (nolock)
      WHERE h.start_time  <= @last_start_time
```

```
        AND NOT EXISTS
               (SELECT 1 FROM dbo.run r (nolock) WHERE r.history_id =
h.oid)
      ORDER BY h.oid ASC;

    /* this is how many records we try to prune (total) */
    DECLARE @prune_size INTEGER;
    SELECT @prune_size=COUNT(*) FROM @oo_pruning_table;

    IF (@verbose > 0)
    BEGIN
        SET @msg = 'Pruning set size is: ' + CAST (@prune_size as
VARCHAR);
        RAISERROR(@msg, 0,1) WITH NOWAIT;
    END;

    DECLARE @batch_start INTEGER;
    SET @batch_start = 1;

    DECLARE @min_id INTEGER;
    DECLARE @max_id INTEGER;

    WHILE (@prune_size > 0 AND @batch_start <= @prune_size)
    BEGIN
            SET @batch_start_time = GETDATE();
            SET @min_id = @batch_start;
            SET @max_id = @batch_start + @prune_batch_size - 1;

            IF (@verbose > 1)
            BEGIN
              SET @msg = 'Deleting chunk: ' + CAST(@min_id AS
VARCHAR) +
                               ' to ' + CAST(@max_id AS VARCHAR);
              RAISERROR(@msg, 0,1) WITH NOWAIT;
            END;

            BEGIN TRANSACTION;

               IF (LOWER(@prune_dashboards) = 'true')
               BEGIN

                  IF (@verbose > 1)
                       RAISERROR('Deleting dashboard data...' , 0,1)
WITH NOWAIT;

                  DELETE dbo.log_record
                    FROM dbo.log_record l
                  INNER JOIN @oo_pruning_table p
```

```
                     ON ((p.oid BETWEEN @min_id AND @max_id) AND
                                        l.run_hist_id = p.run_id)

               END;

                 IF (LOWER(@prune_run_history) = 'true')
                  BEGIN

                  IF (@verbose > 1)
                               RAISERROR('Deleting run history...' ,
0,1) WITH NOWAIT;
               DELETE dbo.run_history
                       FROM dbo.run_history r
                       INNER JOIN @oo_pruning_table p
                          ON ((p.oid BETWEEN @min_id AND @max_id) AND
                          r.oid = p.run_id)

               IF (@verbose > 1)
                       RAISERROR('Updating flow metrics...' ,0,1)
WITH NOWAIT;

               -- update flow metrics if required:
               IF (LOWER(@recompute_flow_metrics) = 'true')
               BEGIN

               IF (@verbose > 2)
               BEGIN
                       RAISERROR('BEFORE:' , 0,1) WITH NOWAIT;
                       SELECT * from flow_metrics with (nolock);
               END;

                 -- update the flow metrics table to subtract the
                 -- counts for the run history rows that we just
                 -- deleted.
               UPDATE dbo.flow_metrics
               SET diagnosed_count = diagnosed_count - d.diagnosedCount,
                 error_count = error_count - d.errorCount,
                failed_count = failed_count - d.failedCount,
                no_action_count = no_action_count - d.noActionCount,
                resolved_count = resolved_count - d.resolvedCount,
                cumulative_time = cumulative_time - d.cumulativeTime,
                dlm_time = GETDATE()
               FROM dbo.flow_metrics
               INNER JOIN (
               SELECT flow_uuid,
                sum(case
                  when execution_state = 0 then 1 else 0
                  end) as diagnosedCount,
                sum(case
                  when execution_state = 1 then 1 else 0
                  end) as resolvedCount,
                sum(case
                  when execution_state = 2 then 1 else 0
                  end) as noActionCount,
                sum(case
                  when execution_state = 3 then 1 else 0
                  end) as errorCount,
                sum(case
                  when execution_state = 2147483647 then 1 else 0
                  end) as failedCount,
                sum(run_time_millis) as cumulativeTime
               FROM @oo_pruning_table
               WHERE oid BETWEEN @min_id AND @max_id
               GROUP BY flow_uuid
                 ) AS d
                 ON dbo.flow_metrics.flow_uuid = d.flow_uuid AND
                 dbo.flow_metrics.flow_version = 0;

               -- now delete the metrics for those flows that are
               -- left with 0 counts across the board
               DELETE FROM dbo.flow_metrics
                WHERE diagnosed_count = 0
                  AND failed_count = 0
                  AND no_action_count = 0
                  AND resolved_count = 0
                  AND error_count = 0
                  AND EXISTS (SELECT 1 FROM @oo_pruning_table p





               IF (@verbose > 2)
               BEGIN
                       RAISERROR('AFTER:' , 0,1) WITH NOWAIT;
                       SELECT * from flow_metrics with (nolock);
               END;

                       END; -- update flow metrics
               END; -- delete run_history
   ELSE  -- don't delete run histories, just runstep history
       BEGIN
         IF (@verbose > 1)
          RAISERROR('Deleting flow step details...' , 0,1) WITH NOWAIT;

          DELETE dbo.runstep_history
```

```
                FROM dbo.runstep_history r
                INNER JOIN @oo_pruning_table p
                  ON ((p.oid BETWEEN @min_id AND @max_id) AND
                        r.run_history_id = p.run_id);

                       END;


            COMMIT TRANSACTION;

            SET @batch_start = @batch_start + @prune_batch_size ;

            IF (@verbose > 1)
            BEGIN
               SELECT @seconds = DATEDIFF(second, @batch_start_
time, GETDATE());
               SET @msg = 'Batch pruning time was: ' +
                         CAST(@seconds as VARCHAR) + ' seconds';
               RAISERROR(@msg, 0,1) WITH NOWAIT;
            END;
    END;

    IF (@verbose > 0)
    BEGIN
        SELECT @seconds = DATEDIFF(second, @prune_start_time,
GETDATE());
        SET @msg = 'Total pruning time was: ' +
                        CAST(@seconds as VARCHAR) + ' seconds';
        RAISERROR(@msg, 0,1) WITH NOWAIT;
    END;
    RETURN @prune_size

-- END PROCEDURE


GO
```

Appendix D: Example Scheduling Scripts

The following script creates a schedule and job to run the database pruning script on a recurring basis. We recommend that you use the text copy of this script contained in the file **sqlserver_oo_schedule_prune_run_history.sql** instead of copying the code below, which has line breaks to make reading easier.

```
USE @DBNAME@
GO
```

```
/*
 * This script attempts to schedule a run of the oo_prune_run_
history stored procedure.
 * Please set the parameters to your preference.
 */
DECLARE @dbName VARCHAR(128),
        @dbUser VARCHAR(32),
        @jobName VARCHAR(128),
        @jobScheduleName VARCHAR(128),
        @jobDesc VARCHAR(512),
        @jobID UNIQUEIDENTIFIER,
        @jobStartTime VARCHAR(12),
        @ooCommand VARCHAR(1024),
        @hours_retained INTEGER,
        @log_file VARCHAR(1024)


/* how many hours to keep (calculated from the last known start_
time in the table run_history) */
SET @hours_retained = 2;

/* what user do you want the job to run as, defaults to current
user */
SELECT @dbUser = CURRENT_USER;

/* what is the database you want pruned, defaults to current
database */
SET @dbName = DB_NAME(DB_ID());

/* how to call the job */
SET @jobName = 'oo_prune_run_history_job' ;

SET @jobScheduleName = 'schedule for ' + @jobName;

SET @jobDesc = 'Purge run histories older than '
        + CAST(@hours_retained AS VARCHAR)
        + ' hours';

/*
 * where to log the output of the job (you can also see this output
if you view the detailed job history in SQLAgent)
 *
 * Note that the success of writing this log file depends on
 * how the user that runs the procedure may be set up (i.e
 * what type of system user is it associated with, and what
 * type of permissions are there for the file).
 */
SET @log_file = 'C:\tmp\oo_prune_history_job.log';
```

```
/* check if SQLAgent is running */
IF (SELECT count(*)
      FROM Master.dbo.SysProcesses
     WHERE Program_Name = 'SQLAgent - Generic Refresher') = 0
BEGIN
    RAISERROR('Could not schedule procedure oo_prune_run_history:
SQLAgent is not running!', 0, 1)
         WITH NOWAIT
    RETURN;
END

SET @ooCommand = 'EXEC dbo.oo_prune_run_history '
        + '@keep_this_many_hours = '
        + CAST(@hours_retained AS VARCHAR)
        + ', @verbose = 2';

/* create a job */
EXEC msdb..sp_add_job
    @job_name = @jobName,
    @enabled = 1,
    @description = @jobDesc,
    @job_id = @jobID OUTPUT;


/* add target server to the job */
EXEC msdb..sp_add_jobserver
    @job_id = @jobID,
    @server_name = @@SERVERNAME;

-- create step1, actual command
EXEC msdb..sp_add_jobstep
    @job_id = @jobID,
    @step_name = 'Run procedure oo_prune_run_history',
    @subsystem = 'TSQL',
    @command = @ooCommand,
    @database_name = @dbName,
    @database_user_name = @dbUser,
    @output_file_name = @log_file,
    @flags = 2; -- append to output file.

-- start the job 10 minutes from now.
-- Schedule proc requires HHMMSS. CONVERT(8) returns HH:MM:SS
--SET @jobStartTime = REPLACE(CONVERT(VARCHAR, DATEADD(minute, 10,
GETDATE()), 8), ':', '');

-- schedule the job
EXEC msdb..sp_add_jobschedule
```

```
    @job_id = @jobID,
    @name = @jobScheduleName,
    @enabled = 1,
    @freq_type = 4, -- daily
    @freq_interval = 1,
    @freq_subday_type = 8, -- hourly
    @freq_subday_interval = 2; -- every 2 hours

GO
```

Appendix E: Performance Implications

Pruning data from the HP OO database can be hard on your system, and there are a few considerations to take into account before you start using the pruning scripts in this guide. There are a number of parameters that can help mitigate the impact of pruning on your system, but you must carefully choose values for these parameters that match your particular situation. The default values shown in the scripts may not be appropriate for your system.

The main reason pruning data impacts the performance of your HP OO system is that when you delete from a table, the table is locked until the delete is completed. This means that nothing else can modify the data in the table. Since HP OO must update the database for every status change of a flow, the server cannot process new flows or move running flows along until the lock is released—in other words, HP OO is stalled until the delete operation is complete.

The pruning scripts use the **prune_batch_size** parameter to divide the data to be deleted into batches in an attempt to avoid creating locks that last long enough to affect performance. However, you must choose a value for the **prune_batch_size** parameter that is appropriate for your system or the pruning scripts may severely impact performance.

The pruning scripts can also generate a lot of temporary data and eat up CPU cycles. However, there's not much you can do about these problems apart from choosing a time to run the pruning scripts when the server is idle, or not trying to delete too many run histories at once.

Running the Script for the First Time

Before you know how the pruning script will affect your system, you should try to limit the number of rows you try and delete. For example, if your HP OO database spans two years, and you haven't run the pruning script yet, don't just run it with the default values for all of the parameters. You'll end up with a script that's trying to delete nearly two years worth of data, which will take a long time, and if the system becomes unusable you'll likely have to kill the pruning script and possibly have to deal with data corruption. A better plan would be something like the following:

1. Determine the time span of you data, with a command such as:

   ```
   SELECT MIN(start_time), MAX(start_time)

   FROM run_history
   ```

2. Figure out the total number of hours in that time span, and try running the pruning script with **keep_this_many_hours** set to a value that will delete a couple days worth of data. For

example, if your date range is six months, that's 4320 hours. So you would run the pruning script to keep 4220 hours.

3. Keep trying the script with decreasing values of **keep_this_many_hours** until you get it to run for 10 minutes or so, and while it's running try and access the system, or run a flow that has a predetermined execution time, and make sure that the pruning scripts are not interfering with normal HP OO operations. If the server becomes too loaded, adjust **prune_batch_size** and repeat step 3 until you find a value that balances server impact and execution time.

Running the Script on an Ongoing Basis

Once you have the correct value of **prune_batch_size** determined for you system, you need to consider two more things: The frequency which you run the prune script, and the number of hours you want to keep in history. These won't have nearly as great an impact on the server as finding the correct prune_batch_size, but it is still worth considering. It is usually a good idea to run the prune script more often, which means that there will be less data to delete each time you run it. Unfortunately there are no hard-and-fast rules to determine, so the best suggestion is to use the information gathered tuning the 1st run to create a schedule, and monitor it for a few runs to make sure the system is healthy.

Here are some recommendations for using the pruning code:

- Choose a pruning set size that is appropriate to your particular situation. This is important for maintaining the well being of your HP OO system. The number of hours retained should be calculated so that the pruning stored procedure deletes small amounts of history while allowing Central to make progress in running flows.

  Having a higher number for pruning set size can affect database performance, and as a result, flow execution performance metrics will decrease. Having a lower number for pruning set size increases the execution time of the database purging script, but maintains an overall better database performance. The chosen pruning set size should be the highest number for which database performance counters are yielding acceptable values. Depending on the database size, for big database sizes, it is recommended to stop Central, Scheduler services for the duration of the execution of the purging script.

- The stored procedure uses a local table variable, which is allocated out of the temporary tablespace. The table contains IDs for the whole set size, not just for one individual batch. Make sure that there is enough space for it.

- In general, it is better to run the pruning procedure run more often with small batches, than less frequently with larger batches. This helps both Central and SQL Server’s throughput, as the pruning jobs can be interleaved with normal processing jobs.

- Although beyond the scope of this document, note that proper allocation of disk space is important when considering the performance of the database. Having separate physical drives for the database file and the transaction log (separate from the operating system) is a good start. Use the perfmon tool and the SQL Server counters to monitor the disk activity during pruning to establish the impact on the disk resource.

Java™
COMPATIBLE