 **Talend User Components:**

**tJobInstance\* - Collection**

http://www.cimt-ag.de

**Purpose**

This collection consists of:

|  |  |
| --- | --- |
| Component | Purpose |
| tJobInstanceStart | Register a job run and provide information about the previous job run.  Setup the logging facility. |
| tJobInstanceEnd | Deregister the job run, collects KPIs and cleanup the logging setup for this job run |
| tJobDataRangeScanner | Collects min/max time range or values of data flows. |
| tJobInstanceLiveCheck | Checks the entries of the job run registry for dead or broken job instances and clean up the job registry. |

These components help to track the execution of jobs in a database table.

Advantages of these components:

* Provides a unique numeric id for the job to mark all data sets processed by the job
* Start-/Stop timestamps
* Return code and error messages (collects all messages)
* Host and PID of the process running this job
* Supports incremental loading
* Supports restart capabilities
* Key figures about moved data sets
* Snapshot of the context at the start and at the end of the job
* Detects the minimum and maximum of value for a flow
* Enables the usage of Log4J in Talend jobs.

**Talend-Integration**

This component can be found in the palette under Management

This component provides several return values.

**Parameters of tJobInstanceStart**

|  |  |
| --- | --- |
| **Property** | **Content** |
| Database Connection | Any database connection pointing to the schema with the control tables.  The main table is JOB\_INSTANCE\_STATUS holding all key information. |
| Close Connection | You can decide to close the connection in case the tJobInstanceEnd component uses its own connection. That is especially useful for long running jobs. |
| Job Name | Name of the job. The default is using the build-in variable jobName |
| Job Display Name | Human readable name of the job for reporting purposes |
| Process Instance Name | Name of the process instance for reporting purposes |
| Job Work Item | Text describing the work item (e.g. a file name or the date to process by this job) |
| Time range start | If the job has to precede data selected by a time range. This could be used instead of a work item to see what work this job instance do. |
| Time range end | See Time range start. The end of the time range to proceed. |
| Value range start | If the job has to precede a portion of data selected by an id range or any other value ranges. |
| Value range end | See Value range start. This is the end of the range. |
| Write Job instance ID to | To use the job instance id in the job typically a context variable will be used.  Set here the context variable, which should contain the job instance id. |
| Read process instance id from | Jobs can combine to processes. In case of the job does not run as embedded job the process instance if can be read from a context variable. |
| Read ext. job instance id from | In case of need to identify a job via an external ID you can read it from this context variable. |
| Persist all context variables at start | If true all context variables will be written as input values in the table: JOB\_INSTANCE\_CONTEXT |
| Load context from job instance if (if >0) | Declare here a context variable containing a job instance id. If this ID is > 0 this job reads the context from this job instance. This provides restart capabilities to a job. |
| Return last instance result | Fetches the information about the last run of this job. All information available as return values of the tJobInstanceStart component. |
| Last successful | The last run is the last successful run of this job (all others will be ignored) |
| Last mus have data inserted or deleted | The last run must have data inserted or deleted. This will be detected via the key figures. See the properties of tJobInstanceEnd. |
| Collecting job instances ids running after previous run | Returns as comma separated list all instance ids of all job, which was running after the last run of this job. This helps to implement incremental jobs. It is necessary to write the job instance id into every data set proceed by the job. |
| Only successful | Only successful job are part of the list above |
| Only with data | Only job which affects more the one dataset will be part of the list above |
| Source job names | Filter the jobs which should part of the list above. This helps to keep the list small in case of having a lot of unrelated jobs in the system. |
| OK Result Codes | This is a String containing a comma-separated list of all return codes, which are related to a successful run. If you want using different return codes for OK please take care the tRunJob components does not die. |
| **Advanced Settings** |  |
| Schema | The schema (or database) will be retrieved from the connection object. In case of you want use a different schema or database, here is the place to say that. |
| Table for job instances | The name of the main table. This table keeps all basic information about job runs. Usually it is called JOB\_INSTANCE\_STATUS. In case of this name violates existing tables or naming conventions, here it can be changed. |
| Job instance ID is auto increment | This have to be switched on if the table use an auto increment e.g. this is supposed for MySQL. |
| Sequence expression | In case of auto increment is off, here set the name of the sequences for the job instance ID. This expression have to return a new value for the job instance ID:  Examples:  MySQL: use auto increment  Oracle: job\_instance\_id\_seq.nextval  PostgreSQL: nextval('job\_instance\_id\_seq')  DB2: NEXTVAL FOR job\_instance\_id\_seq |
| Table for job instance context | In this table the context variables will be saved. Usually it is called: JOB\_INSTANCE\_CONTEXT |
| Table for job instance counters | In this table the named counters will be stored. Usually it is called: JOB\_INSTANCE\_COUNTERS |

**Return values of tJobInstanceStart**

|  |  |
| --- | --- |
| **Return value** | **Content** |
| ERROR\_MESSAGE | Last error message. Unfortunately this is not the error message from the actually running job. This message is build from the tRunTask component. The current TAC web service does not provide this message. |
| JOB\_INSTANCE\_ID | The job instance id used for this job run. |
| SOURCE\_JOB\_INSTANCE\_ID\_LIST | List of all job instance ids which are executed after the last run if this job. This way it is possible to implement incremental steering.  The list can easily be used in SQL e.g.  ...where job\_instance\_id in (“ + ((String)globalMap.get(“tJobInstanceStart\_1\_SOURCE\_JOB\_INSTANCE\_ID\_LIST”) + “) …..” |
| JOB\_START\_DATE | The start date of the current job run. |
| PREV\_JOB\_EXISTS | If true means the job was running in the past at least one time. |
| PREV\_JOB\_START\_DATE | If a previous job run exists (otherwise null):  Contains the start date of the previous job |
| PREV\_JOB\_STOP\_DATE | If a previous job run exists (otherwise null):  Contains the stop date of the previous job |
| PREV\_JOB\_INSTANCE\_ID | If a previous job run exists (otherwise null):  Contains the ID of the previous job |
| PREV\_JOB\_TALEND\_PID | If a previous job run exists (otherwise null):  Contains the Talend-PID of the previous job |
| PREV\_JOB\_HOST\_PID | If a previous job run exists (otherwise null):  Contains the Host-PID (means the process ID of the operating system for this JVM) of the previous job |
| PREV\_JOB\_HOST\_NAME | If a previous job run exists (otherwise null):  Contains the name of the host where the previous job was running |
| PREV\_TIME\_RANGE\_START | If a previous job run exists (otherwise null):  Contains the time range start of the previous job |
| PREV\_TIME\_RANGE\_END | If a previous job run exists (otherwise null):  Contains the time range end of the previous job |
| PREV\_VALUE\_RANGE\_START | If a previous job run exists (otherwise null):  Contains the value range start of the previous job |
| PREV\_VALUE\_RANGE\_END | If a previous job run exists (otherwise null):  Contains the value range end of the previous job |
| PREV\_JOB\_RETURN\_CODE | If a previous job run exists (otherwise null):  Contains the return code of the previous job |
| PREV\_WORK\_ITEM | If a previous job run exists (otherwise null):  Contains the previous work item of the previous job |
| PREV\_RESULT\_ITEM | If a previous job run exists (otherwise null):  Contains the result item of the previous job |
| PREV\_COUNT\_INPUT | If a previous job run exists (otherwise null):  Contains the count inserts of the previous job |
| PREV\_COUNT\_OUTPUT | If a previous job run exists (otherwise null):  Contains the count outputs of the previous job |
| PREV\_COUNT\_UPDATED | If a previous job run exists (otherwise null):  Contains the count updates of the previous job |
| PREV\_COUNT\_DELETED | If a previous job run exists (otherwise null):  Contains the count deletes of the previous job |
| PREV\_COUNT\_REJECTS | If a previous job run exists (otherwise null):  Contains the count rejects of the previous job |

**Properties of tJobInstanceEnd**

|  |  |
| --- | --- |
| **Property** | **Content** |
| Use separate connection | In case of the job runs very long it make sense to use a new connection at the end of the job to update the key figures. |
| Connection | If the previous option is true you can choose a database connection for using in this component. Please take care you do not use the same connection as in the tJobInstanceStart because this does not make sense. |
| Job Instance Start Component | Choose here the tJobInstanceStart component. Both components depend on each other. |
| Job Result | A string representation of the result of the current job. In case the job creates a file it is a good idea to put here the file path. |
| Time range start | If the job has to process data selected by a time range. This could be used instead of an work item to see what work this job instance do. |
| Time range end | See Time range start. The end of the time range to proceed. |
| Value range start | If the job has to process a portion of data selected by an id range or any other value ranges. |
| Value range end | See Value range start. This is the end of the range. |
| Save named counters | Counters can be named, in this case the counter value will be inserted in the table JOB\_INSTANCE\_COUNTERS |
| Save context variables at the end of the job | This way it is possible to provide the context variables as output for other jobs which are not embedded or running in different job servers or later. It is also useful for checks about the job result. |
| Delete previous successful job instances by work item | If checked, the component deletes all successful previous job instances with the same work item. This helps in case of the table job\_instance\_status will be used to keep track of the current data in the DWH and repeated job runs with the same work item replaces previous data. |
| Close Connection | Closes the connection used for managing the job registration |
| Input Counters | Counters describing the result of the job can be added here. The sum of all counters will be written in the JOB\_INSTANCE\_STATUS table in COUNT\_INPUTS. The flag Add can be used to subtract a value instead of adding it. The name column provides the name (see Save named counters option) |
| Output Counters | See Input Counters. Will be used for column COUNT\_OUTPUTS |
| Update Counters | See Input Counters. Will be used for column COUNT\_UPDATED |
| Reject Counters | See Input Counters. Will be used for column COUNT\_REJECTED |
| Delete Counters | See Input Counters. Will be used for column COUNT\_DELETED |

As Counter typically the NB\_LINE return values of the input or output components can be used.

In case of the job has more the one output it is recommended to set names for particular counters to keep the distinct counter values.

**Return values of tJobInstanceEnd**

|  |  |
| --- | --- |
| **Return value** | **Content** |
| ERROR\_MESSAGE | Last error message. |
| RETURN\_CODE | The retrieved return code of the current job |
| RETURN\_MESSAGE | The created return message. This message contains all error messages from all components throwing an error. |

**Properties of tJobDataRangeScanner**

|  |  |
| --- | --- |
| **Property** | **Content** |
| Job Instance Start Component | Choose here the tJobInstanceStart component. Both components depend on each other. |
| Schema | This is necessary to have the schema column available. It is not supposed to change anything here |
| Configure Extraction | For every schema column you can define for which range it will be checked: Time range or Value range. The min and max values will be found even the component runs in iteration. |

**Return values of tJobDataRangeScanner**

|  |  |
| --- | --- |
| **Return value** | **Content** |
| ERROR\_MESSAGE | Last error message in case of the range detection fails for a column. |
| TIME\_RANGE\_START | The min value for the measured time range. |
| TIME\_RANGE\_END | The max value for the measured time range. |
| VALUE\_RANGE\_START | The min value for the measured value range as Long or String |
| VALUE\_RANGE\_END | The max value for the measured value range as Long or String |
| NB\_LINE\_AGGREGATED | The number or rows for this component measured over all iterations |

**Properties of tJobInstanceLiveCheck**

|  |  |
| --- | --- |
| **Property** | **Content** |
| Database Connection | Any database connection pointing to the schema with the control tables.  The main table is JOB\_INSTANCES holding all key information. |
| Close Connection | If true the connection will be closed at the end of the component processing |
| Schema | This component provides a input flow providing information about cleaned job instances |
| Last system start | If the last system start could be determined (currently there is not platform independent implementation to get this information automatically) all older job instance starts will be cleaned. |

**Return values of tJobInstanceLiveCheck**

|  |  |
| --- | --- |
| **Return value** | **Content** |
| ERROR\_MESSAGE | Error message if something in the processing of the component it self went wrong |
| COUNT\_RUNNING\_PROCESSES | The number of all running processes on the current server (regardless if this is a Talend job or not) |
| COUNT\_RUNNING\_JOB\_INSTANCES | The number of as running declared job instances |
| COUNT\_BROKEN\_JOB\_INSTANCES | The number of recognized broken job instances |
| NB\_LINE | Number of rows in the data input flow |

The schema is fully commented and provides the values of the JOB\_INSTANCE\_STATUS table for the broken instances.

**Scenario 1: Simple Job monitoring**



The typical usage is to use tPrejob component to trigger the tJobInstanceStart component and the tPostjob component to trigger tJobInstanceEnd component.

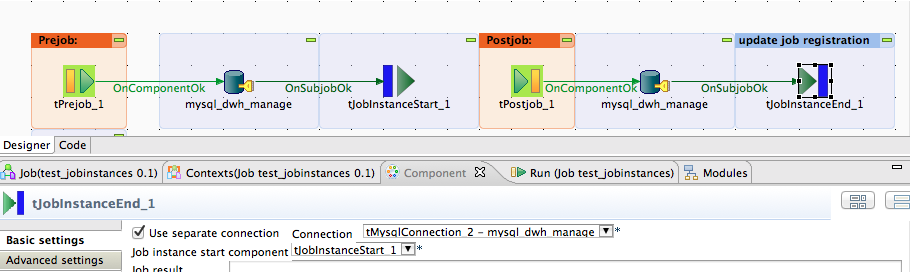
**Scenario 2: Measure the time ranges and/or value ranges**



In this Scenario the flow will be scanned for the start and end values for a time range and the value range. These values could be used to ensure the job quality or to start the next run from the previous end.

**Scenario 3: A long running job has to be monitored.**

In this case the tJobInstanceEnd component needs its own connection.



**Scenario for tJobInstanceLiveCheck**



This example job shows the main purpose of the component. Such kind of job has to run frequently on every job server (servers on which the jobs run).

The component set for broken job instances the return code 999 and as return message “Process died”.

This information can be used to clean up all depending data structures.

**Log4J Integration**

The component contains a full-featured Log4J.

The component can initialize Log4J with a default configuration or by loading a configuration file.

A default logger called “talend” will be added to the logger hierarchy.

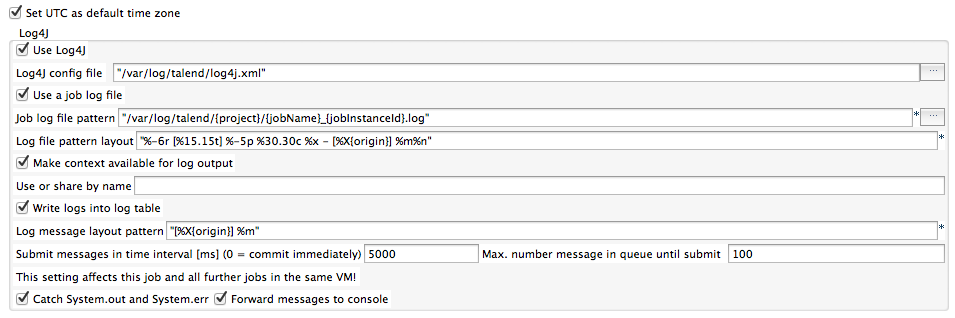
For every job a logger will be added with the name pattern: talend.<Project>.<Job Name>

For every instance of a job an appender will be added (and removed at the end of the job).

Each appender is an extended FileAppender and transports only log events from its own job by filtering the events by the Talend-PID.

If the option “Write logs into log table” is switch on, for every job a second appender will be added (and removed) which sends the messages to the job\_instance\_log table.

For the file output and the output to the table there a dedicated log formats.



The component add to every event the context variables and all default information:

These additional values will be set as MDC key-value-pairs

MDC values can be inserted in message pattern with the expression: %X{<key>}

In file names (log file names) the expression is simply: {<key>}

|  |  |
| --- | --- |
| Variable | Log message pattern (key) |
| Job name | jobName |
| Project | Project |
| Context | context |
| Job Instance ID | jobInstanceId |
| Talend job instance identifier | talendPid |
| Talend parent job instance identifier | talendFatherPid |
| Talend root job instance identifier | talendRootPid |
| Component which causes the message | Origin |
| Work item | workItem |
| tWarn or tDie priority | Priority |
| tWarn or tDie error code | Code |
| tWarn or tDie message type | type |
| Job version | version |
| Context variables | context.<variable> |
| Timestamp of the job start in long format (yyyy-MM-dd HH:mm:ss.SSS) | jobStartTimestampLong |
| Timestamp of the job start in compact format (yyyyMMdd\_HHmmss.SSS) | jobStartTimestampCompact |
| Job start date in long format (yyyy-MM-dd) | jobStartDateLong |
| Job start date in compact format (yyyyMMdd) | jobStartDateCompact |

**Create table scripts for the tables:**

In case of MySQL it is recommended using a serial data type for the column job\_instance\_status.job\_instance\_id.

(Assuming the tables is located in the schema dwh\_manage)

In the advanced settings of the tJobInstanceStart component it is possible to declare the schema and the table names. The option Job Instance ID is auto increment allows the usage of auto increment column for job\_instance\_id in the table job\_instance\_status.

**MySQL:**

**CREATE TABLE** JOB\_INSTANCE\_STATUS (

JOB\_INSTANCE\_ID BIGINT(20) **UNSIGNED NOT NULL AUTO\_INCREMENT**,

PROCESS\_INSTANCE\_ID BIGINT(20),

PROCESS\_INSTANCE\_NAME VARCHAR(255),

JOB\_NAME VARCHAR(255) **NOT NULL**,

JOB\_PROJECT varchar(128),

JOB\_DISPLAY\_NAME VARCHAR(255),

JOB\_GUID VARCHAR(100) **NOT NULL**,

JOB\_EXT\_ID VARCHAR(255),

JOB\_INFO VARCHAR(255),

ROOT\_JOB\_GUID VARCHAR(100),

WORK\_ITEM VARCHAR(1024),

TIME\_RANGE\_START TIMESTAMP,

TIME\_RANGE\_END TIMESTAMP,

VALUE\_RANGE\_START VARCHAR(512),

VALUE\_RANGE\_END VARCHAR(512),

JOB\_STARTED\_AT TIMESTAMP,

JOB\_ENDED\_AT TIMESTAMP,

JOB\_RESULT VARCHAR(1024),

COUNT\_INPUT INT,

COUNT\_OUTPUT INT,

COUNT\_UPDATED INT,

COUNT\_REJECTED INT,

COUNT\_DELETED INT,

RETURN\_CODE INT,

RETURN\_MESSAGE TEXT,

HOST\_NAME VARCHAR(255) ,

HOST\_PID INT,

HOST\_USER VARCHAR(128) ,

**PRIMARY KEY** (JOB\_INSTANCE\_ID)

) **DEFAULT CHARSET=UTF8**;

**CREATE INDEX** JOB\_INSTANCE\_STATUS\_JOB\_GUID ON JOB\_INSTANCES(JOB\_GUID);

**CREATE TABLE** JOB\_INSTANCE\_CONTEXT (

JOB\_INSTANCE\_ID BIGINT **NOT NULL**, -- reference to the job instance

ATTRIBUTE\_KEY VARCHAR(100) **NOT NULL**, -- context variable name

ATTRIBUTE\_VALUE VARCHAR(1024), -- textual representation of the value

ATTRIBUTE\_TYPE VARCHAR(32) **NOT NULL**, -- Java class name of the value

IS\_OUTPUT\_ATTR BOOLEAN **NOT NULL**); -- 0 = Input, 1 = Output

**CREATE INDEX** JOB\_INSTANCE\_CONTEXT\_IDX **ON** JOB\_INSTANCE\_CONTEXT(JOB\_INSTANCE\_ID, ATTRIBUTE\_KEY, IS\_OUTPUT\_ATTR);

**CREATE TABLE** JOB\_INSTANCE\_COUNTERS (

JOB\_INSTANCE\_ID BIGINT NOT NULL, -- reference to the job instance

COUNTER\_NAME VARCHAR(128) NOT NULL, -- name of the counter set in tJobInstanceEnd for a counter

COUNTER\_VALUE INTEGER, -- value of the counter

**CONSTRAINT** PK\_JOB\_INSTANCE\_COUNTERS **PRIMARY KEY** (JOB\_INSTANCE\_ID, COUNTER\_NAME));

**CREATE TABLE** JOB\_INSTANCE\_LOGS (

JOB\_INSTANCE\_ID BIGINT **NOT NULL**,

LOG\_TS TIMESTAMP **NOT NULL**,

LOG\_LEVEL VARCHAR(10),

LOG\_NAME VARCHAR(128) **NOT NULL**,

LOG\_MESSAGE TEXT);

**CREATE INDEX** JOB\_INSTANCE\_LOGS\_JOBID **ON** JOB\_INSTANCE\_LOGS(JOB\_INSTANCE\_ID);

**PostgreSQL**:

create table dwh\_manage.job\_instance\_status (

job\_instance\_id bigint not null,

process\_instance\_id integer,

process\_instance\_name varchar(255),

job\_name varchar(255) not null,

job\_project varchar(128),

job\_info varchar(512),

job\_display\_name varchar(255),

job\_guid varchar(100) not null,

job\_ext\_id varchar(255),

root\_job\_guid varchar(100),

work\_item varchar(1024),

time\_range\_start timestamp,

time\_range\_end timestamp,

value\_range\_start varchar(512),

value\_range\_end varchar(512),

job\_started\_at timestamp not null,

job\_ended\_at timestamp,

job\_result varchar(1024),

count\_input integer,

count\_output integer,

count\_updated integer,

count\_rejected integer,

count\_deleted integer,

return\_code integer,

return\_message varchar(1024),

host\_name varchar(255),

host\_pid integer,

host\_user varchar(128),

constraint job\_instances\_pkey primary key (job\_instance\_id));

create index job\_instances\_job\_guid on dwh\_manage.job\_instance\_status(job\_guid);

create sequence dwh\_manage.job\_instance\_id\_seq start with 1;

create table dwh\_manage.job\_instance\_context (

job\_instance\_id bigint not null,

attribute\_key varchar(255) not null,

attribute\_value varchar(1024),

attribute\_type varchar(32) not null,

is\_output\_attr boolean not null);

create index job\_instances\_context\_idx on dwh\_manage.job\_instance\_context(job\_instance\_id, is\_output\_attr, attribute\_key);

create table dwh\_manage.job\_instance\_counters (

job\_instance\_id bigint not null,

counter\_name varchar(128) not null,

counter\_value integer not null);

create index job\_instance\_counters\_idx on dwh\_manage.job\_instance\_counters(job\_instance\_id, counter\_name);

create table dwh\_manage.job\_instance\_logs (

job\_instance\_id bigint not null,

log\_ts timestamp not null,

log\_name varchar(128) not null,

log\_level varchar(128) not null,

log\_message text);

create index job\_instance\_logs\_jobid on dwh\_manage.job\_instance\_logs(job\_instance\_id);

**Oracle**:

CREATE TABLE JOB\_INSTANCE\_STATUS (

JOB\_INSTANCE\_ID NUMBER(16) NOT NULL,

PROCESS\_INSTANCE\_ID INTEGER,

PROCESS\_INSTANCE\_NAME VARCHAR2(255),

JOB\_NAME VARCHAR2(255) NOT NULL,

JOB\_PROJECT VARCHAR2(128),

JOB\_INFO VARCHAR2(512),

JOB\_DISPLAY\_NAME VARCHAR2(255),

JOB\_GUID VARCHAR2(100) NOT NULL,

JOB\_EXT\_ID VARCHAR2(255),

ROOT\_JOB\_GUID VARCHAR2(100),

WORK\_ITEM VARCHAR2(1024),

TIME\_RANGE\_START DATE,

TIME\_RANGE\_END DATE,

VALUE\_RANGE\_START VARCHAR2(512),

VALUE\_RANGE\_END VARCHAR2(512),

JOB\_STARTED\_AT DATE NOT NULL,

JOB\_ENDED\_AT DATE,

JOB\_RESULT VARCHAR2(1024),

COUNT\_INPUT INTEGER,

COUNT\_OUTPUT INTEGER,

COUNT\_UPDATED INTEGER,

COUNT\_REJECTED INTEGER,

COUNT\_DELETED INTEGER,

RETURN\_CODE INTEGER,

RETURN\_MESSAGE VARCHAR2(1024),

HOST\_NAME VARCHAR2(255),

HOST\_PID INTEGER,

HOST\_USER VARCHAR(128),

CONSTRAINT JOB\_INSTANCES\_PKEY PRIMARY KEY (JOB\_INSTANCE\_ID));

CREATE INDEX JOB\_INSTANCES\_JOB\_GUID ON JOB\_INSTANCE\_STATUS(JOB\_GUID);

CREATE SEQUENCE JOB\_INSTANCE\_ID\_SEQ START WITH 1;

CREATE TABLE JOB\_INSTANCE\_CONTEXT (

JOB\_INSTANCE\_ID NUMBER(16) NOT NULL,

ATTRIBUTE\_KEY VARCHAR2(255) NOT NULL,

ATTRIBUTE\_VALUE VARCHAR2(1024),

ATTRIBUTE\_TYPE VARCHAR2(32) NOT NULL,

IS\_OUTPUT\_ATTR NUMBER(1) NOT NULL);

CREATE INDEX JOB\_INSTANCES\_CONTEXT\_IDX ON JOB\_INSTANCE\_CONTEXT(JOB\_INSTANCE\_ID, IS\_OUTPUT\_ATTR, ATTRIBUTE\_KEY);

CREATE TABLE JOB\_INSTANCE\_COUNTERS (

JOB\_INSTANCE\_ID NUMBER(16) NOT NULL,

COUNTER\_NAME VARCHAR2(128) NOT NULL,

COUNTER\_VALUE INTEGER NOT NULL);

CREATE INDEX JOB\_INSTANCE\_COUNTERS\_IDX ON JOB\_INSTANCE\_COUNTERS(JOB\_INSTANCE\_ID, COUNTER\_NAME);

-- this table will be written from the Log4J appender in tJobInstanceStart

CREATE TABLE JOB\_INSTANCE\_LOGS (

JOB\_INSTANCE\_ID NUMBER(16) NOT NULL,

LOG\_TS DATE NOT NULL,

LOG\_LEVEL VARCHAR2(10) NOT NULL, -- INFO, DEBUG, WARN, ERROR

LOG\_NAME VARCHAR2(128) NOT NULL,

LOG\_MESSAGE CLOB);

CREATE INDEX JOB\_INSTANCE\_LOGS\_JOBID ON JOB\_INSTANCE\_LOGS(JOB\_INSTANCE\_ID);

**IBM DB2:**

--drop table dwh\_manage.job\_instances;

create table dwh\_manage.job\_instance\_status (

job\_instance\_id bigint not null,

process\_instance\_id integer,

process\_instance\_name varchar(255),

job\_name varchar(255) not null,

job\_project varchar(128),

job\_info varchar(512),

job\_display\_name varchar(255),

job\_guid varchar(100) not null,

job\_ext\_id varchar(255),

root\_job\_guid varchar(100),

work\_item varchar(1024),

time\_range\_start timestamp,

time\_range\_end timestamp,

value\_range\_start varchar(512),

value\_range\_end varchar(512),

job\_started\_at timestamp not null,

job\_ended\_at timestamp,

job\_result varchar(1024),

count\_input integer,

count\_output integer,

count\_updated integer,

count\_rejected integer,

count\_deleted integer,

return\_code integer,

return\_message varchar(1024),

host\_name varchar(255),

host\_pid integer,

host\_user varchar(128),

constraint job\_instances\_pkey primary key (job\_instance\_id));

create index job\_instances\_job\_guid on dwh\_manage.job\_instance\_status(job\_guid);

create sequence dwh\_manage.job\_instance\_id\_seq start with 1;

create table dwh\_manage.job\_instance\_context (

job\_instance\_id bigint not null,

attribute\_key varchar(255) not null,

attribute\_value varchar(1024),

attribute\_type varchar(32) not null,

is\_output\_attr smallint not null);

create index job\_instances\_context\_idx on dwh\_manage.job\_instance\_context(job\_instance\_id, is\_output\_attr, attribute\_key);

--drop table dwh\_manage.job\_instance\_counters;

create table dwh\_manage.job\_instance\_counters (

job\_instance\_id bigint not null,

counter\_name varchar(128) not null,

counter\_value integer not null);

create index job\_instance\_counters\_idx on dwh\_manage.job\_instance\_counters(job\_instance\_id, counter\_name);

--drop table dwh\_manage.job\_instance\_logs;

create table dwh\_manage.job\_instance\_logs (

job\_instance\_id bigint not null,

log\_ts timestamp not null,

log\_level varchar(10), -- INFO, WARN, ERROR, DEBUG, TRACE

log\_name varchar(128) not null,

log\_message clob);

create index job\_instance\_logs\_jobid on dwh\_manage.job\_instance\_logs(job\_instance\_id);

**Exasol:**

create table job\_instance\_status (

job\_instance\_id bigint identity primary key,

process\_instance\_id integer,

process\_instance\_name varchar(255),

job\_name varchar(255) not null,

job\_info varchar(512) UTF8,

job\_display\_name varchar(255) UTF8,

job\_guid varchar(100) UTF8 not null,

job\_ext\_id varchar(255) UTF8,

root\_job\_guid varchar(100) UTF8,

work\_item varchar(1024) UTF8,

time\_range\_start timestamp,

time\_range\_end timestamp,

value\_range\_start varchar(512) UTF8,

value\_range\_end varchar(512) UTF8,

job\_started\_at timestamp not null,

job\_ended\_at timestamp,

job\_result varchar(1024) UTF8,

count\_input integer,

count\_output integer,

count\_updated integer,

count\_rejected integer,

count\_deleted integer,

return\_code integer,

return\_message varchar(4000) UTF8,

host\_name varchar(255) UTF8,

host\_pid integer,

host\_user varchar(128) UTF8);

create table job\_instance\_context (

job\_instance\_id bigint not null,

attribute\_key varchar(255) UTF8 not null,

attribute\_value varchar(1024) UTF8,

attribute\_type varchar(32) UTF8 not null,

is\_output\_attr boolean not null);

create table job\_instance\_counters (

job\_instance\_id bigint not null,

counter\_name varchar(128) not null,

counter\_value integer not null);

create table job\_instance\_logs (

job\_instance\_id bigint not null,

log\_ts timestamp not null,

log\_name varchar(128) not null,

log\_level varchar(128) not null,

log\_message varchar(10000));