



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 | Deregisters the job run, collects the metrics and cleans up the logging settings for this run. |
| tJobDataRangeScanner | Collects min/max values of timestamps or numeric values within a data flows. These min(max values for timestamps and any other values are set for the job-time-range and job-value-range. |
| tJobInstanceLiveCheck | Checks the entries of the job run registry for dead or broken job instances and cleans 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
- Tracks the memory usage and detects peaks
- Detects parallel job runs (also based on the work item) and provide steering possibilities

Talend-Integration

This component can be found in the palette under Management

This component provides several return values.

Component tJobInstanceStart

Basic Settings

| Property | Content |
|-----------------------------------|--|
| Use data source (connection pool) | If true the component takes the database connection from a database connection pool. |
| Data source alias | The name of the database connection pool providing the database connection. |
| Database Connection | Any database connection pointing to the schema with the control tables. The main table is JOB_INSTANCES holding all key information. |
| Job Name | Name of the job. The default is using the build-in variable jobName |
| Task name | A job can be used with different parameters for different purposes called task (just like the methodology in the TAC). If the job runs only for one purpose keep the job name here otherwise extend the job name with "-" or "#" and add a name extension to declare this way the task name. This information will be stored in the field job_display_name for |

| | |
|---|---|
| | compatibility reasons. |
| 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) |
| Take empty as null | If the given value is empty, it will be taken as null |
| 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 id 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 >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. |
| Singleton Behavior | See the section Singleton Behavior later in this document. |
| 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 run must have inserted or deleted data | The last run must have data inserted or deleted. This will be detected via the key figures. See the properties of tJobInstanceEnd. |
| Last run must have read data | The last run must have read data to be relevant as choice for the last run. |
| For the current work item | If true: only jobs with the same work item will be used to get the last instance results. This allows you to have one job for multiple purposes with the full support of the "Return last instance result" feature. |
| 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. |
| 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. |
| Replacement for prev. job information | These values can be used to simplify the job design because you can avoid the check if the job has a previous run. Please refer to the return values. |
| job-instance-id | Set here a default long value for the return value PREV_JOB_INSTANCE_ID |
| job-start-date | Set here the Date typed default value for a previous job start date |
| time-range-end | Set here the Date typed default value for a previous time range end. |
| value-range-end | Set here the String typed default value for a previous value range end. |
| result-item | Set here the String typed default value for a previous job result item. |
| Set UTC as default time zone | This changes the default setting of the virtual machine for time zone from the local time zone to UTC. It affects the current JVM instance (means all job called by tRunJob and not as independent child job). |

| | |
|--------------------------------------|--|
| Memory Usage Monitoring | This starts a thread, which collects every second, the used memory and detect the maximum and when it happened. In the tJobInstanceEnd component return values you can get these values. |
| Print job instance id to the console | If true the job instance id will be printed to System.out |
| Expression to print | Define here how to print the job instance id. Actually it is not limited to the job instance id, you can use all possible return values here. |

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. In former releases this table had the default name JOB_INSTANCES. Starting with this release there will be no table renamed anymore because of the wide usage of this component. |
| 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. |
| Read from Generated Keys.... | If true, the component avoids to re select the job instance id instead it uses the jdbc driver feature to deliver the generated keys. Sometimes this does not work, and in this case deactivate this option. |
| Sequence expression | In case of auto increment is off, here set the name of the sequences for the job instance ID. This expression has to return a new value for the job instance ID: Examples: MySQL: use auto increment Oracle: job_instance_id_seq.nextval PostgreSQL: nextval('dwh_manage.job_instance_id_seq') DB2: NEXTVAL FOR dwh_manage.job_instance_id_seq |

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 |

Singleton Behavior

This component has the capability based in the information of the `JOB_INSTANCE_STATUS` table to detect already running instances of the same job (optional with the same work item).

Here the necessary basic settings to use this feature:

| Property | Content |
|---|--|
| Check if another job instance of the same job is already running | If true the component checks the <code>JOB_INSTANCE_STATUS</code> table for a already running job instance |
| Singleton for the work item | Use the work item to identify a job instance. |
| Prevent creating a job instance status entry if another job instance is already running | The option helps to avoid useless entries in the <code>JOB_INSTANCE_STATUS</code> table. |

Dedicated return values

| Return value | Content |
|--|--|
| <code>JOB_RUNS_ALONE</code> | The result of the singleton-check. This variable is not set if the check has not happened, this should help to prevent using this value if the actual check was not performed. |
| <code>ALREADY_RUNNING_JOB_START_DATE</code> | Start date (+time) of the already running job |
| <code>ALREADY_RUNNING_JOB_INSTANCE_ID</code> | The job instance id of the other job still running |
| <code>ALREADY_RUNNING_JOB_HOST_NAME</code> | On which server the other job instance is still running |
| <code>ALREADY_RUNNING_JOB_HOST_PID</code> | The native process id of the other job instance |
| <code>ALREADY_RUNNING_JOB_WORK_ITEM</code> | What work item the other job is still processing |

Component tJobInstanceEnd

Basic Settings

| Property | Content |
|---|--|
| Use separate connection | If true: the end component uses a separate database connection. This could help for jobs running very long to avoid problems with long standing connections. |
| Connection | Choose here the separate connection if you have chosen the option above. It must be a different connection component as used for the start component. |
| 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 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 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. |
| MEMORY_AVAILABLE | This is the memory (in byte) what is maximum available for the job. Typically it is set with the JVM parameter <code>-Xmx1024m</code> (e.g. for 1GB RAM) |
| MEMORY_MAX_USED | The maximum of the used memory (in byte) what was allocated in the JVM. Please keep in mind, if you call other jobs with tRunJob (not independently) they must be taken into account because they use the same JVM instance. |
| MEMORY_MAX_USED_PERCENTAGE | The percentage between the available memory and the maximum used memory as value between 0 and 100. |

Component tJobDataRangeScanner

This component measures the min/max values from any values (like IDs) or timestamps (like last_modified). These min/max values will be used to set the column values in the table job_instance_status (value_range_start, value_range_end, time_range_start, time_range_end).

All these values can be retrieved for the last run as return values of the tJobInstanceStart component.

Please refer to the return values of the tJobInstanceStart component.

Basic settings

| 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

| 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 of rows for this component measured over all iterations |

Component tJobInstanceLiveCheck

This component checks if jobs still alive. To do this, you have to build a very simple job (checkout the scenarios) and let them run on every job server you have.

Basic settings

| 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 | If true the connection will be closed at the end of the component processing |
| Schema | This component provides an 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. |

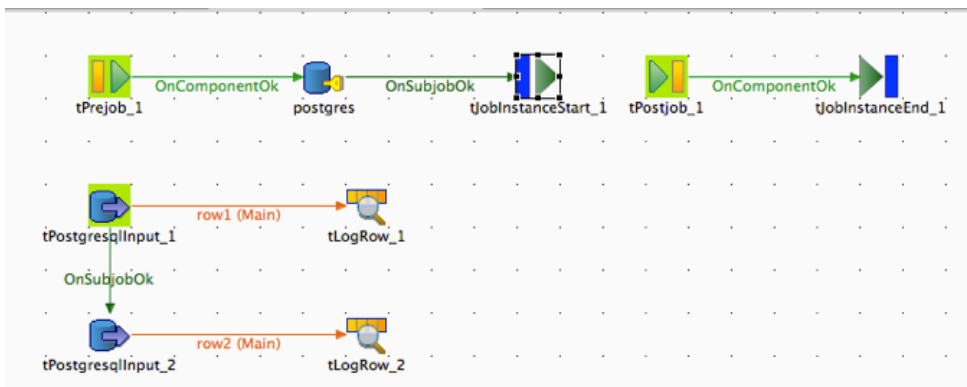
Return values

| 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.

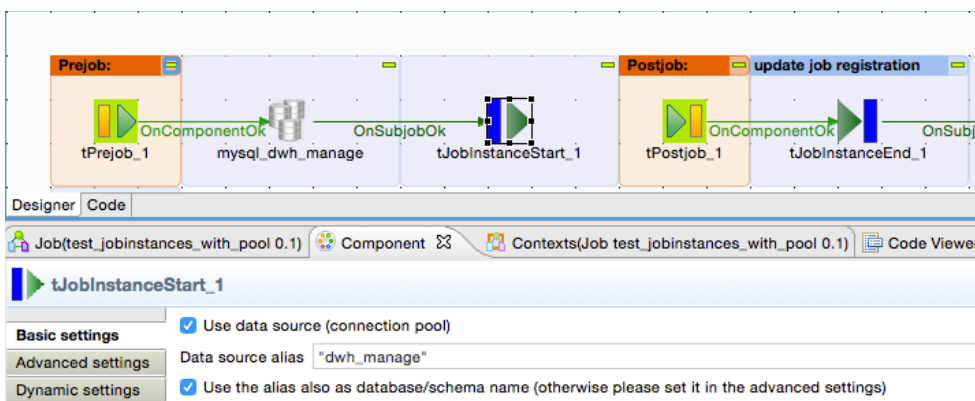
Scenarios

Scenario 1: Simple Job monitoring



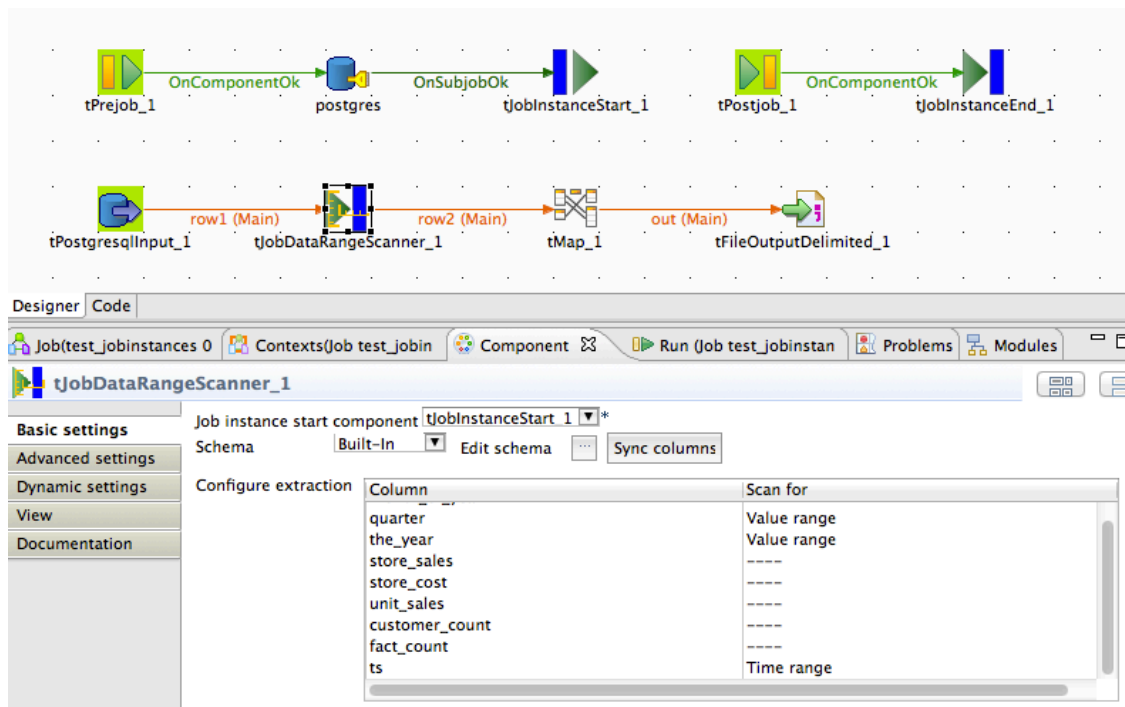
The typical usage is to use **tPrejob** component to trigger the **tJobInstanceStart** component and the **tPostjob** component to trigger **tJobInstanceEnd** component.

Scenario 2: Using a connection pool



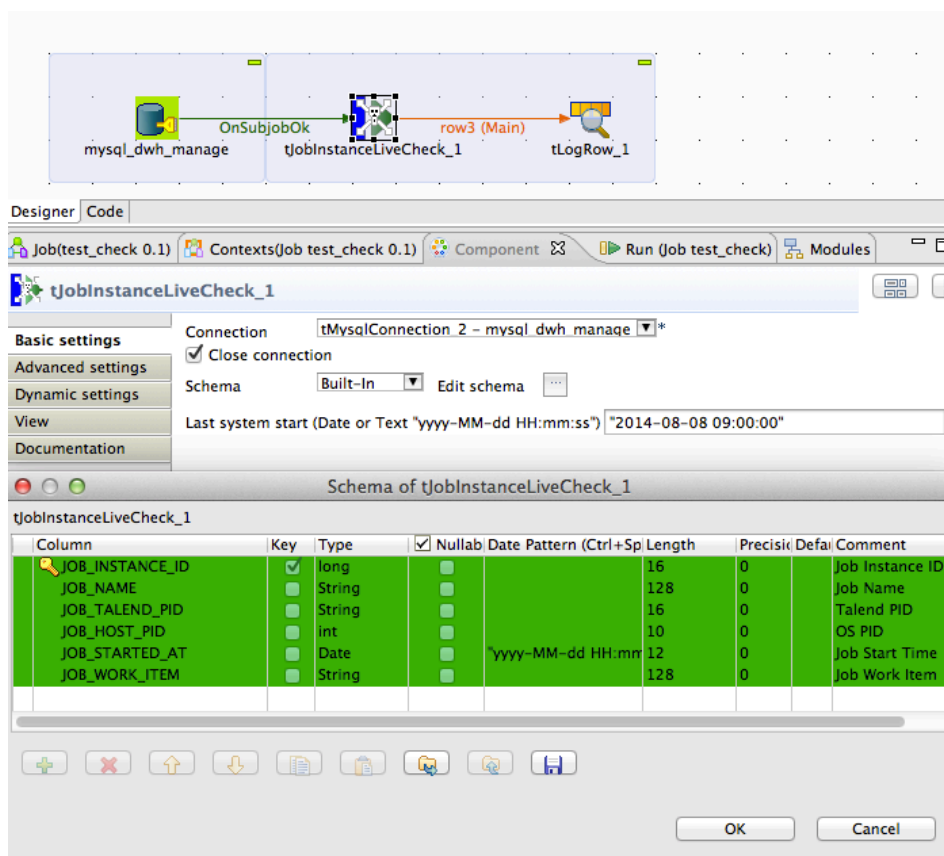
The connection pool can be established anywhere unless it is before the initialization of the **tJobInstanceStart** component. Also child jobs can use the same connection pool. It is a good practice to name the pool like the addressed database. This way the configuration more convenient.

Scenario 3: 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 for tJobInstanceLiveCheck

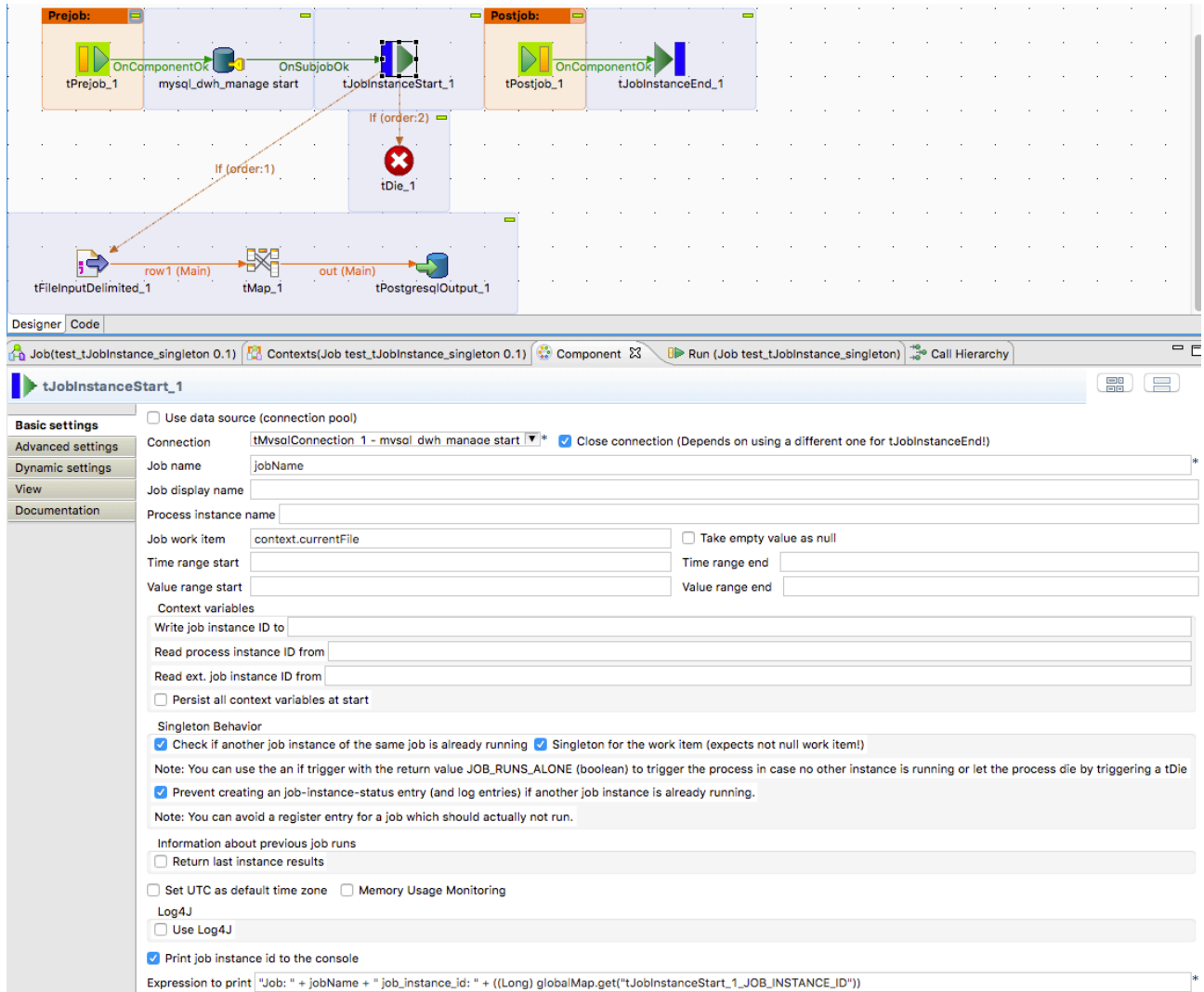


This example job shows the main purpose of the component. Such kind of job has to run frequently on every job server (servers on which the jobs run).

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.

Scenario 4: Using the singleton check

The check depends in the information of the job_instance_status table. If you decide to use this check it is highly recommended to establish also a live check for job instances (use tJobInstanceLiveCheck component).



The if-trigger contains the return variable JOB_RUNS_ALONE to trigger the actual job or in this case to let the job die.

Checklist to use this component suite

1. Use a OLTP database for the tables used by this component. Column oriented database are mostly too slow for the possible high frequency insert/updates from a large number of simultaneous running jobs! Typical bad choices: Teradata, Infobright engine of MySQL... Typical capable databases: Oracle, MySQL, DB2, PostgreSQL, H2
2. Build a separate database schema (e.g. call it `dwh_manage` or `dwh_meta` or what you want ;-)
3. Think about the primary key in the `JOB_INSTANCE_STATUS` table. You can use a database sequence (this is the preferred way) or you can use a self-incrementing data type (e.g. serial or identity column types). If you use a sequence, please set the SQL code to get the next value in the advanced settings of the `tJobInstanceStart` component. By the way, in large projects it could be helpful to set these settings per default directly in the component in the `tJobInstanceStart_java.xml` file of the component.
4. If you use the component, please check if the job is long running job and, in this case, use a separate connection for the `tJobInstanceEnd` component or consider the usage of a connection pool to avoid problems with server-side disconnected database connections.

Log4J Integration

The Log4J integration was removed in version 8 of the components because of incompatibility to Talend 7 Runtime and the fact, Talend 7 has replaced the Log4J v1 framework in favor of Log4J v2 but we have a lot of Log4J v1 installations. The internal logging will be supported by the SL4J framework.

Descriptions of the tables used

Table: job_instance_status

| column name | column type | description |
|-----------------------|---------------|--|
| job_instance_id | bigint | ID of the entry. This entry will be generated by an auto increment column or a sequence. You have to configure in the tJobInstanceStart component which way you have choosen. |
| process_instance_id | integer | The job_instance_id of the root process. It is empty for jobs which does not have a parent or root |
| process_instance_name | varchar(255) | You can provide a name for the job instance – only for information in reporting tools |
| job_name | varchar(255) | The actual job name. |
| job_project | varchar(128) | The project in which the job is developed |
| job_info | varchar(512) | A summary information about the job including version and context under which the current job runs |
| job_display_name | varchar(255) | A human readable name for the job |
| job_guid | varchar(100) | A unique identifier for the job (we uses normally the Talend-PID but the component is aware of the fact, this ID is by far not really unique) |
| job_ext_id | varchar(255) | Ther possibility to provide an external guid for the job. Especcially of you want to trigger the job from another tool and this other tool wants to check the status of the job. This ID can help to identify the entry. |
| root_job_guid | varchar(100) | A unique identifier for the root job (we uses normally the Talend-PID but the component is aware of the fact, this ID is by far not really unique) |
| work_item | varchar(1024) | In case you work with partitioned data, here you will find the name of the data partition the job has as task. |
| time_range_start | timestamp | In case of incremental loads or partitioned data with date ranges. This field contains the start of the range. |
| time_range_end | timestamp | In case of incremental loads or partitioned data with date ranges. This field contains the end of the range. |
| value_range_start | varchar(512) | In case of incremental loads or partitioned data with value ranges. e.g. the IDs from 1000 to 1999. This field contains the start of the range. |
| value_range_end | varchar(512) | In case of incremental loads or partitioned data with value ranges. e.g. the IDs from 1000 to 1999. This field contains the end of the range. |
| job_started_at | timestamp | Timestamp when the start has been started. This is the timestamp measured within the job when the job really has been started. |
| job_ended_at | timestamp | Timestamp when the start has been stopped. This is the timestamp measured within the job when the job really has been stopped. |
| job_result | varchar(1024) | A field which can contains a result like a created file path or similar. |
| count_input | integer | The sum of all input counters definined in tJobInstanceEnd |
| count_output | integer | The sum of all output counters definined in tJobInstanceEnd |
| count_updated | integer | The sum of all update counters definined in tJobInstanceEnd |
| count_rejected | integer | The sum of all reject counters definined in tJobInstanceEnd |
| count_deleted | integer | The sum of all delete counters definined in tJobInstanceEnd |
| return_code | integer | The exit code of the job. |
| return_message | varchar(1024) | All error messages available on the job. It contains all messages from tDie components. |
| host_name | varchar(255) | The host name of the server where this job runs |
| host_pid | integer | The PID from the operating system for this job. This helps to identify the job in a process list. |
| host_user | varchar(128) | The user in the operating system under which the job runs. |

Table: job_instance_counters

| column name | column type | description |
|-----------------|--------------|--|
| job_instance_id | bigint | Related job_instance_id |
| counter_name | varchar(128) | Name of the counter. Set this name in the tJobInstanceEnd as name for an entry in the counter table. |
| counter_type | Varchar(20) | Type of the counter (input, output,...) |
| counter_value | integer | The value of the counter |

Create table scripts for the tables

Not all databases are capable to work for this use case. Generally, all OLTP databases work fine. It could be problematic to host these tables on a column-based database or a database with distributed storage like Teradata. Such database tends to be very slow for single inserts and updates or they lock the whole table while such operations and this could lead to significant performance problems!

The well-tested databases are MySQL (MyISAM or INNODB), Oracle, PostgreSQL, H2 and IBM DB2

In case of MySQL it is recommended using a serial data type for the column JOB_INSTANCE_STATUS(JOB_INSTANCE_ID).

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. In former releases some tables had slightly different names, but the meaning and structure is the same. You can adapt old names in the configuration if the tJobInstanceStart advanced settings.

MySQL (use auto increment)

```
CREATE TABLE JOB_INSTANCE_STATUS (
  JOB_INSTANCE_ID BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  PROCESS_INSTANCE_ID BIGINT DEFAULT NULL,
  PROCESS_INSTANCE_NAME VARCHAR(255) DEFAULT NULL,
  JOB_NAME VARCHAR(255) NOT NULL,
  JOB_PROJECT varchar(128),
  JOB_DISPLAY_NAME VARCHAR(255) DEFAULT NULL,
  JOB_GUID VARCHAR(100) NOT NULL,
  JOB_EXT_ID VARCHAR(255) DEFAULT NULL,
  JOB_INFO VARCHAR(255) DEFAULT NULL,
  ROOT_JOB_GUID VARCHAR(100) DEFAULT NULL,
  WORK_ITEM VARCHAR(1024) DEFAULT NULL,
  TIME_RANGE_START TIMESTAMP(3) NULL DEFAULT NULL,
  TIME_RANGE_END TIMESTAMP(3) NULL DEFAULT NULL,
  VALUE_RANGE_START VARCHAR(512) DEFAULT NULL,
  VALUE_RANGE_END VARCHAR(512) DEFAULT NULL,
  JOB_STARTED_AT TIMESTAMP(3) NULL DEFAULT NULL,
  JOB_ENDED_AT TIMESTAMP(3) NULL DEFAULT NULL,
  JOB_RESULT VARCHAR(1024) DEFAULT NULL,
  COUNT_INPUT INT(11) DEFAULT NULL,
  COUNT_OUTPUT INT(11) DEFAULT NULL,
  COUNT_UPDATED INT(11) DEFAULT NULL,
  COUNT_REJECTED INT(11) DEFAULT NULL,
  COUNT_DELETED INT(11) DEFAULT NULL,
  RETURN_CODE INT(11) DEFAULT NULL,
  RETURN_MESSAGE TEXT,
  HOST_NAME VARCHAR(255) DEFAULT NULL,
  HOST_PID INT(11) DEFAULT NULL,
  HOST_USER VARCHAR(128) DEFAULT NULL,
  PRIMARY KEY (JOB_INSTANCE_ID)
) DEFAULT CHARSET=UTF8;

CREATE INDEX JOB_INSTANCE_STATUS_JOB_GUID ON JOB_INSTANCE_STATUS (JOB_GUID);
CREATE INDEX JOB_INSTANCE_STATUS_JOB_NAME ON JOB_INSTANCE_STATUS (JOB_NAME);
```

```

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_TYPE VARCHAR(20),            -- type of the counter
    COUNTER_VALUE INTEGER,               -- value of the counter
    CONSTRAINT PK_JOB_INSTANCE_COUNTERS PRIMARY KEY (JOB_INSTANCE_ID, COUNTER_NAME));

```

PostgreSQL (uses a sequence)

```

--drop table dwh_manage.job_instances;
create table dwh_manage.job_instance_status (
    job_instance_id bigint not null,
    process_instance_id bigint,
    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 index job_instances_job_name on dwh_manage.job_instance_status(job_name);

--drop sequence dwh_manage.job_instance_id;
create sequence dwh_manage.seq_job_instance_id start with 1;

--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_type varchar(20),
    counter_value integer not null);

create index job_instance_counters_idx on dwh_manage.job_instance_counters(job_instance_id,
counter_name);

```

Oracle (uses a sequence)

```

-- drop table JOB_INSTANCE_STATUS;
CREATE TABLE JOB_INSTANCE_STATUS (
    JOB_INSTANCE_ID NUMBER(19) NOT NULL,
    PROCESS_INSTANCE_ID NUMBER(19),
    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 TIMESTAMP,
    TIME_RANGE_END TIMESTAMP,

```



```

VALUE_RANGE_START VARCHAR2(512),
VALUE_RANGE_END VARCHAR2(512),
JOB_STARTED_AT TIMESTAMP NOT NULL,
JOB_ENDED_AT TIMESTAMP,
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 INDEX JOB_INSTANCES_JOB_NAME ON JOB_INSTANCE_STATUS(JOB_NAME);

CREATE SEQUENCE SEQ_JOB_INSTANCE_ID START WITH 1;

CREATE TABLE JOB_INSTANCE_COUNTERS (
    JOB_INSTANCE_ID NUMBER(19) NOT NULL,
    COUNTER_NAME VARCHAR2(128) NOT NULL,
    COUNTER_TYPE VARCHAR2(20),
    COUNTER_VALUE INTEGER NOT NULL);

CREATE INDEX JOB_INSTANCE_COUNTERS_IDX ON JOB_INSTANCE_COUNTERS(JOB_INSTANCE_ID, COUNTER_NAME);

```

IBM DB2

```

--drop table dwh_manage.job_instance_status;
create table dwh_manage.job_instance_status (
    job_instance_id bigint not null,
    process_instance_id bigint,
    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 dwh_manage.job_instances_job_guid on dwh_manage.job_instance_status(job_guid);
create index dwh_manage.job_instances_job_name on dwh_manage.job_instance_status(job_name);

--drop sequence dwh_manage.job_instance_id;
create sequence dwh_manage.seq_job_instance_id start with 1;

--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_type varchar(20),
        counter_value integer not null);

create index job_instance_counters_idx on dwh_manage.job_instance_counters (job_instance_id,
counter_name);

```

Exasol

```

-- drop table dwh_manage.job_instance_status;
create table dwh_manage.job_instance_status (
    job_instance_id bigint identity primary key,
    process_instance_id bigint,
    process_instance_name varchar(255),
    job_name varchar(255) not null,
    job_project varchar(512) UTF8,
    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 dwh_manage.job_instance_counters (
    job_instance_id bigint not null,
    counter_name varchar(128) not null,
    counter_type varchar(20),
    counter_value integer not null);

```