**Interface Control System – Stream Documentation**

1. **Stream Configuration**  
   A stream defines a hierarchy of tasks that each execute one or more events that must be completed before the task is completed. The stream hierarchy supports two task types.  
   1. **Execution Task**  
      An execution task defines one or more events that must be completed before the task is set to complete. Once a task is completed, the direct descendent child tasks are opened for execution, and this process is repeated until the stream task hierarchy has been completely processed. An execution task parent can be one of the following; the stream root, that is, the task is opened when the stream is opened, an execution task, that is, the task is opened when the parent task is complete, a gate task, that is, the task is opened when the gate task dependencies have all been satisfied.
   2. **Gate Task**  
      A gate task defines one or more dependant execution tasks that must be completed before the gate task is opened. Once a gate task is opened, the direct descendent child execution tasks are opened for execution and this process is repeated until the stream task hierarchy has been completely processed from the task gate down. A gate task parent can only be the stream root, that is, a gate task cannot be nested. The gate task provides the stream hierarchy with cross branch dependency.

Multiple stream execution tasks can be open at the same time for the one stream execution instance when there is more than one execution child for the stream hierarchy root, a completed execution task, or an opened gate task. When multiple stream execution tasks are open at the same time they are considered to be executing in parallel, that is, the events for all opened execution tasks are queued for execution and therefore also considered to be executing in parallel. In order to achieve serialization of events (application functionality), the serial events need to be placed in their own tasks that are chained in the stream hierarchy in the desired sequence.

While the events of an open execution task are considered to be executing in parallel, in practice, the following locking rules and resource restrictions will restrict the level of parallel processing that can be achieved.

* 1. **Event Lock**  
     The event lock defines a locking string that is enforced across all executing stream instances to prevent deadlocks and data corruption. An event lock of \*NONE will automatically be converted into a unique lock for the executing event so that no lock contention is enforced. The stream poller ensures that the event lock is honoured globally across all opened stream instances for event execution, such that, when multiple queued events exist, regardless of the stream instance, with the same lock string, the queued events will remain in a queued status until such time as the lock becomes available at the completion of the event currently holding the lock. Queued events waiting on the same lock are processed sequentially; by stream instance sequence number (effectively submit time) and hierarchy sequence, as the lock becomes available.
  2. **Job Group**

The event is executed from the LICS\_STREAM\_PROCESSOR package using the ICS trigger processing functionality. That is, the execution is moved to a separate processing job which allows for a level of parallel processing to be achieved. The event is assigned to a job group that is attached to a number of ICS processing jobs. The number of ICS processing jobs defined for the job group will determine the level of parallel processing achieved at any point in time. For example, at a point time there are 4 opened events all pointing to the same job group and there are 6 jobs defined for the job group, in this instance, all 4 events will execute at the same time. However, if at a point time there are 10 opened events all pointing to the same job group and there are 6 jobs defined for the job group, then only 6 events will execute at the same time while the remaining 4 events will wait for a job slot to become available.

The stream definition allows for several substitution patterns that provide for property refinement at execution time.

* 1. **Supplied Procedure**  
     The supplied procedure allows for an execution time procedure string to be substituted for an event execution procedure at execution time. The special value \*SUPPLIED in the event execution procedure will forced the stream execution loader to substitute the supplied procedure string into the event definition at load time. An example of this technique is used in the CDW Invoice Summary interface monitor package that invokes the CDW Sales Aggregation stream.
  2. **Parameter Substitution**Parameter substitution can be applied to an event lock, procedure and job group. The parameter value is substituted into the event value where a match is found using the syntax <PARAMETER\_CODE>. For example, the Integrated Planning load streams use the MOE parameter substitution in the event lock so that the same stream can be used for all current and future MOE files without a need to change the stream (e.g. DF\_DEMAND\_FINAL\_<MOE>).

1. **Stream Execution**  
     
   Streams are executed using one of two methods; submitted from a stored procedure or manually submitted from the ICS website. The stored procedure LICS\_STREAM\_LOADER is used to submit a stream for execution. The stream loader creates a unique stream sequence number and copies the current selected stream information to the stream execution tables with the unique stream sequence number. This means that a stream can be submitted multiple times for execution with each execution instance processing in isolation. Each execution instance retains the stream structure as at the time of submission so it is possible to have multiple execution instances of the same stream with different stream structures if the stream structure was modified between the submissions of each execution instance. Stream execution instances are serialized by the unique stream code so that only one execution instance of a particular stream will execute at any one time. Subsequent execution instances of the same stream are queued until the previous instance has been either completed or cancelled. Paused execution instances will also block subsequent instances.  
     
   A stream instance can have one of the following statuses.  
   * + **\*PENDING** – the loaded stream instance is pending execution. The stream instance will remain in this status until all earlier instances of the same stream code have been completed or cancelled.
     + **\*OPENED** – the stream instance is opened and executing. Opened stream instances block later instances of the same stream code, leaving them in a pending status.
     + **\*COMPLETED** – the stream instance has been successfully completed.
     + **\*OPNPAUSED** – the stream instance has been paused. Currently opened events will continue to execute but no additional tasks or events will be processed until the stream has been resumed. Opened paused stream instances continue to block later instances of the same stream code, leaving them in a pending status.
     + **\*OPNCANCEL** – the stream instance has been requested to cancel. Currently opened events will continue to execute but all pending tasks and events will be cancelled. Opened cancelling stream instances continue to block later instances of the same stream code, leaving them in a pending status.
     + **\*CANCELLED** – the stream instance has been cancelled. The status can be the result of a cancel request or a failed event that will cancel the remainder to the stream.

A stream instance task can have one of the following statuses.

* + - **\*PENDING** – the task is pending execution. The task will remain in this status until the parent task has been successfully completed or the parent gate has been opened.
    - **\*OPENED** – the task is opened and executing.
    - **\*COMPLETED** – the task has been successfully completed.
    - **\*FAILED** – the task has failed. The task will move to a failed status when one or more of the task events have failed.
    - **\*CANCELLED** – the task has been cancelled.

A stream instance gate can have one of the following statuses.

* + - **\*PENDING** – the gate is pending execution. The gate will remain in this status until the gate depend tasks have all been successfully completed.
    - **\*COMPLETED** – the task gate has been opened.
    - **\*CANCELLED** – the gate has been cancelled.

A stream instance event can have one of the following statuses.

* + - **\*PENDING** – the event is pending execution. The event will remain in this status until the parent task has been has been opened.
    - **\*QUEUED** – the event is queued and waiting to be opened. The event will only be opened when the event lock becomes available.
    - **\*OPENED** – the event is opened on a job queue.
    - **\*WORKING** – the event is executing.
    - **\*COMPLETED** – the event has been successfully completed.
    - **\*FAILED** – the event has failed.
    - **\*CANCELLED** – the event has been cancelled.

An opened stream can be managed as follows:

* 1. **Cancel stream execution instance**

A stream execution instance can be cancelled from the ICS website. The cancel request is logged against the stream and will be processed by the stream poller. If the stream is in a pending status then the stream and all child tasks and events are set to a cancelled status, that is , the stream execution instance will never execute. If the stream is in an opened status then the stream will be set to an opened cancelling status, all pending tasks set to a cancelled status, and all pending or queued events set to a cancelled status. All opened tasks and events will be allowed to complete or fail before the stream status is set to cancelled. Until the current execution is cancelled all subsequent instances of the same stream code will be blocked, that is, remain in a pending status.

* 1. **Pause stream execution instance**

An opened stream execution instance can be paused from the ICS website. The pause request is logged against the stream and will be processed by the stream poller. The stream is set to an open paused status.

* 1. **Resume stream execution instance**

An opened paused stream execution instance can be resumed from the ICS website. The pause request is logged against the stream and will be processed by the stream poller. The stream is set to an open status.

1. **Stream Loading - LICS\_STREAM\_LOADER**  
     
   Streams are submitted for execution using one of two methods; submitted from a stored procedure or manually submitted from the ICS website. The stored procedure LICS\_STREAM\_LOADER is used to submit a stream for execution. The stream loader creates a unique stream sequence number and copies the current selected stream information to the stream execution tables with the unique stream sequence number. This means that a stream can be submitted multiple times for execution with each execution instance processing in isolation. Each execution instance retains the stream structure as at the time of submission so it is possible to have multiple execution instances of the same stream with different stream structures if the stream structure was modified between the submissions of each execution instance. Stream execution instances are serialized by the unique stream code so that only one execution instance of a particular stream will execute at any one time. Subsequent execution instances of the same stream are queued until the previous instance has been either completed or cancelled. Paused execution instances will also block subsequent instances.  
     
   Stream can be submitted from a stored procedure using the following pattern:

lics\_stream\_loader.load('STREAM\_CODE', ‘Execution Text’, ‘\*SUPPLIED Procedure’);

lics\_stream\_loader.set\_parameter('PARAMETER\_CODE', ‘PARAMETER\_VALUE’);

lics\_stream\_loader.execute;

Where the following rules apply:

* The \*SUPPLIED procedure can be either NULL or a stored procedure execution string that will be substituted in any events that have \*SUPPLIED specified for the execution procedure.
* The set parameter method must be executed for each stream parameter that has \*SUPPLIED specified for the parameter value.

1. **Stream Polling - LICS\_STREAM\_POLLER**

The stream poller is the control package that manages the execution of all stream instances. This package utilises the ICS polling functionality (job type \*POLLER) that executes the stream poller at 10 second intervals. Only one instance of this package executes for the entire ICS instance as it performs the management of all stream instances for the ICS instance to enable the queuing of multiple instances of the same stream and enforcement of cross stream event locking. The package performs the following management logic in this sequence.

* 1. **Process stream requests**  
       
     Retrieves all stream instances with a status of pending, opened or opened paused (\*PENDING, \*OPENED, \*OPNPAUSED) that have an outstanding request. The stream instance request is updated from the website and can be; \*CANCEL – a request to cancel the pending or opened stream instance, \*PAUSE – a request to pause an opened stream instance, \*RESUME – a request to resume a paused stream instance.  
       
     The cancel request locks the stream instance and ensures that the status is still valid for the cancel request. Cancelling a pending stream instance will set the stream instance, along with all child tasks and events, to a cancelled status (\*CANCELLED). Cancelling an opened stream instance will set the stream instance to an opened cancel status (\*OPNCANCEL), all child tasks with a pending status (\*PENDING) to a cancelled status (\*CANCELLED), and all events with a pending or queued status (\*PENDING or \*QUEUED) to a cancelled status (\*CANCELLED). Any currently opened tasks and events will continue to process until end. The event procedure can call the LICS\_STREAM\_PROCESSOR.CALLBACK\_IS\_CANCELLED function at logical points to test if the parent stream instance is in an opened cancelled status (\*OPNCANCEL) and shutdown the procedure in a way that is logical for the application.  
       
     The pause request locks the stream instance and ensures that the status is still valid for the pause request. Pausing an opened stream instance will set the stream instance to an opened paused status (\*OPNPAUSED). Any currently opened events will continue to process until end. The stream instance will be ignored in the subsequent open stream processing but will still block pending instances of the same stream from opening.  
       
     The resume request locks the stream instance and ensures that the status is still valid for the resume request. Resuming a paused stream instance will set the stream instance to an opened status (\*OPENED). The stream will then participate in the subsequent open stream processing.
  2. **Process open streams**  
       
     Retrieves all stream instances with a status of opened or opened cancel (\*OPENED or \*OPNCANCEL) and updates the status of the stream instance and opened child tasks as follows.  
     1. The task status is set to completed (\*COMPLETED) when all child events have successfully completed. All direct child tasks are set to opened (\*OPENED) and related events set to queued (\*QUEUED) to make the events available for execution.
     2. The task status is set to failed (\*FAILED) when any child events have failed. All child tasks and events are then cancelled recursively to the end of the current hierarchy branch. Any task gates and their children that are dependent on the failed task are also cancelled.
     3. The stream instance is set to completed (\*COMPLETED) when all child tasks have been completed.
  3. **Open pending streams**  
       
     Retrieves all stream instances with a status of pending (\*PENDING), that do not have a currently open instance of the same stream code, and sets the status to open (\*OPENED).
  4. **Submit stream events**Retrieves all stream instance events a status of queued (\*QUEUED) honouring the event lock string. Queues events where the lock is available are set to opened (\*OPENED) and the event procedure is triggered for execution using the ICS trigger functionality and the LICS\_STREAM\_PROCESSOR.

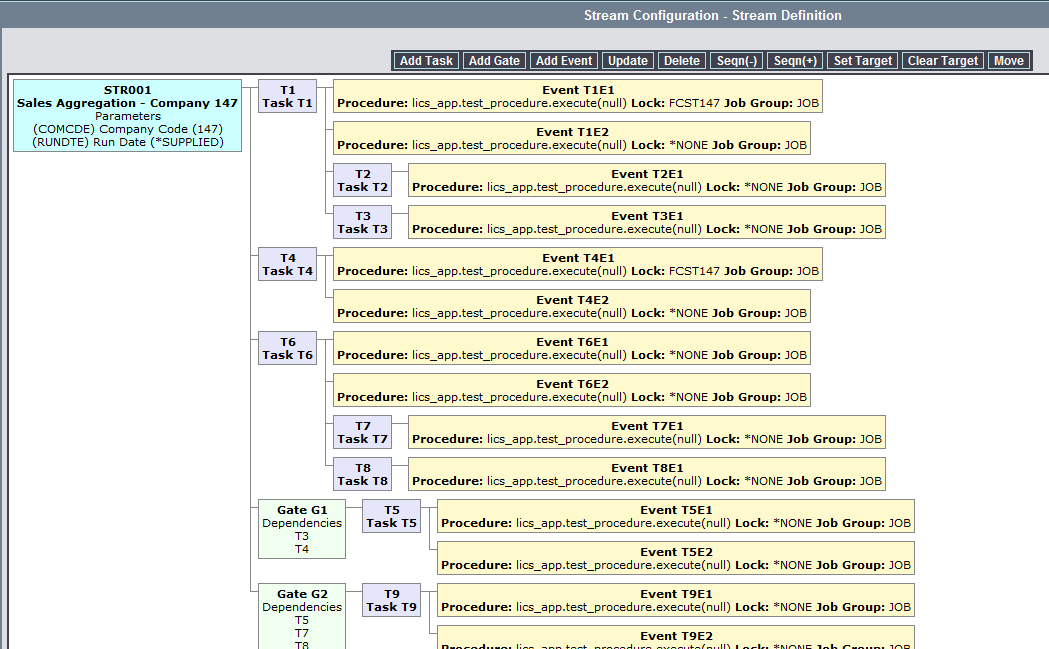
1. **Stream Event Processing - LICS\_STREAM\_PROCESSOR**  
     
   The stream event processor is a wrapper package that executes and provides call back functionality to the stream event procedure. The package executes the stream event procedure and updates the stream event with either a completed (\*COMPLETED) status or a failed (\*FAILED) status along with the stream event procedure exception text. While the stream event procedure is executing it can use the call back functionality to retrieve information about the event. The call back functionality can be invoked by making calls to the following functions from the stream event procedure code (effectively a function call to the invoking parent procedure).  
   1. **Event Function** - LICS\_STREAM\_PROCESSOR.CALLBACK\_EVENT **-** This function returns the event code for the currently executing event. For use in logging or exception handling.
   2. **Text Function** - LICS\_STREAM\_PROCESSOR.CALLBACK\_TEXT - This function returns the event text for the currently executing event. For use in logging or exception handling.
   3. **Lock Function** - LICS\_STREAM\_PROCESSOR.CALLBACK\_LOCK - This function returns the event lock string for the currently executing event. For use in lock handling (LICS\_LOCKING). Note that while the stream poller ensures that the event lock is honoured globally across all opened stream instances for event execution, it is the responsibility of the event procedure to implement the lock handling using the LICS\_LOCKING functionality.
   4. **Alert Function** - LICS\_STREAM\_PROCESSOR.CALLBACK\_ALERT - This function returns the event alert string for the currently executing event. For use in exception handling.
   5. **Email Function** - LICS\_STREAM\_PROCESSOR.CALLBACK\_EMAIL **-** This function returns the event email string for the currently executing event. For use in exception handling.
   6. **Parameter Function** - LICS\_STREAM\_PROCESSOR.CALLBACK\_PARAMETER **-** This function returns the stream parameter value for the requested parameter code.
   7. **Is Cancelled Function** (LICS\_STREAM\_PROCESSOR.CALLBACK\_IS\_CANCELLED) **-** This function returns the stream cancelled indicator. Can be used at logical points to test if the parent stream instance is in an opened cancelled status (\*OPNCANCEL) allowing shutdown of the procedure in a way that is logical for the application.

**Interface Control System – Stream Configuration**

Stream configuration allows a user to create and modify stream definitions. Once created or modified, the stream is available for execution. Each execution retains a snapshot of the stream definition as it existed at the time of submission. The stream snapshot is used to paint the stream execution review.

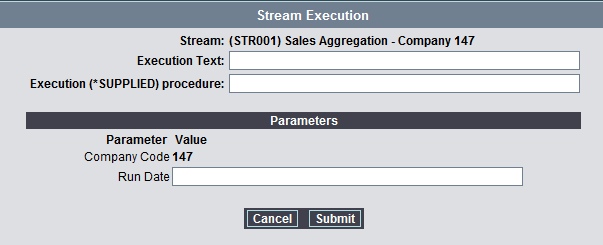
All changes are retained in memory until the user presses the Accept button at the bottom of the Stream Definition screen. Accept replaces the stream definition on the database and Cancel ignores the changes in memory.

Tasks, gates and events can be re-sequenced at the same hierarchy level using the Seqn + and - buttons. Tasks and events can be moved between hierarchy levels using the Set Target and Move buttons.



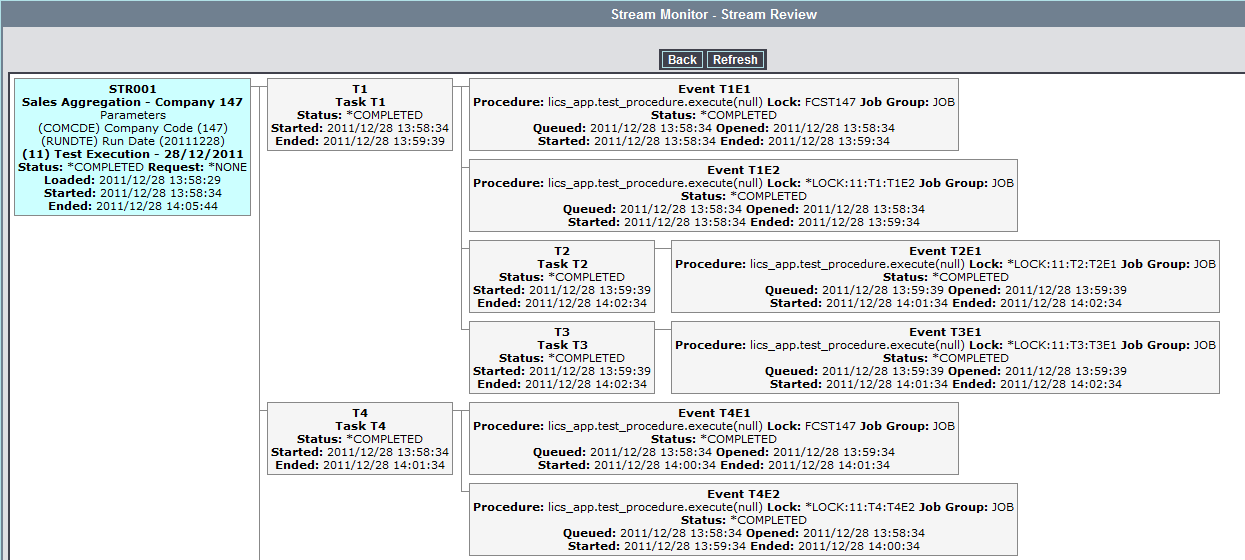
**Interface Control System – Stream Execution**

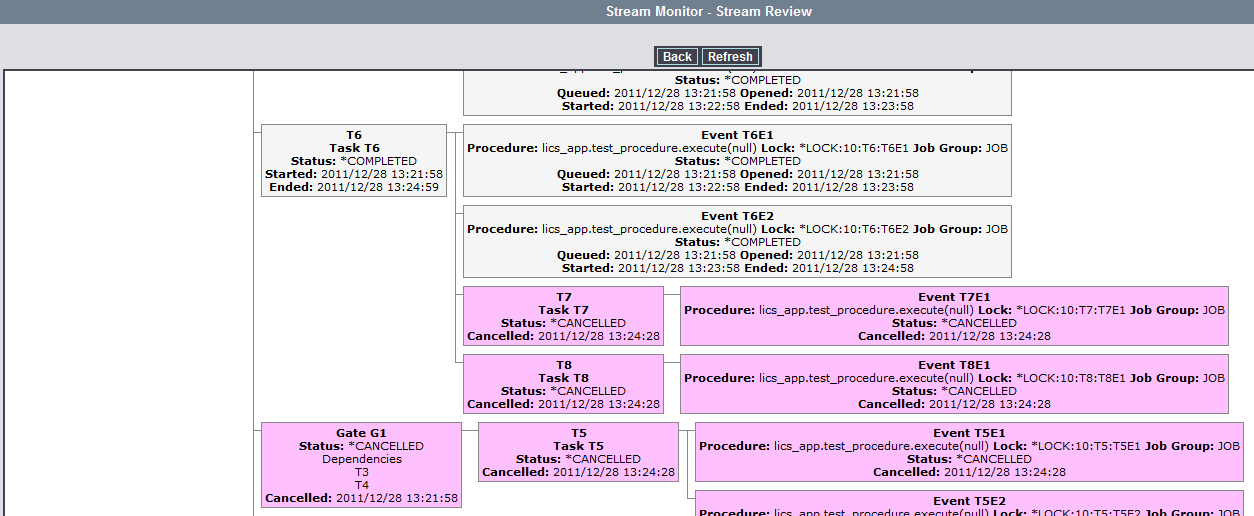
Stream execution allows the user to manually submit a stream for execution. The user must supply the execution text and parameter values for all \*SUPPLIED parameters. The execution procedure is optional and is only required where the event procedure special value \*SUPPLIED has been used.

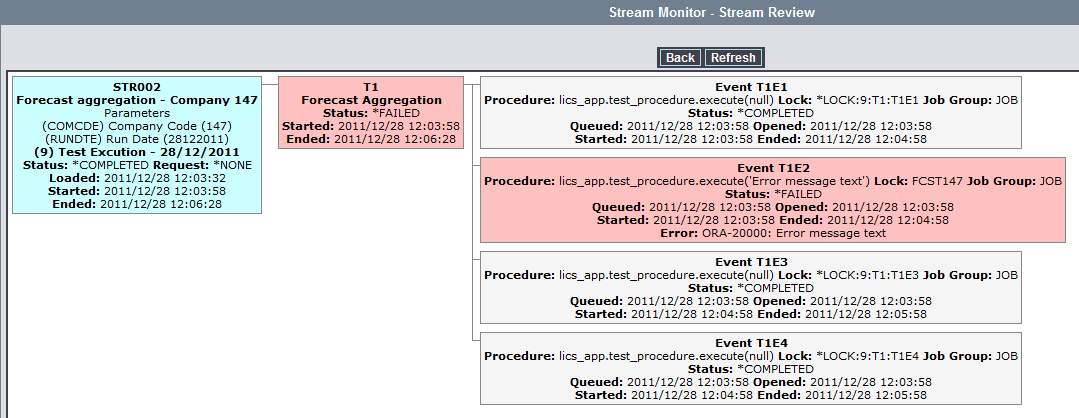


**Interface Control System – Stream Monitor**

Stream monitor allows the user to review a stream execution during and after completion of the execution. The user can also cancel, pause and resume a stream execution from this menu option.

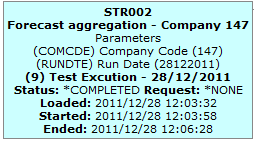






The stream review uses the following colour coding to show the various processing and results of a stream.

**Stream**

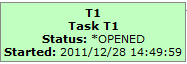


**Task**

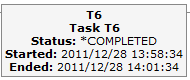
Task Pending



Task Opened



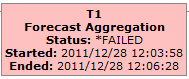
Task Completed



Task Cancelled

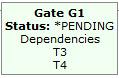


Task Failed

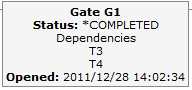


**Gate**

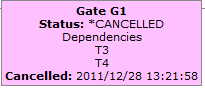
Gate Pending



Gate Completed



Gate Cancelled

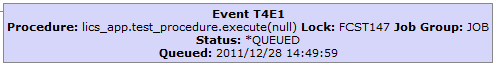


**Event**

Event Pending



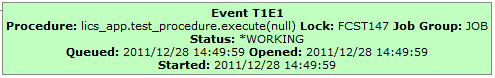
Event Queued



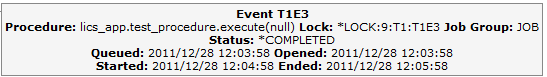
Event Opened



Event Working



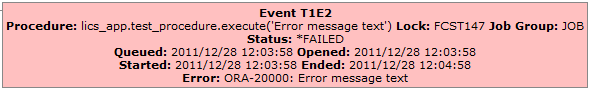
Event Completed



Event Cancelled



Event Failed



**Interface Control System – Stream Data Entity Diagram**

**Stream Definition Tables**

**Stream Execution Tables/Sequence**

LICS\_STR\_HEADER

LICS\_STR\_TASK

LICS\_STR\_EVENT

LICS\_STR\_DEPEND

LICS\_STR\_PARAM

**LICS\_STREAM\_SEQUENCE**

Prefixed to execution tables primary key

LICS\_STR\_EXE\_HEADER

LICS\_STR\_EXE\_TASK

LICS\_STR\_EXE\_EVENT

LICS\_STR\_EXE\_DEPEND

LICS\_STR\_EXE\_PARAM

**Interface Control System – Stream and Triggered Overview Diagram**

**LICS and LICS\_APP Schemas**

**SERVER**

**ICS – Interface Control System**

**ORACLE**

**Local Application Schemas (LADS, BDS, etc.)**

Triggered Procedure

Stream Invocation

Procedure

Schema

Tables

**Stream and Triggered Processing**

Damon

Processors

Parallel

LICS\_TRIGGERED

LICS\_STR\_EXE\_HEADER

Stream Poller

Serial

LICS\_STR\_HEADER

Stream Loader

Stream Configuration

Trigger Loader

**Interface Control System – Stream Configuration and Processing**

**Configuration**

Task T1

Task T2

Event T1E1

Event T1E2

Task T3

Event T2E1

Event T3E1

Task T4

Event T4E1

Event T4E2

Stream

Gate G1

Task T5

Event

Event

Gate Dependencies

T2, T4

Task T6

Task T7

Event

Event

Task T8

Event

Event

Gate G2

Task T9

Event

Event

Gate Dependencies

T3, T5, T7

**Interface Control System – Stream Configuration and Processing**

**Processing**

**LICS\_STREAM\_LOADER**

1. Supports the creation of runtime substitution parameters.
2. Creates a new stream action sequence and creates new LICS\_STR\_ACTION rows for each event in the selected stream header based on the current stream configuration data.

**Configuration**

Logical Job Group

**STR**

**Daemon Processing Job S1** – Job Group = **STR**#01

**Job Procedure**

lics\_trigger\_processor.execute\_from\_daemon

**LICS\_TRIGGERED\_PROCESSOR**

1. Wakes up.
2. Retrieves all LICS\_TRIGGERED rows for the logical job group.
   1. Attempts to lock the LICS\_TRIGGERED row.
   2. When LICS\_TRIGGERED row locked executes the trigger procedure.
   3. Deletes the LICS\_TRIGGERED row.
3. Sleeps.

LICS\_TRIGGERED

Schedule LICS\_STREAM\_POLLER as a \*POLLER job with an execution interval of 30 seconds.

Wakes up all related jobs for the Processing Group

(Where matches Job Group up to the parallel marker #)

**Daemon Processing Job S1** – Job Group = **STR**#02

**Job Procedure**

lics\_trigger\_processor.execute\_from\_daemon

**Daemon Processing Job S1** – Job Group = **STR**#03

**Job Procedure**

lics\_trigger\_processor.execute\_from\_daemon

**LICS\_TRIGGER\_LOADER**

1. Creates a new LICS\_TRIGGERED row using the stream action event configuration.
2. Wakes up the associated processing jobs using the Job Group from the stream action event configuration.

Stream Header

LICS\_STR\_EXE\_HEADER

Stream Task

Stream Task

Stream Event

Stream Event

Stream Task

Stream Event

Stream Event

Stream Task

Stream Event

Stream Event

**LICS\_STREAM\_POLLER**

1. Retrieve the list of active streams and controls the event processing based on the stream task hierarchy and lock information.
2. Executes the LICS\_TRIGGERED\_LOADER for each event that is ready for processing using the stream action information, that is, the trigger procedure becomes the stream action event procedure.

Scheduled job or stored procedure executes LICS\_STREAM\_LOADER with the requested stream header identifier.